P.R.ENGINEERING COLLEGE

VALLAM, THANJAVUR

DEPARTMENT OF IT

SEMESTER IV

IT2251 – SOFTWARE ENGINEERING AND QUALITY ASSURANCE

PREPARED BY

S.PREMKUMAR /AP/ IT

IT2251 – SOFTWARE ENGINEERING AND QUALITY ASSURANCE

LTPC

3003

UNIT I SOFTWARE PRODUCT AND PROCESS 9

Introduction – S/W Engineering paradigm – Verification – Validation – Life cycle models – System engineering – Computer based system – Business process engineering overview – Product engineering overview.

UNIT II SOFTWARE REQUIREMENTS 9

Functional and non-functional – Software document – Requirement engineering process – Feasibility studies – Software prototyping – Prototyping in the software process – Data – Functional and behavioral models – Structured analysis and data dictionary.

UNIT III ANALYSIS, DESIGN CONCEPTS AND PRINCIPLES 9

Systems engineering – Analysis concepts – Design process and concepts – Modular design – Design heuristic – Architectural design – Data design – User interface design – Real time software design – System design – Real time executives – Data acquisition system – Monitoring and control system.

UNIT IV TESTING 9

Taxonomy of software testing – Types of S/W test – Black box testing – Testing boundary conditions – Structural testing – Test coverage criteria based on data flow mechanisms – Regression testing – Unit testing – Integration testing – Validation testing – System testing and debugging – Software implementation techniques.

UNIT V SOFTWARE QUALITY ASSURANCE 9

Process and product quality – Quality assurance and standards – Quality planning and control – Software metrics – Process improvement – Software configuration management.

TEXT BOOKS

1. Ian Sommerville, "Software Engineering", 7th Edition, Pearson Education, 2007.

2. Pressman, R.S., "Software Engineering - A Practitioner's Approach", 6th Edition, McGraw-Hill International Edition, 2005.

REFERENCES

1. Humphrey, W.S., "A Discipline for Software Engineering", Pearson Education, 2007.

2. Peters, J.F. and WitoldPedrycz, "Software Engineering - An Engineering Approach", Wiley-India Pvt. Ltd., 2007.

3. Schach, S.R., "Software Engineering", Tata McGraw-Hill Publishing Company Limited, 2007.

Unit – I

Software Process

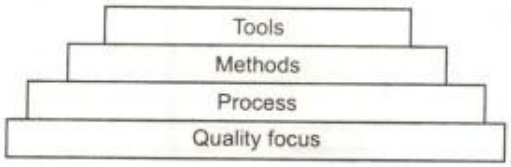
1.1 Introduction

Software engineering is the establishment and sound engineering principles applied to obtain reliable and efficient software in an economical manner

Software engineering includes process, management techniques, technical methods, and the use of tool. While building any software, the software process provides the interaction between user and developer. In this chapter we will understand the basic concept of process and process models. We will discuss how to use various life cycle models for building qualitative software in an economic manner. Finally we will focus on system engineering.

1.1.1 Layered Technology

- Software engineering is a layered technology. Any software can be developed using these .layered approaches. Various layers on which the technology is based are quality focus layer, process layer, methods layer, tools layer.



- A disciplined quality management is a backbone of software engineering technology.
- Process layer is a foundation of software engineering. Basically, process defines the framework for timely delivery of software.
- In method layer the actual method of implementation is carried out with the help of requirement analysis, designing, coding using desired programming constructs and testing.
- Software tools are used to bring automation in software development process.

Thus software engineering is a combination of process, methods, and tools for development of quality software.

1.2 Software Process

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

The fundamental activities are

- Specification
- Design and implementation
- Validation
- Evolution

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

1.2.1 Common Process Framework

The process framework is required for representing the common process activities. It is as shown below.

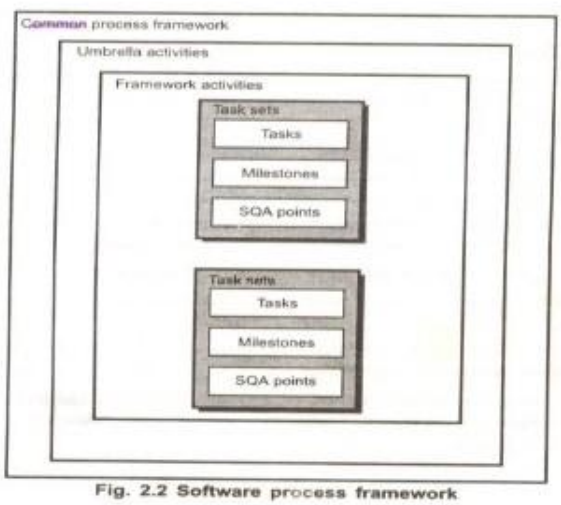


Fig. 2.2 Software process framework

As shown in figure the software process is characterized by process framework activities, task sets and umbrella activities.

Process Framework Activities

- Communication
    - By communicating customer requirement gathering is done.
- Planning - Establishes engineering work plan, describes technical risks, lists resource requirements, work products produced, and defines work schedule.
- Modeling - The software model is prepared by:
    - Analysis of requirements
    - Design
- Construction ~ The software design is mapped into a code by:
    - Code generation.
    - Testing
- Deployment - The software delivered for customer evaluation and feedback is obtained.

Task Sets - The task set defines the actual work done in order to achieve the software objective. The task set is used to adopt the framework activities and project team requirements using

Collection of software engineering work tasks

Project milestones

Software quality assurance points

Umbrella activities - The umbrella activities occur throughout the process. They focus on project management, tracking and control. The umbrella activities are

1. Software project tracking and control - This is an activity in which software team can assess progress and take corrective action to maintain schedule.

2. Risk management- The risks that may affect project outcomes or quality can be analyzed.

3. Software quality assurance - These are activities required to maintain software quality.

4. Formal technical reviews - It is required to assess engineering work products to uncover and remove errors before they propagate to next activity.

5. Software configuration management - Managing of configuration process when any change in the software occurs.

6. Work product preparation and production - The activities to create models, documents, logs, forms, and lists are carried out.

7. Reusability management - It defines criteria for work product reuse.

8. Measurement - In this activity, the process can be defined and collected. Also project and product measures are used to assist the software team in delivering the required software.

1.2.2 Capability Maturity Model (CMM)

The Software Engineering Institute (SEI) has developed a comprehensive process meta-model emphasizing process maturity. It is predicated on a set of system and software capabilities that should be present when organizations reach different levels of process capability and maturity .

' The Capability Maturity Model (CMM) is used in assessing how well an organization's processes allow to complete and manage new software projects.

Various process maturity levels are

Level 1 : Initial - Few processes are defined and individual efforts are taken.

Level 2 : Repeatable - To track cost schedule and functionality basic project management processes are established .Depending on earlier successes of projects with similar applications necessary process discipline can be repeated.

Level 3 : Defined - The process is standardized, documented and followed. All the projects use documented and approved version of software process which is useful in developing and supporting software.

Level 4 : Managed - Both the software process and product are quantitatively understood and controlled using detailed measures.

Level 5 : Optimizing - Establish mechanisms to plan and implement change. Innovative ideas and technologies can be tested.

Thus CMM is used for improving the software project.

1.3 Software Engineering Paradigm

Software engineering paradigm or process model IS an abstract representation of a process.

The process model is chosen based on nature of software project and application and then to obtain deliverable product method and tools are applied.

Using problem solving loop the software development can be done. The problem solving loop includes.

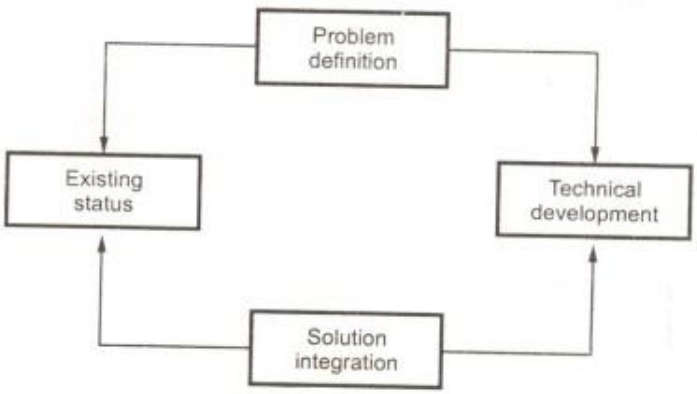


Fig. 2.3 Problem solving loop

The **existing status** that represents current state of affairs.

In the **problem identification phase** particular problem is identified.

The **technical development stage** is for solving the identified problem using an appropriate technology.

Finally **solution integration** is responsible for delivering the results.

But applying such problem solving loop in software development process is very difficult because we can not strictly categorize the development in these phases. There may be a requirement of cross talk within and across stages. Hence some software process models are suggested depending upon nature of software. Such models are called generic software models.

Generic software process models are

- The Waterfall Model – Separate and distinct phases of specification and development.
- Prototyping Model – A quick design approach is adopted.
- Incremental Models – It emphasizes on short development cycle.
  - Rapid Application and Development (RAD) Model
- Evolutionary Process Models – Specification, development and validation are interleaved.
  - Incremental Model
  - Spiral Model
  - WIN-WIN spiral model
  - Concurrent Development

1.4 Life Cycle Models

A life cycle is the sequence in which a project specifies, prototypes, designs, implements, tests, and maintains a piece of software. In software engineering, the life cycle model depicts various stages of software development process. Using life cycle model various development issues can be solved at the appropriate time.

1.4.1 Waterfall Model

- The waterfall model is also called as classic life cycle model. It is the oldest software paradigm. It follows the sequential approach to software development process.
- In this kind of modeling, the software development process starts with communication phase. In communication phase the requirement gathering is done by communicating with the customer. The software requirement specification (SRS) is prepared.

- The second phase is planning. In planning phase we schedule the project and also perform some primitive estimation of project. The software project plan is prepared.
- Modeling is the next phase in which requirement analysis is done and we Simply design our software. Various models such as data flow diagram, ERD are prepared.
- The construction phase comes after modeling. In construction phase actual implementation is performed. We do actual coding with the help ~f suitable programming language After preparing the code, we test it with all possible input set. And complete software has to be prepared which can be delivered to the customer.
- Finally deployment is carried out. In deployment phase we actually deliver the software and give the necessary support to the customer. Frequent feedback from the customer is to be taken in order to maintain the software.

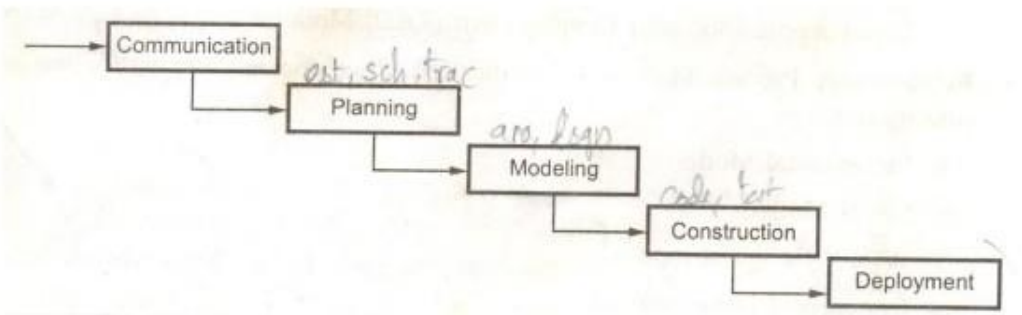


Fig. 2.4 Waterfall model

Drawbacks of waterfall model

There are some problems that are encountered if we apply the waterfall model and those are

1. It is difficult to follow the sequential flow in software development process. If some changes are made at some phases then it may cause some confusion.
2. The requirement analysis is done initially, and sometimes it is not possible to state all the requirements explicitly in the beginning. This causes difficulty in the project.
3. The customer can see the working model of the project only at the end. After reviewing of the working model; if the customer gets dissatisfied then it causes serious problems.

4. Linear nature of waterfall model induces blocking states, because certain tasks may be dependant on some previous tasks. Hence it is necessary to accomplish all the dependant tasks first. It may cause long waiting time.

1.4.2 Incremental Model

- The incremental model has same phases that are in waterfall model. But it is iterative in nature. The incremental model has following phases.

1. Analysis

2. Design

3. Code

4. Test

- The incremental model delivers series of releases to the customer. These releases are called increments. More and more functionality is associated with each increment.
- The first increment is called core product. In this release the basic requirements are implemented and then in subsequent increments new requirements are added.
- The word processing software package can be considered as an example of incremental model. In the first increment only the document processing facilities are available. In the second increment, more sophisticated document producing and processing facilities, file management functionalities are given. In the next increment spelling and grammar checking facilities can be given. Thus in incremental model progressive functionalities are obtained with each release.

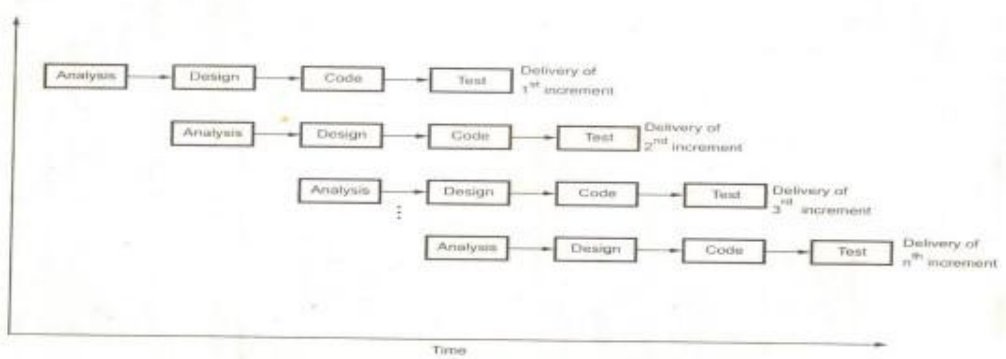


Fig. 2.5 The incremental model

When to choose it?

1. When requirements are reasonably well-defined.
2. When overall scope of the development effort suggests a purely linear effort.
3. When limited set of software functionality needed quickly.

Merits of incremental model

1. The incremental model can be adopted when there are less number of people involved in the project.
2. Technical risks can be managed with each increment.
3. For a very small time span, at least core product can be delivered to the customer.

1.4.2.1 Rapid Application Development (RAD) Model

- The rapid application development model is type of incremental software process model in which there is extremely short development cycle.
- This model is similar to waterfall model which achieves the high speed development using compoi1ent based construction.
- To develop the fully functional system within short time period using this model it is necessary to understand the requirements fully and to have a restricted project scope.

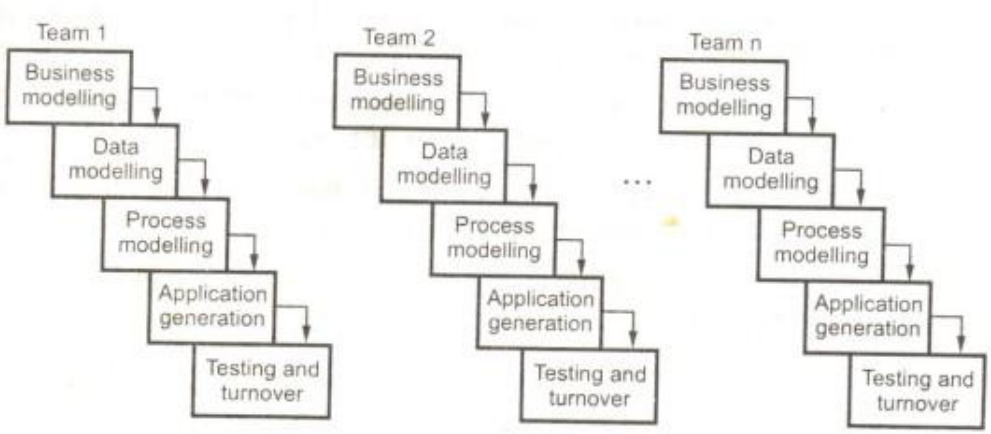


Fig. 2.6 Rapid application development model

- Various phases of RAD model are

1) **Business modeling** - In business modeling, the information flow is modeled into various business functions. These business functions collect following information.

- Information that drives the business process.
- The type of information being generated.
- The generator of information.
- The information flow.
- The processor of information.

2) **Data modeling** - In this phase the information obtained in business model' classified into data objects. The characteristics of data objects (attributes) are identified. The relationship among various data objects is defined.

3) **Process modeling** - In this phase the data objects are transformed into processes. These processes are to extract the information from data objects and are responsible for implementing business functions.

4) **Application generation** - For creating software various automation tools can be used. RAD also makes use of reusable components or creates reusable components to have rapid development of software.

5) **Testing and turnover** - As RAD uses reusable components the testing efforts are reduced. But if new components are added in software develop men process then such components need to be tested. It is equally important to test all the interfaces.

1.4.3 Spiral Model

- This model possesses the iterative nature of prototyping model and controlled and systematic approaches of the linear sequential model.

- This model gives efficient development of incremental versions of software. In this model, the software is developed in series of increments.

- The spiral model is divided into a number of framework activities. These framework activities are denoted by task regions.
- Usually there are six tasks regions. The spiral model is as shown in Fig.

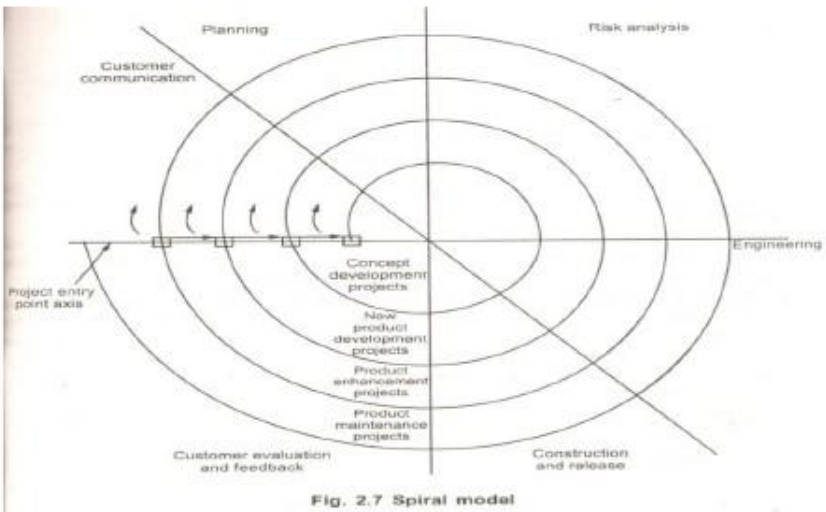


Fig. 2.7 Spiral model

- In the initial pass, product specification is built and in subsequent passes around the spiral the prototype gets developed and then more improved versions of software gets developed.
- During planning phase, the cost and schedule of software can be planned and adjusted based on feedback obtained from customer evaluation.
- In spiral model, project entry point axis is defined. This axis represents starting point for different types of projects.

For instance concept development project will start at core of spiral and will continue along the spiral path. If the concept has to be developed into actual project at entry point 2 the product development process starts. Hence entry point 2 is called product development project entry point. The development of the project can be carried out in iterations.

- The task regions can be described as

i. Customer communication - In this region it is suggested to establish customer communication.

ii. Planning - All planning activities are carried out in order to define resources time-line and other project related activities.

iii. Risk analysis - The tasks required to calculate technical and management risks are carried out.

iv. Engineering - In this task region, tasks required to build one or more representations of applications are carried out.

v. Construct and release - All the necessary tasks required to construct, test, install the application are conducted. Some tasks that are required to provide user support are also carried out in this task region.

vi. Customer evaluation - Customer's feedback is obtained and based on customer evaluation required tasks are performed and implemented at installation stage.

- In each region, numbers of work tasks are carried out depending upon the characteristics of project. For a small project relatively small number of work tasks is adopted but for a complex project large number of work tasks can be carried out.
- In spiral model, the software engineering team moves around the spiral in a clockwise direction beginning at the core.

Drawbacks of spiral model

- It is based on customer communication. If the communication is not proper then the software product that gets developed will not be the up to the mark.
- It demands considerable risk assessment. If the risk assessment is done properly then only the successful product can be obtained.

1.4.4 WIN-WIN Spiral Model

- As in spiral model the customer communication is important for obtaining the requirements of the project, the WIN-WIN model also suggests proper communication with customer. In reality customer and developers undergo through the process of

negotiation. Successful negotiation occurs when both the sides win. This is called win-win' result.

- Customer's win means - obtaining the system that satisfies most of the needs.
- Developer's win means - getting the work done with realistic and achievable budgets and deadlines.
- In WIN-WIN spiral model negotiation activities are carried out at the beginning of each pass of the spiral.
- Various activities that can be carried out in WIN-WIN spiral model are

1. Identification of 'stakeholders'.
2. Determination of „stakeholders" wins condition.
3. Negotiations of stakeholders striving for win condition. With the concerned software project team reconcile for win-win result. Then determine next level objectives, constraints and alternatives.
4. Evaluate process and product. Analyze and resolve the risks.
5. Define next level of product and process.
6. Validate process and product definitions.
7. Take a review of product and give necessary comments on it.

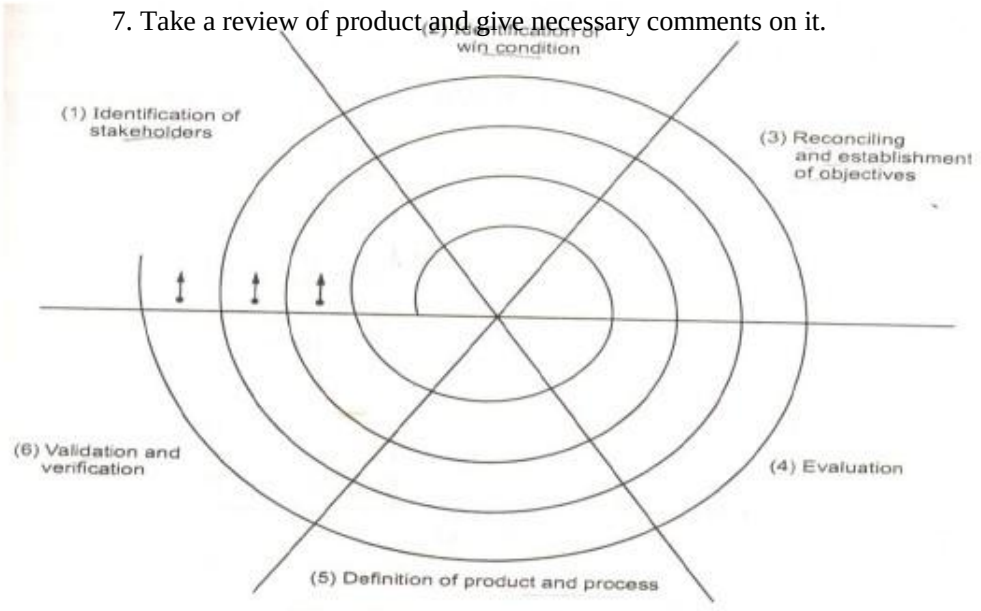


Fig. 2.8 WIN-WIN spiral model

- There are three anchor points that can be defined in WINWIN spiral model.

1. LCO - That means Life Cycle Objective. It defines the objectives for majol software engineering activities.

2. LCA - That means Life Cycle Architecture. It defines the software architectures that can be produced with all the objectives are set.

3. IOC - That means Initial Operational Capability. It represents software with all the desired initial operational capabilities.

1.4.5 Prototyping

- In prototyping model initially the requirement gathering is done.
- Developer & customer define overall objectives; identify areas needing more requirement gathering.
- Then a quick design is prepared. This design represents what will be visible to user- in input and output format.
- From the quick design a prototype is prepared. Customer or user evaluates the prototype in order to refine the requirements. Iteratively prototype is tuned for satisfying customer requirements. Thus prototype is important to identify the software requirements.
- When working prototype is built, developer use existing program fragments or program generators", to throwaway the prototype and rebuild the system to high quality

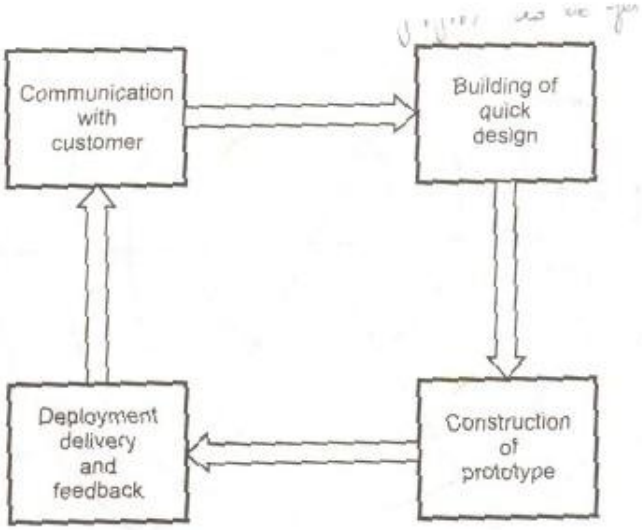


Fig. 2.9 Prototyping

- Certain classes of mathematical algorithms, subset of command driven systems and other applications where results can be easily examined without real time interaction can be developed using prototyping paradigm.

When to choose it?

- Software applications that are relatively easy to prototype almost always involve human-machine interaction (He!) the prototyping model is suggested.
- A general objective of software is defined but not detailed input, processing or output requirements. Then in such a case prototyping model is useful.
- When the developer is unsure of the efficiency of an algorithm or the adaptability of an operating system then prototype serves as a better choice.

Drawbacks of Prototyping

1. In the first version itself, customer often wants "few fixes" rather than rebuilding of the system. Whereas rebuilding of new system maintains high level of quality.
2. The first version may have some compromises.
3. Sometimes developer may make implementation compromises to get prototype working quickly. Later on developer may become comfortable with compromises and forget why they are inappropriate.

1.4.6 Object Oriented Model

The fountain model is a kind of object oriented model.

- The iterations can be applied within as well as between the phases.
- There is incremental development of software product.
- The parallelism and iteration between the phases is possible.

See Fig. 2.10 on next page.

1.5 System Engineering

System engineering means designing, implementing, deploying and operating systems which include hardware, software and people)

- System engineering is a process that focuses on variety of system elements, analyzing, designing, and organizing those elements into a system that can be a product, a service or a technology.
- The system engineering process is also called business engineering when used for business enterprises.

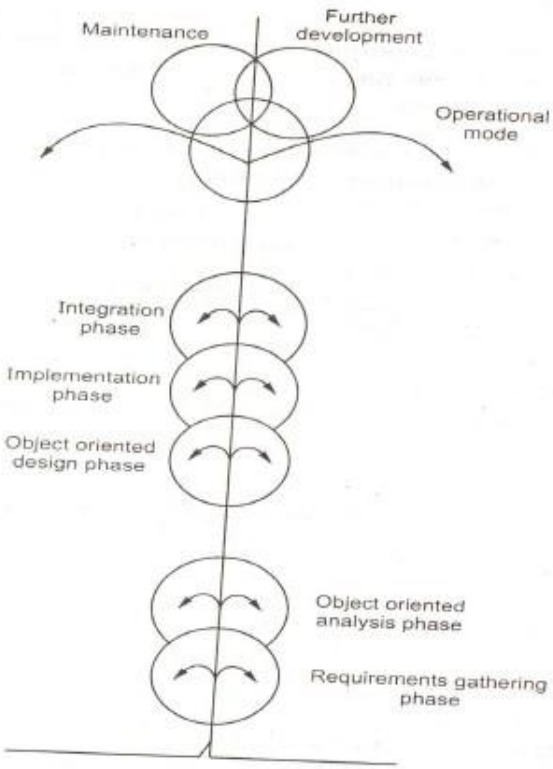


Fig. 2.10 Object oriented design model

- The system engineering process is also called product engineering when a product is to be built.
- The system engineering works for understanding the requirements of computer based system with the help of customers, users and stakeholders.

- The system engineering should produce an effective representation of system. The effective representation can be prototype, specification or a symbolic model. This representation should have project operational, functional and behavioral characteristics of the system to be built.

1.6 Computer Based System

A system can be defined as a purposeful collection of inter-related components working together to achieve some common objective. The system components are dependant on other system components.

The computer based system can be defined as "a set or an arrangement of elements that arc organized to accomplish some predefined goal by processing information",

Various elements of computer based system are

- **Software** - Computer software is a collection of computer programs, data structures, and related documentation that builds the logical method, procedures or control that is required.
- **Hardware** - Hardware is a collection of electronic devices that provide computing capability, interconnectivity devices for communicating with external world connectivity. Examples of such interconnecting components are network switches, telecommunication devices.
- **People** - Users and operators of hardware and software.
- **Database** - Database is a large and organized collection data that can be accessed by software.
- **Documentation** - Supporting descriptive information that represents the use and operation of the system. The documentation can be in the form of hard copy manuals, online help files or web sites.
- **Procedures** - A series of steps that define the specific use of each system element.

The elements combine together to transform the information. Complex systems are actually a hierarchy of macro-elements that are themselves systems. The role of system engineer is to define the elements for a specific computer based system by considering the context of the system.

For example : A factory automation system is collection of various systems that themselves are the combination of various elements.

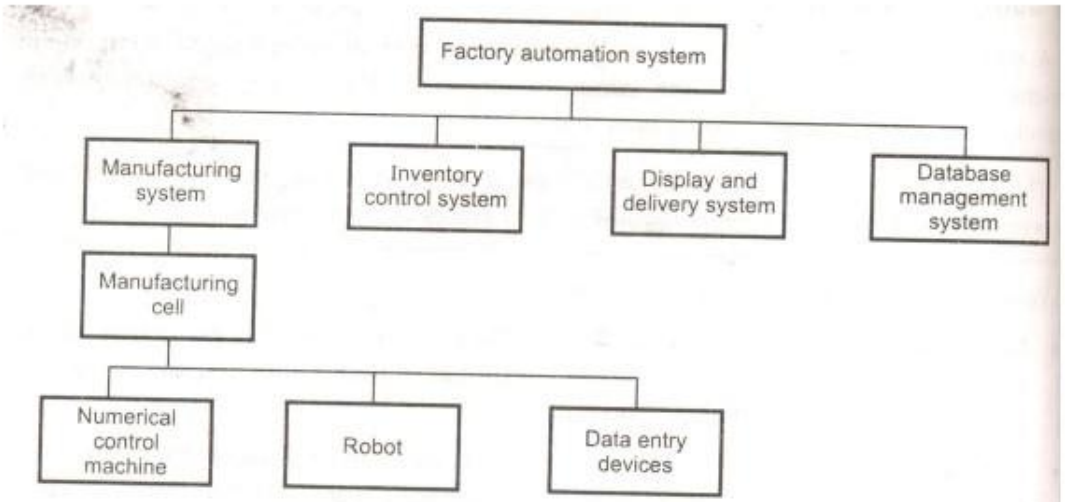


Fig. 2.11 Factory automation system

Various Macro-elements such as Numerical control machine, Robot, Data entry devices form the lowest level of the system. These macro-elements are basically formed by combining various elements such as hardware, software, procedures, and databases. These lowest level elements together form manufacturing cell. One or more such cells form manufacturing system. Thus the manufacturing system is the next level in the hierarchy. Along with manufacturing system, inventory control system, database management system build the factory automation system.

1.7 Verification and Validation

The purpose of verification and validation is to confirm system specification and to meet the requirements of system customers. Verification represents the set of activities that are carried

out to confirm that the software correctly implements the specific functionality. Validation represents set of activities that ensure that the software that has been built is satisfying the customer requirements.

The verification and validation involve checking and review of processes and system testing.

System testing means executing the system with various test cases. The test cases are derived from specification of input data which is to be processed by the system.

The testing can be carried out using following steps.

1. **Unit testing -** In this type of testing individual components are tested.
2. **Module testing -** Related collection of independent components are tested.
3. **Sub-system testing** - This is a kind of integration testing. Various module~ ~;.integrated into a subsystem and the whole subsystem is tested. The focus is to test the integration or to test an interface.
4. **System testing -** In this testing, the whole system is tested.
5. **Acceptance testing** - This type of testing involves testing of the system with customer data. If the system behaves as per customer need then it is acceptable.

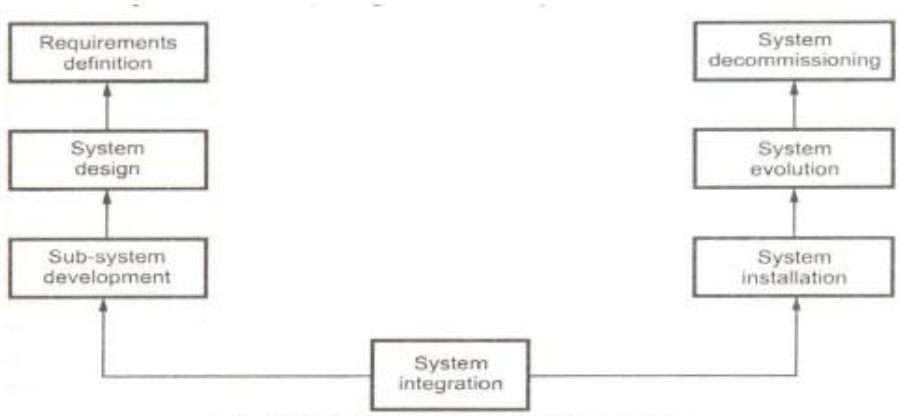


Fig. 2.12 System engineering process

1.8 Life Cycle Process

System engineering process usually follows a waterfall model because of the need for parallel development of different parts of the system.

1. System requirements definition

In this phase three types of requirement can be defined.

- Abstract functional requirements - That means system functions are defined in an abstract way.
- System properties - Nonfunctional requirements for the system in general are defined.
- Undesirable characteristics - Means unacceptable system behavior is specified. While specifying this type of requirement the overall organizational objectives for the system should also be specified.

System objectives

It is the most important part of system requirement definition to establish overall objectives of the system. The functional and organizational objectives define the system objective. It is necessary to clearly specify the system objective.

System requirements problems

Various problems associated with the requirements need to be analyzed before proceeding. Changing as the system is being specified. Must anticipate hardware and communications developments over the lifetime of the system.Hard to define non-functional requirements (particularly) without an impression of component 6 structure of the system.

2. The system design process

The system design process can be carried out using following steps.

- Partition requirements - Organize all the requirements into related groups.
- Identify sub-systems - Identify a set of sub-systems which collectively can meet the requirements of system.
- Assign requirements to sub-systems - Individual subsystem may have some arise requirements. During the assignment of requirements problems may when Commercial - Off - The - Shelf (COTS) components are integrated.

- Specify sub-system functionality.
- Define sub-system interfaces.

3. Sub-system development

After system design the system development starts. The system as a whole can be d developed by developing the sub-systems. Typically it involves development of el parallel projects, development of required hardware, software and necessary communication between them. It may involve use of some COTS (Commercial Off - The - Shelf) procurement.

An overall architectural description should be produced to identify sub-systems 2 making up the system. Once such subsystems have been identified, they may be specified in parallel with other systems and the interfaces between sub-systems defined.

4. System integration

It is the process of putting hardware, software and people together to make a system. It should be tackled incrementally so that sub-systems are integrated one at a time. At this stage interface problems between sub-systems can be recognized.

5. System installation

In this phase the system has to be installed in customer's environment.

Various issues that need to be handled are

- Environmental assumptions may be incorrect.
- There may be human resistance to the introduction of a new system.
- System may have to coexist with alternative systems for some period.
- There may arise some physical installation problems (e.g. cabling problems).
- Operator training has to be identified.
-

6. System evolution

- The lifetime of large systems is long. They must evolve to meet changing requirements.
- The evolution may be costly because

- Changes must be analyzed from a technical Lind business perspective.

- Sub-systems interact so unanticipated problems can arise.

- System structure is corrupted as changes are made to it.

- Existing systems which must be maintained are sometimes called legacy sys1ems.

7. System decommissioning

Taking the system out of service after its useful lifetime is called as system decommissioning.

It may require removal of materials (for e.g. dangerous chemicals) which pollute the environment. Such a removal should be planned for in the system design by encapsulation.

Sometimes system decommissioning may require data to be restructured and converted to the form useful to other system.

1.9 Development Process

The development process is one of the processes involved in systems engineering. It interacts with system procurement process and with the operational process.

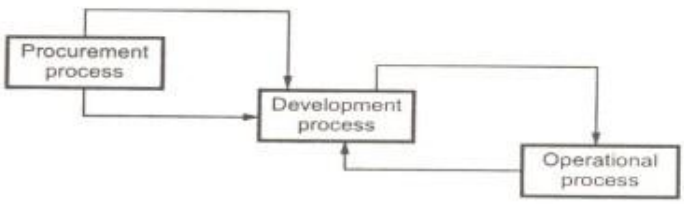


Fig. 2.13 System process

- The procurement process is a process in which procurement of the system is involved. System procurement means acquiring a system for an organization to meet some need.

Some system specification and architectural design is usually necessary before procurement.

- The specification may allow using a Commercial Off - The - Shelf (COTS) system. This is cheaper than building the system from scratch.
- Large complex systems usually consist of combination of off - the - shelf and specially designed components. The procurement processes for different types of component are usually different.
- Operational processes are the processes that perform all the functionalities required for its intended purpose. For a new system, the operational processes need to be defined during system design itself.
- Operational processes should be flexible and should not force operations to be done in a particular way.

1.10 System Engineering Hierarchy

System engineering hierarchy can be built in following manner.

- The system engineering hierarchy begins with the world view in which entire product domain is examined.
- The world view can be refined to focus the specific domain of interest in the domain view.
- Within a specific domain a system element can be analyzed. The system element can be data, software, hardware or people. This becomes an element view.
- Finally analysis, design and construction of targeted element is initiated.
- Thus at the top of the hierarchy a broad context is established and at the bottom detailed technical activities can be carried out.

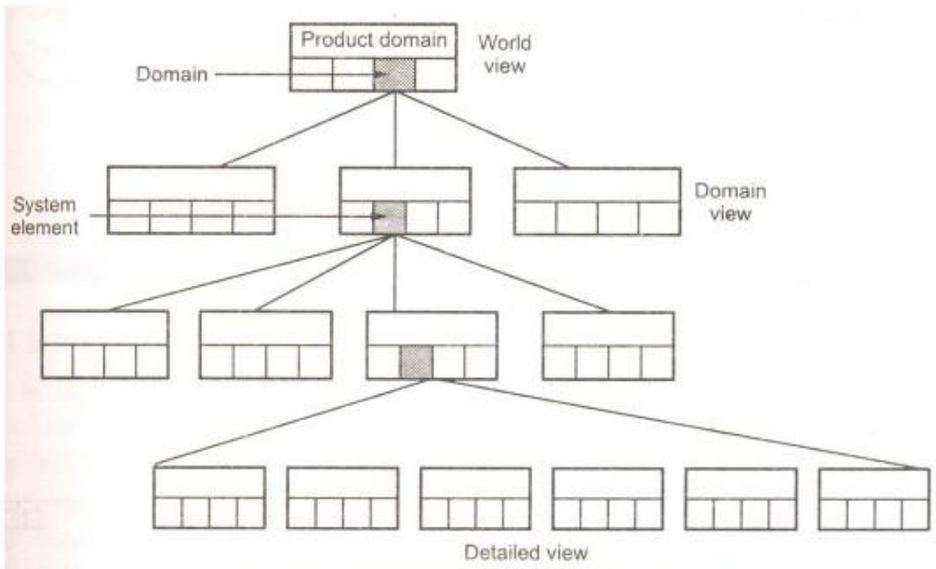


Fig. 2.14 The system engineering hierarchy

The system hierarchy can be specified in a formal manner as

$W_v = (D_1, D_2, \cdots, D_n)$

Where W_v means world view which is a collection of corresponding domain Di.

D, = (E_1, E_2, _ .. , E_n)

Here each domain is composed of specific element E_i

$E_i = (C_1, C_z, \cdots C_n)$

Here each element is specified by corresponding system components Ci.

In this way the system engineer narrows his focus as he' moves from top to down.

In this chapter we have seen that software engineering is a discipline that combines Processes. Process plays an important role in developing software in a qualitative manner. Various process models suggest a systematic approach in software development process.

The systems engineering process includes specification, design, development, integration and testing. Thus system engineering help to translate customer's need into system model that makes use of one or more system elements.

Questions

1. Justify the statement “Software Engineering is a layered technology”.

2. What do you mean by the software process?

3. What are the common framework activities of software process?

4. Explain several of CMM.

5. What are the drawbacks of waterfall model?

6. Explain the incremental model.

7. Compare Waterfall model and spiral model.

8. Describe the Rapid Application development model.

9. Describe the demerits of Prototyping model.

10. Define the term: System Engineering.

11. Describe the computer based system.

12. Explain the system engineering Hierarchy.

UNIT – II

Software Requirements

2.1 Introduction

In requirement engineering there is a systematic use of principles, technique and tools for cost effective analysis, documentation and user needs. Both the software engineer and customer take an active role in requirement engineering.

In this chapter we will discuss the concept of user and functional requirements. We describe functional and non functional requirements. Finally we will learn how software requirements may be organized in requirements document.

What is requirement engineering?

Requirement engineering is the process of

- establishing the services that the customer requires from system
- and the constraints under which it operates and is developed

The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

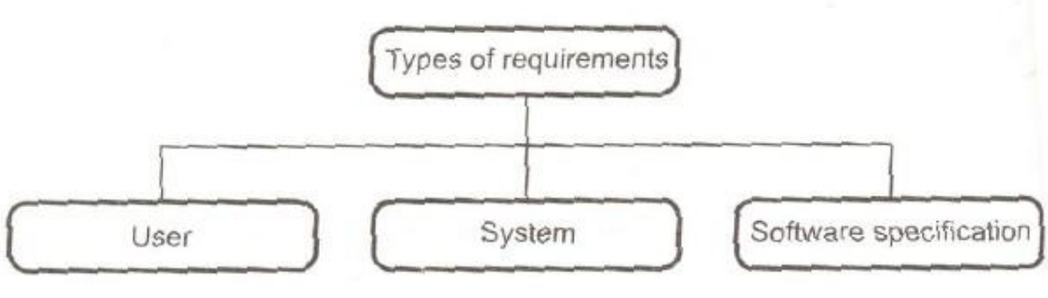
What is a requirement?

A requirement can range from a high-level abstract statement of a servi<:e or of a system constraint to a detailed mathematical functional specification.

The requirement must be open to interpretation and it must be defined in detail.

Types of requirements

The requirements "n be classified as



- **User requirements**

It is a collection of statements in natural language plus description of the service the system provides and its operational constraints. It is written for customers.

- **System requirements**

It is a structured document that gives the detailed description of the system services. It is written as a contract between client and contractor.

- **Software specification**

It is a detailed software description that can serve as a basis for design or implementation. Typically it is written for software developers.

2.2 Functional and Non Functional

Software system requirements can be classified as functional and non functional requirements.

2.2.1 Functional Requirements

- Functional requirements should describe all the required functionality or system services.
- The customer should provide statement of service. It should be clear how the system should react to particular inputs and how a particular system should behave in particular situation.
- Functional requirements are heavily dependent upon the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.

For example: Consider a library system in which there is a single interface provided. to multiple databases. These databases are collection of articles from different libraries. A user can search for, download and print these articles for a personal study.

From this example we can obtain functional Requirements as-

1. The user shall be able to search either all of the initial set of databases or select a subset from it.
2. The system shall provide appropriate viewers for the user to read documents in the document store.
3. A unique identifier (ORDER_ID) should be allocated to every order. This identifier can be copied by the user to the account's permanent storage area.

2.2.1.1 Problems Associated with Requirements

- Requirements imprecision

  1. Problems arise when requirements are not precisely stated.
  2. Ambiguous requirements may be interpreted in different ways by developers and users.
  3. Consider meaning of term 'appropriate viewers'
     - User intention - special purpose viewer for each different document type;
     - Developer interpretation - Provide a text viewer that shows the contents of the document.

- Requirements completeness and consistency -

The requirements should be both complete and consistent. Complete means they should include descriptions of all facilities required. Consistent means there should be no conflicts or contradictions in the descriptions of the system facilities.

Actually in practice, it is impossible to produce a complete and consistent requirements document.

2.2.2 Non Functional Requirements

- The non functional requirements define system properties and constraints.

Various properties of a system can be: Reliability, response time, storage requirements. And constraints of the system can be: Input and output device capability, system representations etc.

- Process requirements may also specify programming language or development method.
- Non functional requirements are more critical than functional requirements. 1£ the non functional requirements do not meet then the complete system is of no use.

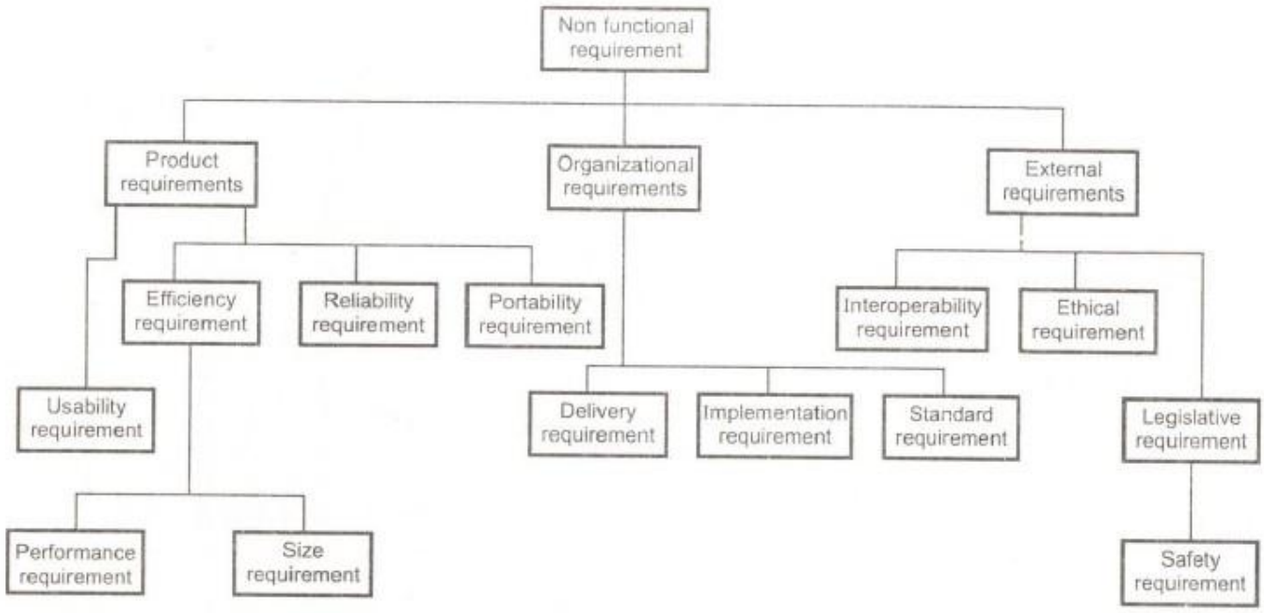


Fig. 3.2 Types of non functional requirement

Product requirements

These requirements specify how a delivered product should behave in a particular way. For instance: execution speed, reliability.

Organizational requirements

The requirement which are consequences of organizational policies and procedures come under this category. For instance: process standards used implementation requirements.

External requirements

These requirements arise due to the factors that are external to the system and its development process. For instance: interoperability requirements, legislative requirements.

In short, non functional requirements arise through

i) User needs

ii) Because of budget constraints

iii) Organizational policies

iv) The need for interoperability with other software or hardware systems v) because of external factors such as safety regulations.

2.3 User Requirements

- The user requirements should describe functional and non functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.

- User requirements are defined using natural language, tables and diagrams because these are the representations that can be understood by all users.

2.3.1 Guidelines for Writing User Requirements

For example

Consider a spell checking and correcting system a word processor. The user requirements can be given in natural language as

- The system should posses a traditional word dictionary and user supplied dictionary. It shall provide a user-activated facility which checks the spelling of words in the document against spellings in the system dictionary and user-supplied dictionaries.

- When a word is found in the document which is not given in the dictionary, then the system should suggest 10 alternative words. These alternative words should be based on a match between the word found and corresponding words in the dictionaries.

- When a word is found in the document which is not in any dictionary, the system should propose following options to user:

1. Ignore the corresponding instance of the word and go to next sentence.

2. Ignore all instances of the word

3. Replace the word with a suggested word from the dictionary

4. Edit the word with user-supplied text

5. Ignore this instance and add the word to a specified dictionary

2.4 System Requirement

- System requirements are more detailed specifications of system functions, services and constraints than user requirements.

- They are intended to be a basis for designing the system.

- They may be incorporated into the system contract.

- The system requirements can be expressed using system models.

- The requirements specify what the system does and design specifies how it does.

- System requirement should simply describe the external behavior of the system and its operational constraints. They should not be concerned with how the system should be designed or implemented.

- For a complex software system design it is necessary to give all the requirements in detail.
- Usually, natural language is used to write system requirements specification and user requirements.

Why requirement and design are inseparable?

- A system architecture may be designed to structure the requirements;
- The system may inter-operate with other systems and that may generate design requirements;
- The use of a specific design may be a domain requirement.

2.5 Requirement Engineering Process

- The requirement engineering processes are the processes used to discover, analyze and validate the system requirements.
- The processes used for requirement engineering vary widely. These processes are dependant on the application domain, the people involved and the organization developing the requirements.

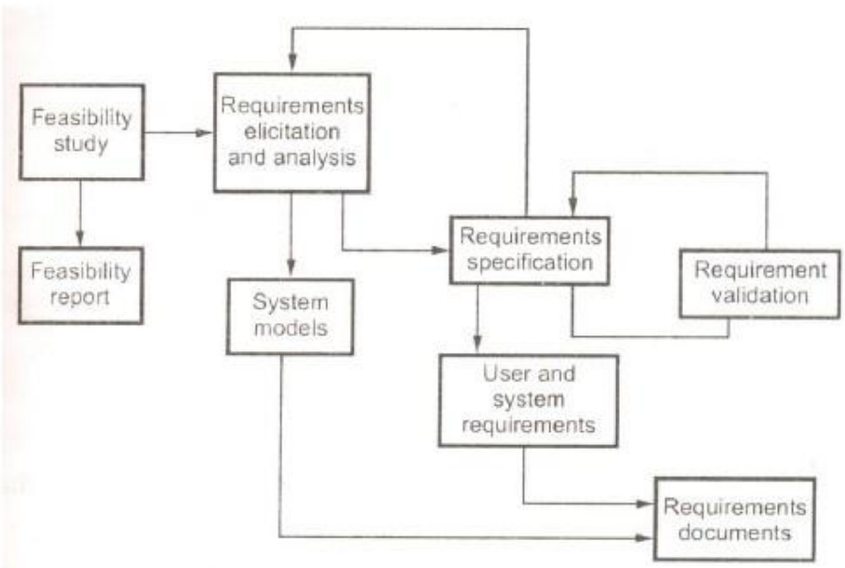


Fig. 3.3 Requirement engineering process

- The generic activities that are common to all processes are given as below-

1. Requirements elicitation;
2. Requirements analysis;
3. Requirements validation;
4. Requirements management.

2.6 Feasibility Studies

Definition: A feasibility study is a study made to decide whether or not the proposed system is worthwhile.

The focus of feasibility study is to check

- If the system contributes to organizational objectives.
- If the system can be engineered using current technology
- J f the system is within the given budget
- If the system can be integrated with other useful systems.

The implementation of feasibility study is based on the information assessment (what is required), information collection and report writing.

While performing the feasibility study, following questionnaires to the people in the organization should be asked -

1. What if the system wasn't implemented?
2. What are current process problems?
3. How will the proposed system help?
4. What will be the integration problems?
5. Is new technology needed? What skills?
6. What facilities must be supported by the proposed system?

2.7 Elicitation

Before requirements can be analyzed, modeled they must undergo through the proces~ of elicitation process,

- The requirements elicitation means requirements discovery.
- The most commonly used elicitation technique is to conduct meeting. It involves technical staff working with customers in order to find the application domain, the services that the system should provide and the system's operational constraints.
- In requirement elicitation various entities are involved such as end-users, managers, engineers involved in maintenance, domain experts, and trade unions. These all are called stakeholders.
- Problems in requirement analysis are -

1. Stakeholders don't know what they want
2. Stakeholders may have unrealistic expectations.
3. Different stakeholders may have different requirements.
4. Certain Political factors may affect requirements.
5. Business or economic changes create dynamic environment.

2.7.1 Facility Application Specification Techniques (FAST)

Facility application specification technique is an approach in which joint team of customers and developers work together to identify the pr.Q!2lem, propose elements of solution,

negotiate different approaches and prepare a specification for preliminary set of solution requirements.

Guideline for FAST approach -

1: A meeting should be conducted and attended by both software engineers and customers. The place of meeting should be a neutral site.

2: Rules for preparation and participation must be prepared.

3: An agenda should be prepared in such a way that it covers all the important point as well as it allows all the new innovative ideas.

4: A facilitator controls the meeting. He could be customer, developer or outsider.

5: A definition mechanism is used. The mechanism can be work sheets, flip charts, wall stickers, electronic bulletin board, chart room, virtual forum.

6: The goal is to identify the problem, decide the elements of solution, negotiate different approaches and specify the preliminary set of solution requirements.

- In FAST meeting each FAST attendee is asked to prepare - a list of objects, list of services and a list of constraints.
- The list of objects consists of all the objects used in the system, the objects that are produced by the system and the objects that surround the system.
- The list of services contain all the required functionalities that manipulate or interact with the objects.
- The list of constraints consists of all the constraints of the system such as cost, rules, memory requirement, speed accuracy etc.
- As the FAST meeting begins, the very first issue of discussion is the need and justification for the new product. Once everyone agrees upon the fact that the product is justified, each participant has to present his lists.
- These lists are then discussed, manipulated and these modified or refined lists are combined by a group.
- The combined list eliminates redundant entries adds new ideas that come up during the discussion. The combined list is refined in such a way that it helps in building the system.
- The combined list should be prepared in such a way that a "consensus lists" can be prepared, for object, services and constraints.
- A team is divided into subteams. Each sub team develops a minispecification from each consensus list.
- Finally a complete draft specification is developed.

For example -

A FAST team is working on a commercial product. A following product description is given as below -

"Nowadays the market for video game is growing rapidly. We would like to enter this market with more features, like attractive GVl, multiple sound setting, realistic (3D) animations. This product tentatively called 'Gamefun'. At the end of game, scores of each player should be displayed".

The FAST attendee prepare following lists -

1. List of objects - Display, menu, a sound, an event (moving from one level to another) and so on.

2. List of services - setting sounds, setting colors in GUI, HELP, instructions for players, score card etc.

3. List of constraints - must be user friendly, must have high speed, must accommodate less size, should have less cost.

The minispecification for Menu (object) can be as given below-

- Contains 'Start game' and 'exit' options.
- List of all functional keys with corresponding functionality.
- Software provides interaction guidance, quick tour, sound controls.
- All players will play or interact through keys.
- Software provides facility for change in the look of GUI.
- Software displays score~ of each player.

2.8 Validation and Management

Requirement validation is a process in, which it is checked that whether the gathered requirements represent the same system that customer really wants.

- In requirement validation the requirement errors are fixed. Requirements error costs are high so validation is very important. Fixing a requirements error atter delivery may cost up to 100 times the cost of fixing an implementation error.

- Requirement checking can be done in following manner -

1. Validity - Does the system provide the functions which best support the customer's needs?

2. Consistency- Are there any requirements conflicts?

3. Completeness- Are all functions required by the customer included?

4. Realism- Can the requirements be implemented according to budget and technology?

5. Verifiability-Can the requirements be checked?

6. Requirements validation techniques

Requirements validation techniques

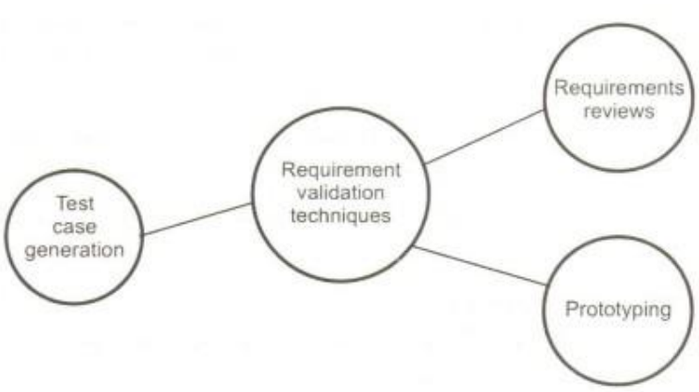


Fig 2.4 Requirement validation technique

1. **Requirements reviews** - Requirement review is a systematic manual analysis of the requirements.

- The requirement review should be taken only after formulation of requirement definition. And both the customer and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal.
- Good communications should take place between developers, customers and users. Such a healthy communication helps to resolve problems at an early stage.

Prototyping - The requirements can be checked using executable model of system.

3. **Test-case generation** - In this technique, the various tests are developed for requirements. The requirement check can be carried out with -

- Verifiability: Is the requirement realistically testable?
- Comprehensibility: Is the requirement properly understood?
- Traceability: Is the origin of the requirement clearly stated?
- Adaptability: Can the requirement be changed without a large impact on ther requirements?

Requirements management

Why requirements get change?

- Requirements are always incomplete and inconsistent. New requirements occur during the process as business needs change and a better understanding of the system is developed.
- System customers may specify the requirements from business perspective that can conflict with end user requirements
- During the development of the system, its business and the technical environment may get changed.

Requirement management process

Following things should be planned during requirement process.

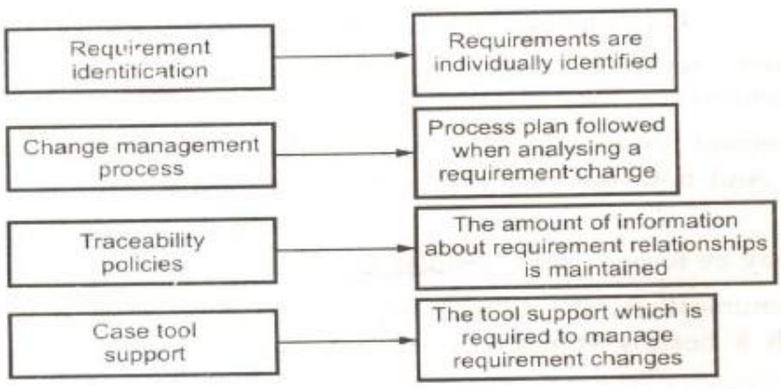


Fig. 3.5 Planning in requirement management process

- Traceability is concerned with relationship between requirements their sources and the system design.

Various types of traceability are

1. **Source traceability-** These are basically the links from requirement to stakeholders who propose these requirements.

2. **Requirements traceability** These are the links between dependant requirements. .

3. **Design traceability-** These are the links from requirements to design.

- Case tool support is required for

1. Requirement storage

2. Change management

3. Traceability management

2.9 Software Prototyping

Definition : System prototyping is a rapid software development for validating the requirement.

The use of system prototypes is to help customers and developers to understand the system requirements.

Under software prototyping various activities being carried out are

1. Requirements elicitation - User can perform various experiments with the prototype to check the system support.

2. Requirements validation - Prototype can show errors and omissions in requirement.

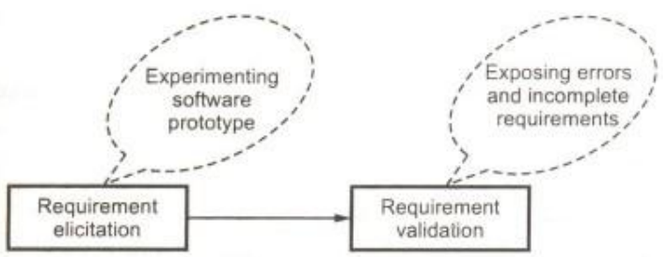


Fig. 3.6 Software prototyping activities

Benefits of Software Prototyping

1. Improvement in software user and developer gets improved.

2. If any service is missing or confusing then that can be identified.

3. Prototype can serve as a basis for deriving system specification.

4. Working system becomes available as there is a closer match of prototype with actual system.
5. Design quality can be improved.
6. System can be maintained easily.
7. Development efforts may get reduced.

8. System usability can be improved.

2.10 Prototyping in Software Process

There are two approaches

1. **Evolutionary Prototyping** - In this approach of system development, the initial prototype is prepared and it is then refined through number of stages to final stage.

2. **Throw-away Prototyping** - Using this approach a rough practical implementation of the system is produced. The requirement problems can be identified from this implementation. It is then discarded. System is then developed using some different engineering paradigm.

2.10.1 Evolutionary Prototyping

- Objective

The principle objective of evolutionary model is to deliver the working system to the end-user. The process of development starts with well understood requirements.

It must be developed for the systems where the specifications can not be developed in advance.

For example AI systems

For systems that allow rapid system development the evolutionary prototype is used.

- Advantages

1. Fast delivery of the working system.

2. User is involved while developing the system.

3. More useful system can be delivered.

4. Specification, design and implementation work in co-ordinated manner.

- Problems

1. **Management problems** - Typically a waterfall model is adopted in \·vhich the development skill is required in all the development teams.

2. **Maintenance problem** - Continuous changes may lead to changes in the system structure that may cause maintenance problems.

3. **Verification** - It is impossible as there is no specification.

- **Incremental development**

After designing the overall architecture the system is developed and delivered in series of increments. User may perform experiments with the delivered version of increment which

becomes a basis for development of the system. Thus overall objective 01 incremental development is to manage the processes and provide the better :;ystem structure.

2.10.2 Throw Away Prototyping

- Objective

The principle objective of throwaway prototype is to validate or derive the system requirements. The process of development starts with poorly understood requirements.

The throwaway prototype is developed to reduce the requirement risks.

The prototype is developed from initial specification, delivered to user for cAperiments 'llld then discarded.

- Advantages

If -the delivered model is not as per the customers need, then it can be discarded and development can occur with another new engineering paradigm.

Requirement risks are very less.

- **Problems**

1. Developers may be pressurized to deliver throwaway prototype as final system. This is not recommended.

2. The throwaway prototype can be undocumented.

3. Sometimes organizational quality standard may not be strictly applied.

4. Changes made during the software development process may degrade the system structure.

3.11 Rapid Prototyping Techniques

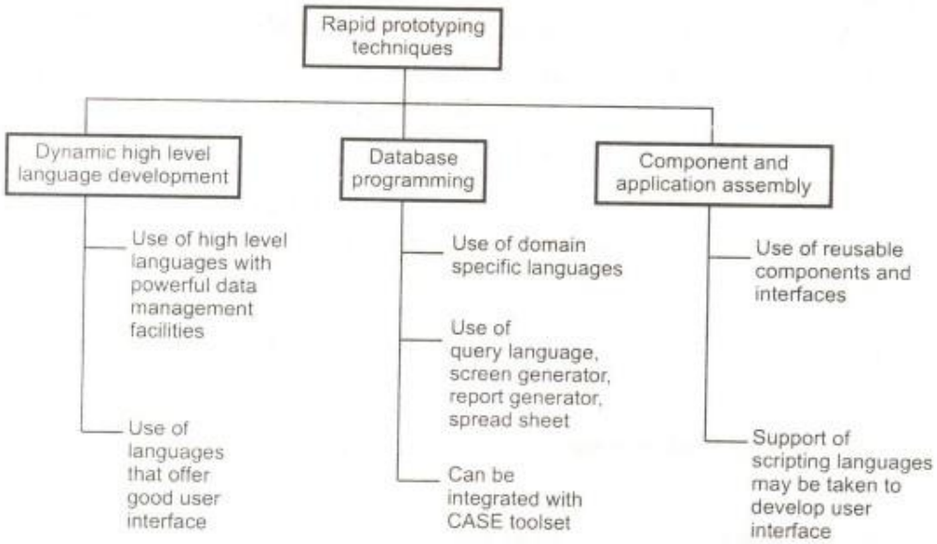


Fig. 3.7 Rapid prototyping technique

As shown in the Fig. 3.7 various rapid prototyping techniques are -

1. Dynamic high level language development.

2. Database programming

3. Component and application assembly

Along with these techniques visual programming is used for rapid prototyping.

Dynamic High level languages

The dynamic high level languages with efficient data management facilities can be llsed in rapid prototyping.

While choosing the prototyping language following issues are to be considered

- What is the application domain of the problem?
- What kind of user interactio!1 is required?
- What is the supporting environment for the language?

Different programming languages can be used for different parts of the system. But there should be a proper communication between these languages.

Database Programming

Various database supporting languages are used for management of different databases. These include query languages, screen generators, report generators, ~preadsheets.

A database programming is cost effective to small to medium size business systems.

This technique is also called as forth generation techniques.

Component and application assembly

Some reusable components can be used for rapid development of the system. The availability and functionality of such components should be considered.

In this technique two levels of development are used

1. Application level development

   Entire application can be integrated in the system and its functionality can be shared by other components of the system.

2. Component level development

   Individual component can be integrated in standard framework of the system. For example: COM, ORBA. A large library of such components is available for rapid prototype development.

2.12 User Interface Prototyping

- This prototype is used to pre-specify the look and feel of user interface in an effective way.
- The user interface generator ~s can be used to draw the interface and simulate the required functionality.
- Web interfaces may be prototyped using web site editors.
- The user interface consumes an increasing part of overall system development cost.
- By effective user interface the cost of overall system get increased.

2.12 Software Document

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is not a design document. As far as possible, it should set of what the system should do rather than how it should do it.

Software Requirements Specification

The software requirements provide a basis for creating the Software Requirements Specifications (SRS).

The SRS is useful in estimating cost, planning team activities, performing tasks, and tracking the team's progress throughout the development activity.

Typically software designers use IEEE STD 830-1998 as the basis for the entire Software Specifications. The standard template for writing SRS is as given below

Document Title

Author(s)

Affiliation

Address

Date

Document Version

1. Introduction

1.1 Purpose of this document

Describes the purpose of the document.

1.2 Scope of this document

Describes the scope of this requirements definition effort. This section also details any constraints that were placed upon the requirements elicitation process, such as schedules, costs.

1.3 Overview

Provides a brief overview of the product defined as a result of the requirements elicitation process.

2. General Description

- Describes the general functionality of the product such as similar system information, user characteristics, user objective, general constraints placed on design team.
- Describes the features of the user community, including their expected expertise with software systems and the application domain.

3. Functional Requirements

This section lists the functional requirements in ranked order. A functional requirement describes the possible effects of a software system, in other words, what the system must accomplish. Each functional requirement should be specified in following manner

- Short, imperative sentence stating highest ranked functional requirement.

1. Description

A full description of the requirement.

2. Criticality

Describes how essential this requirement is to the overall system.

3. Technical issues

Describes, any design or implementation issues involved in satisfying this requirement.

4. Cost and schedule

Describes the relative or absolute costs of the system.

5. Risks

Describes the circumstances under which this requirement might not able to be satisfied.

6. Dependencies with other requirements

Dependencies with other requirements Describes interactions with other requirements.

7. … any other appropriate

4. Interface Requirements

This section describes how the software interfaces with other software products or users for input or output. Examples of such interfaces include library routines, toker, streams, shilred memory, data streams, and so forth.

- **4.1 User Interfaces**

  Describes how this product interfaces with the user.

  4.1.1 GUI

  Describes the graphical user interface if present. This section should include a set of screen dumps to illustrate user interface features.

  4.1.2 CLI

  Describes the command-line interface if present. For each command, a description of all arguments and example values and invocations should be provided.

  4.1.3 API

  Describes the application programming interface, if present.

- **4.2 Hardware Interfaces**

  Describes interfaces to hardware devices.

- **4.3 Communications Interfaces**

  Describes network interfaces.

- **4.4 Software Interfaces**

  Describes any remaining software interfaces not included above.

5. Performance Requirements

Specifies speed and memory requirements.

6. Design Constraints

Specifies any constraints for the design team such as software or hardware limitations.

7. Other non-functional attributes

Specifies any other particular non functional attributes required by the system. Such as:

7.1 Security

7.2 Binary Compatibility

7.3 Reliability

7.4 Maintainability

7.5 Portability

7.6 Extensibility

7.7 Reusability

7.8 Application Compatibility

7.9 Resource Utilization

7.10 Serviceability

... others as appropriate

8. Operational Scenarios

This section should describe a set of scenarios that illustrate, from the user's perspective, what will be experienced when utilizing the system under various situations.

9. Preliminary Schedule

This section provides an initial version of the project plan, including the major tasks to be accomplished, their interdependencies, and their tentative start/stop dates.

10. Preliminary Budget

This section provides an initial budget for the project.

11. Appendices

11.1 Definitions, Acronyms, Abbreviations

Provides definitions terms, and acronyms, can be provided.

11.2 References

Provides complete citations to all documents and meetings referenced.

3.13.1 Characteristics of SRS

Various characteristics of SRS are

- Correct - The SRS should be made up to date when appropriate requirements are identified.
- Unambiguous - When the requirements are correctly understood then only it is possible to write an unambiguous SRS.

- Complete - To make the SRS complete, it should be specified what a software designer wants to create a software.
- Consistent - It should be consistent with reference to the functionalities identified.
- Specific - The requirements should be mentioned specifically.
- Traceable - What is the need for mentioned requirement? This should be correctly identified.

3.14 Analysis and Modeling

Analysis modeling is a technical representation of the syste,.

- The software engineer (basically called as analyst) builds the model using the requirements elicited from customer.
- In analysis modeling a combination of text and diagrams are used to represent the software requirements in an understandable manner.
- By building analysis models it becomes easy to uncover requirement inconsistencies and omissions.
- Analysis Model Objectives -
- To describe what the customer requires.
- To establish a basis for the creation of a software design.
- To devise a set of valid requirements after which the software can be built.

Analysis Modeling Approaches -

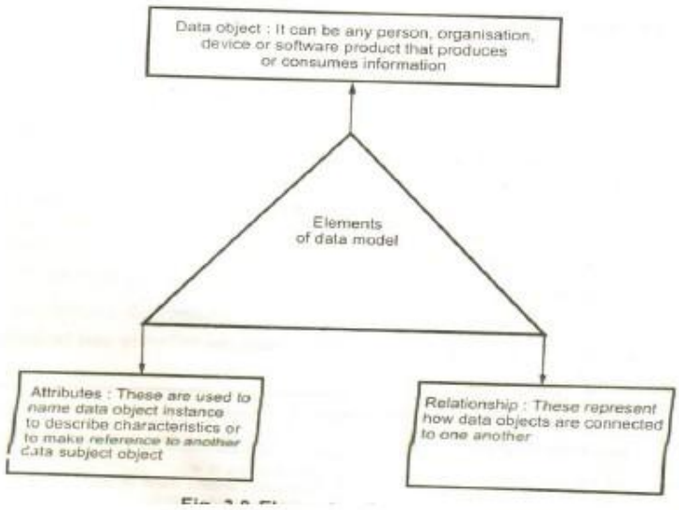
Analysis Modeling Approach	
Structured Approach	**Object Oriented Approach**
The analysis is made on data and processes in which data is transformed as separate entities.	The analysis is made on the classes and interaction among them in order to meet the customer requirements.
Data objects are modeled in a way in which data attributes and their relationship is defined in structured approach.	Unified modeling language and unified processes are used in object oriented modeling approach.

- Team chooses one approach and makes use of all the representations from it. But the effective technique is to choose best from both the approaches.
- The model representation should be such that the best model of software requirements should be given to stakeholders. And this model should help in building software design.

3.15 Data Modeling

- Data modeling is the basic step in the analysis 1odeling. In data modeling the data objects are examined independently of processing.
- The data domain is focused. And a model is created at the customer's level of abstraction.
- The data model represents how data objects are related with one another.

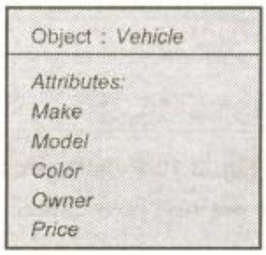
3.15.1 Data Objects, Attributes and Relationships



What is data object?

- Data object is a set of attributes (data items) that will be manipulated within the software(system).
- Each instance of data object can be identified with the help of unique identifier. For example: A student can be identified by using his roll number.
- The system cannot perform without accessing to the instances of object.
- Each data object is described by the attributes which themselves are data items.

Data object is a collection of attributes that act as an aspect, characteristic, quality, or descriptor oj the object.



The vehicle is a data object which can be defined or viewed with the help of set of attributes.

Typical data objects are

- External entities such as printer, user, speakers
- Things such as reports, displays, signals
- Occurrences or events such as interrupts, alarm, telephone call
- Roles such as manager, engineer, customer
- Organizational units such as division, departments
- Places manufacturing floor, workshops
- Structures students records, accounts, file

What are attributes?

Attributes define properties of data object

Typically there are three types of attributes -

1. Naming attributes - These attributes are used to name an instance of data object. For example: In a vehicle data object make and model are naming attributes.

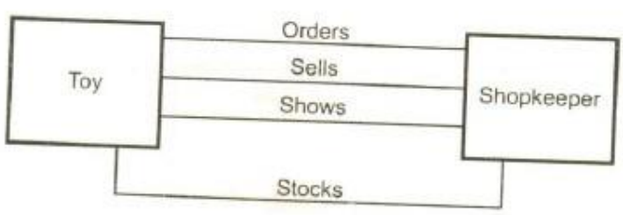
2. Descriptive attributes - These attributes are used to describe the characteristics or features of the data object. For example: In a vehicle data object color is a descriptive attribute.

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

What is relationship?

Relationship represents the connection between the data objects. For example

The relationship between a shopkeeper and a toy is as shown below



Here the toy and shopkeeper are two objects that share following relationships..

- Shopkeeper orders toys
- Shopkeeper sells toys
- Shopkeeper shows toys
- Shopkeeper stocks toys.

2.15.2 Cardinality and Modality

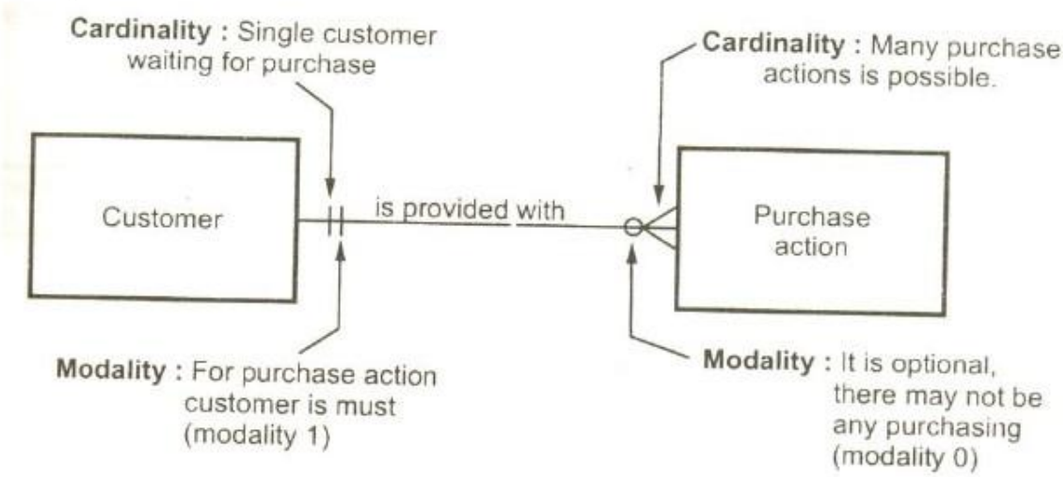
Cardinality in data modeling, cardinality specifies how the number of occurrences of one object is related to the number of occurrences of another object.

- One to one (1:1)- one object can relate to only one other object.
- One to many(1:N)- one object can relate to many objects.
- Many to many (M:N) - some number of occurrences of an object can relate to some other number of occurrences of another object.

Modality indicates whether or not a particular data object must participate in the relationship.

Modality of a relationship is 0 (zero) if there is no explicit need for the relationship to occur or the relationship is optional. The modality is 1 (one) if an occurrence of the relationship is mandatory.

Example



2.15.3 Entity Relationship Diagram

- The object relationship pair can be graphically represented by a diagram called entity relationship diagram(ERD).
- The ERD is mainly used in database applications but now it is more commonly used in data design.
- The ERD was originally proposed by Peter Chen for design of relational database systems.
- The primary purpose of ERD is to represent the relationship between data objects.
- Various components of ERD are -

Entity

- Drawn as a rectangle.
- An entity is an object that exists and is distinguishable.
- Similar to a record in a programming language with attributes.

Relationship

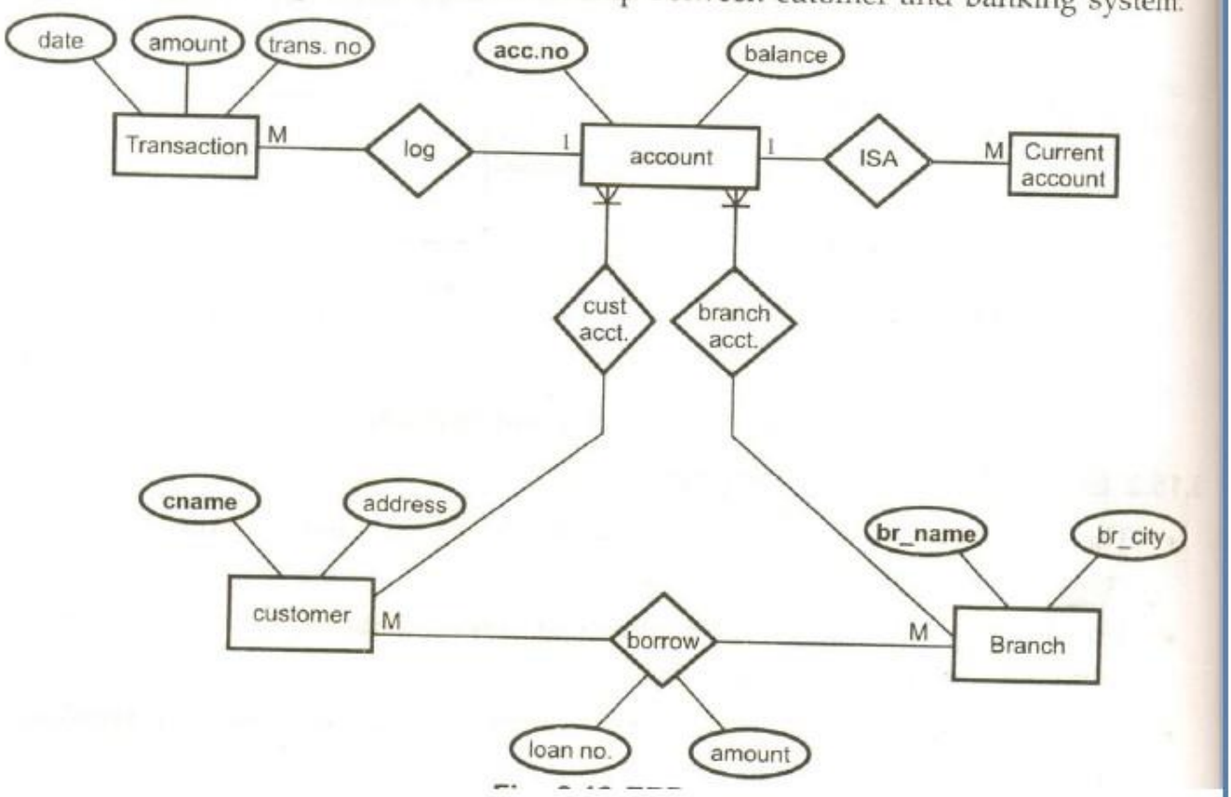
- Drawn as a diamond.
- An association among several entities.
- Relationships may have attributes.
- Relationships have cardinality (e.g., one-to-many)

Attribute

- Drawn as ellipses.
- Similar to record fields in a programming language.
- Each attribute has a set of permitted values, called the domain.
- Primary key attributes may be underlined.

Example

Following ERD represents the relationship between cutomer and banking system.



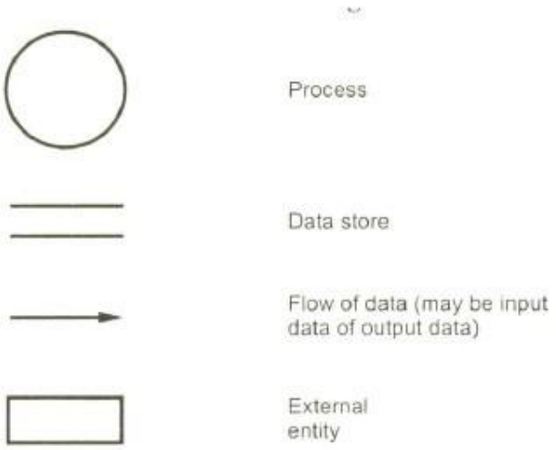
The data modeling and entity relationship diagram helps the analyst to observe the data within the context of software application.

3.16 Functional Modeling

- Functional models are used to represent the flow of information in any computer based system.
- The functional models are used to represent three generic functionalities: input, process and output.
- When functional models of application are prepared the software engineer focuses on problem specific functions.
- The basic model is prepared and over the series of iterations more and more functional details are provided.
- In structured analysis approach the functional modeling is done by using the data flow diagrams.

2.16.1 Data Flow Diagrams

- The data flow diagrams depict the information flow and the transforms that are applied on the data as it moves from input to output.
- The symbols that are used in data flow diagrams are –

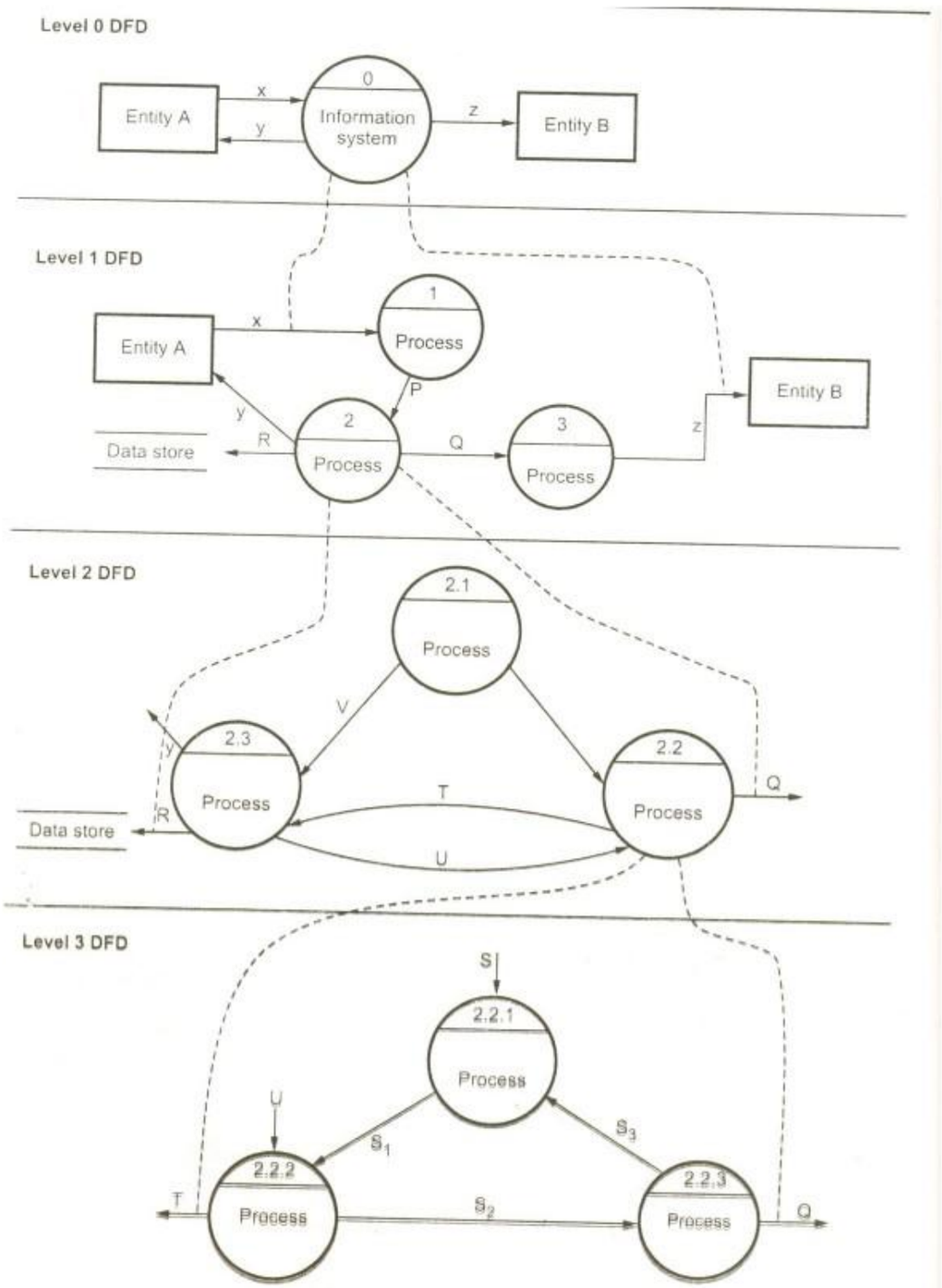


- The data flow diagrams are used to represent the system at any level of abstraction.
- The DFD can be partitioned into levels that represent increase in information flow and detailed functionality.
- A level 0 DFD is called as 'fundamental system model' or 'context model'. In the context model the entire software system is represented by a bubble with input and Ol.ltput indicated by incoming and outgoing arrows.
- Each process shown in level 1 represents the sub functions of overall system.
- The number of levels in DFD can be increased until every process reF; esents the basic functionality.
- As the number of levels gets increased in the DFD, each bubble gets refined. The following figure shows the leveling in DFD. Note that the information flow continuity must be maintained.

See Fig. 3.13 on next page.

2.16.2 Control Flow Diagrams

- The control flow diagrams show the same processes as in data flow diagrams but rather than showing data flow they show control flows
- The control flow diagrams show how events flow among processes. It also shows how external events activate the processes.
- The dashed arrow is used to represent the control flow or event.



- A solid bar is used to represent the window. This window is used to control the processes used in the DFD based on the event that is passed through the window.

- Instead of representing control processes directly in the model the specifiG1tiuns are used to represent how the processes are controlled.

- There are two commonly used representations of specifications: Control specification (CSPEC) and Process Specification (PSPEC).

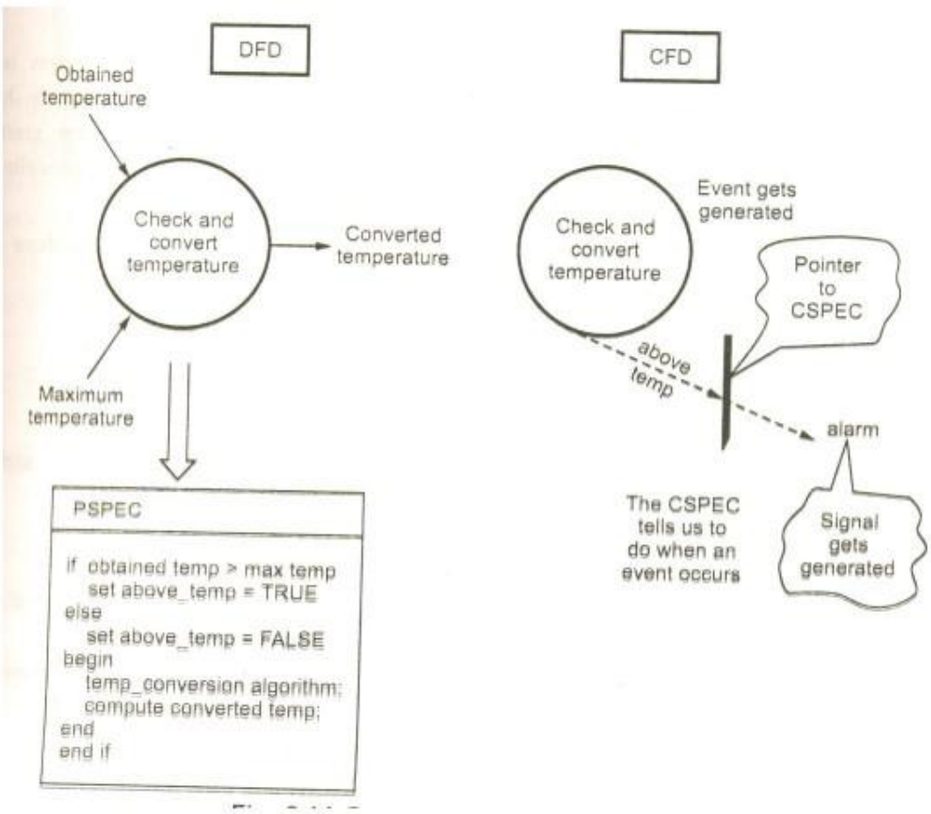
- The CSPEC is used to indicate -

1. How the software behaves when an event or signal is sensed.

2. Which processes are invoked as a consequence of the occurrence of event?

- The PSPEC is used to describe the inner workings of the process represented III ,1 flu\\' di,lgram.

- When a data input is given to the process a data condition should occur to get the control output. For Example



2.17 Behavioral Models

- Behavioural models are used to describe the overall behaviour of a system.
- The state transition diagrams are used to represent the behaviour of the system.
- The state transition diagram is basically a collection of states and events. The events cause the system to change its state.
- The state transition diagram also represents what actions are to be taken on occurrence of particular event.

State chart diagram

To understand the design of state chart diagram consider following exampt\. .... -,

Consider an elevator for n floors has n buttons one for each floor. The working of such elevator can be given as

1. There is a set of buttons called 'elevator buttons'. If we want to visit a particular floor then the elevator button for corresponding floor is pressed. It causes an illumination and elevator starts moving to visit the desired floor. The illumination is cancelled on reaching to destination.

2. There is another set of buttons called 'floor button' .When a person on particular floor want to visit another floor then the floor button has to be pressed. This makes an illumination at floor button and the elevator starts moving towards the floor where on the person is. And illumination is cancelled when the elevator reaches on the desired floor.

3. When an elevator has no request it remains at its current floor with its door is closed.

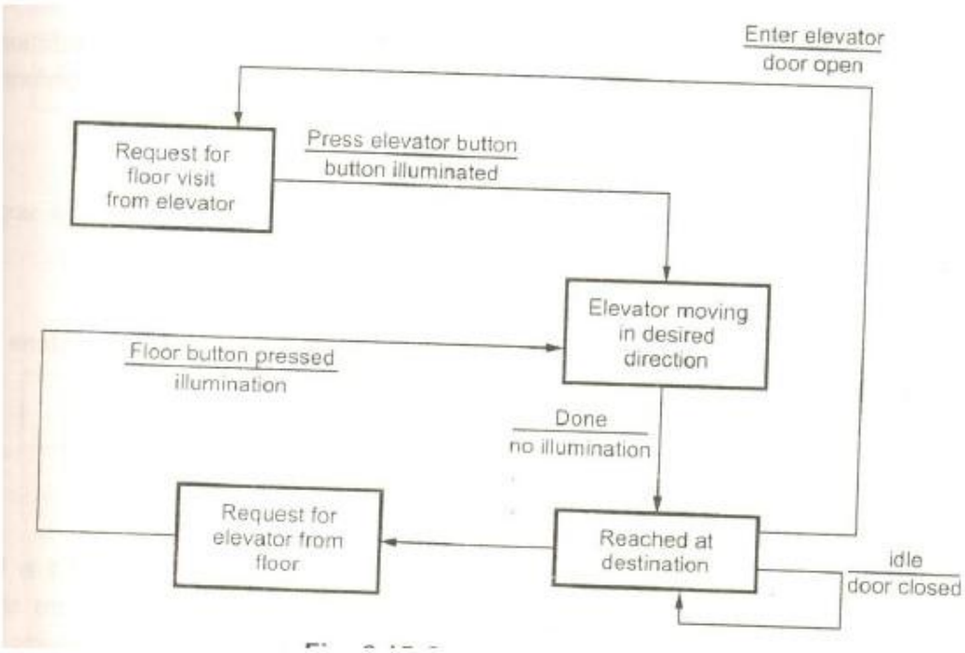
The state chart diagram is as shown in Fig. 3.15.

See Fig. 3.15 on next page.

2.18 Structured Analysis

- The structured analysis is mapping of problem domain to flows and transformations.
- The system can be modeled using :
  - Entity Relationship diagram are used to represent the data model.
  - Data flow diagram and Control flow diagrams are used to represent the functional model.
- Along with system modeling the specification can be written for the system using

1. Process Specification
2. Control Specification.



2.18.1 Designing Entity Relationship Diagrams

The entity relationship diagram is used to represent the data objects their attributes and the relationship among these data objects. The ERD is constructed in iterative manner. Following guideline is used while drawing the ERD.

1. During the requirement elicitation process, the requirements should be collected in such a way that we can evolve input, output data objects and external entities for system modeling.
2. The analysis and customer should be in a position to define the relationship between the data objects.
3. When ever a connection between data objects is identified the object relationship pair must be established. Thus iteratively relationship between all thL' objects must be established.
4. For each object relationship pair the cardinality and modality is set.
5. The attributes of each entity must be defined.
6. The entity relationship diagram is formalized and reviewed.

7. All the above steps are repeated until data modeling is complete.

2.18.2 Designing Data Flow Diagrams

- The data flow diagrams are used to model the information and function domain. Refinement of DFD into greater levels helps the analyst to perform functional decomposition.

- The guideline for creating a DFD is as given below -

1. Level a DFD i.e. Context level DFD should depict the system as a single bubble.

2. Primary input and primary output should be carefully identified.

3. While doing the refinement isolate processes, data objects and data stores to represent the next level.

4. All the bubbles (processes) and arrows should be appropriately named.

5. One bubble at a time should be refined.

6. Information flow continuity must be maintained from level to level.

- A simple and effective approach to expand the level a DFD to level 1 is to perform "grammatical parse" on the problem description. Identify nouns and verbs from it. Typically nouns represent the external entities, data objects or data stores and verbs represent the processes. Although grammatical parsing is not a foolproof but we can gather much useful information to create the data flow diagrams.

Example 1:

The data flow diagram for reservation system

This system typically works for reserving the seats for scheduled train. The timetable for various trains travelling from one city to another is maintained. The booking system is responsible for booking ticket for the travel request made by passenger.

Level 0: Context diagram

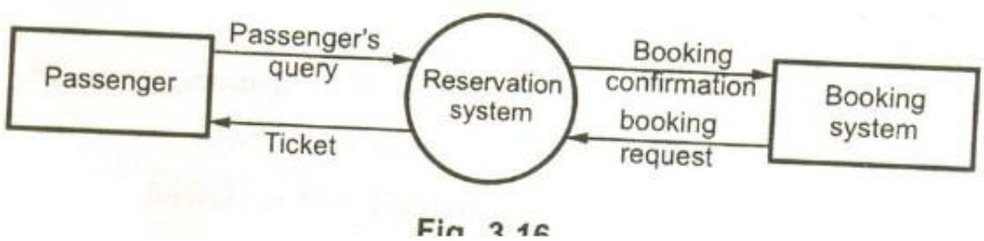


Fig. 3.16

Level 1 :

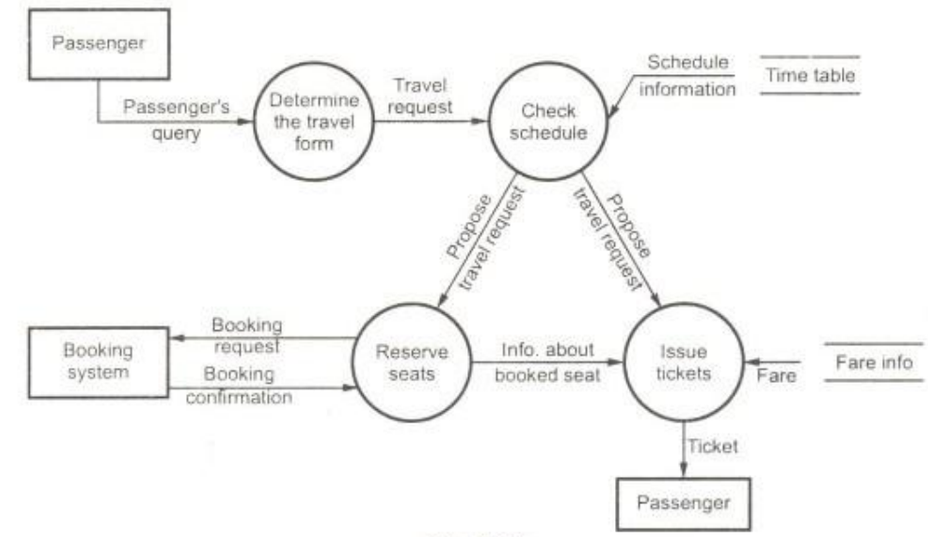
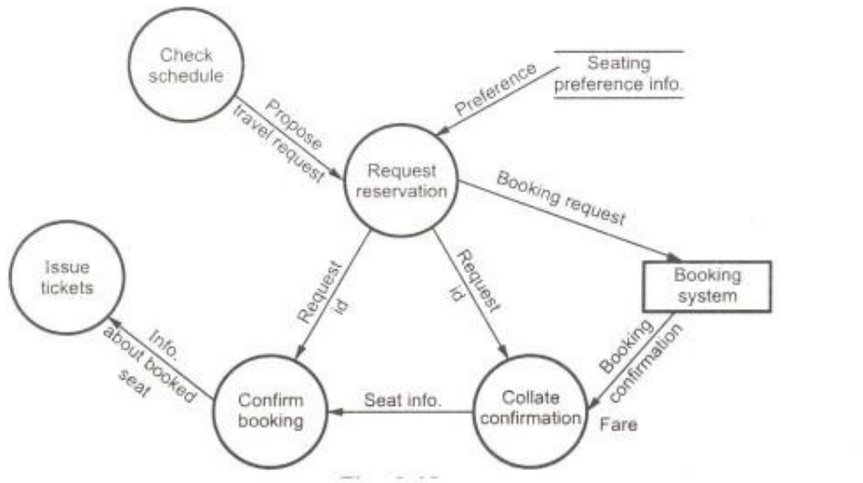


Fig. 3.17

Level 2 : In this level the task "reserve seat" is designed in detail as follows



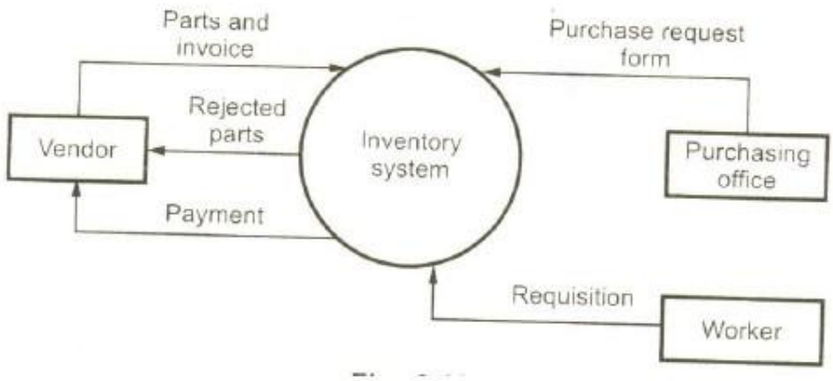
Example 2:

The DFD for inventory system is as given below.

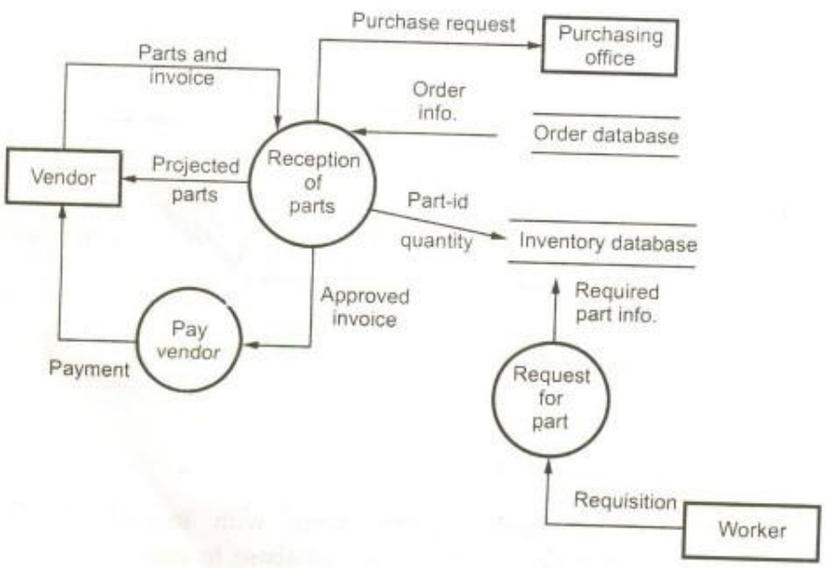
Inventory system

When parts are received from vendors along with invoice, the receiving department checks the invoice against the orders database to ensure that the correct parts were delivered. If the parts were not ordered or the order was not fulfilled correctly then parts are returned. If parts are correct, then the inventory database is updated. This is done by increasing data element quantity on hand by the quantity record tor the part id. Then a payment for the vendor is prepared. The payment transaction is also entered in general ledger. The worker of factory who needs the parts, requests tor the part by submitting requisition to the part clerk. The information on requisition form is used to update inventory file. The part clerk prepares a report by listing all the parts which are less in the stock and needs to be reordered. For preparing such report inventory database is used. For the parts listed on the report, a purchase request form is prepared and sent to purchasing office.

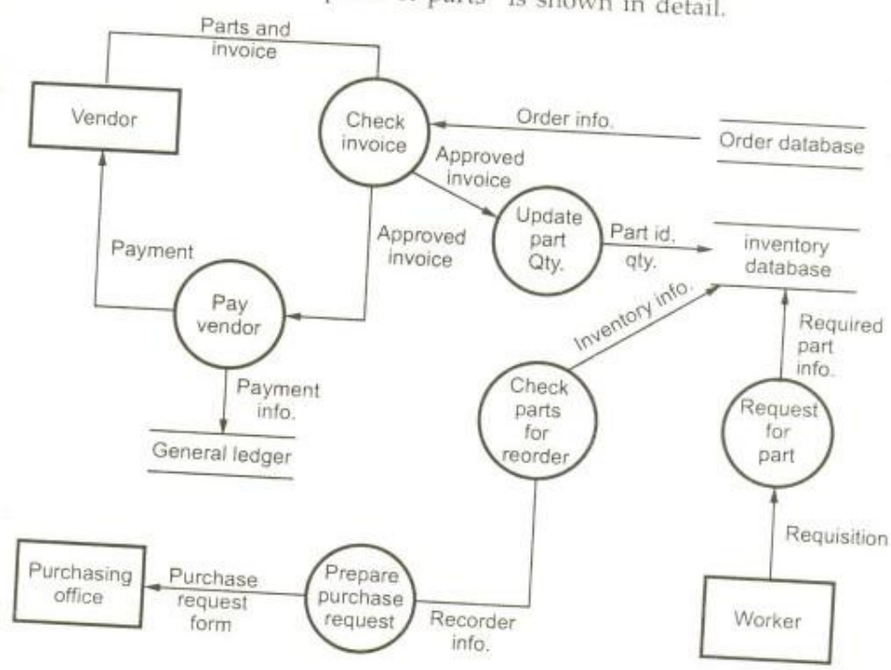
Level 0 :



Level 1 :



Level 2 : In this level "reception of parts" is shown in detail.



2.18.3 Designing Control Flow Diagrams

- There are certain applications which are event driven rather than being data driven. They produce control information rather than producing data(may be report, displays). Such applications can be modeled with the control information along with data flow modeling.
- A graphical model used to represent the control information along with the data flow model is called control flow model.
- The following guideline is used while drawing the control flow diagrams

1. List all the sensors that can be read.
2. List all the interrupt conditions.
3. List all the data conditions.
4. List all the switches actuated by the operator.
5. Use noun/verb parsing technique to identify the control information.
6. Describe behavior of the system by identifying the states. Define the transition between the states.
7. Avoid common errors while specifying the control.

2.19 Data Dictionary

The data dictionary can be defined as an organized collection of all the data element of the system with precise and rigorous definitions so that user and system analyst will have a common understanding of inputs, outputs, components of stores and intermediate

- The data models are less detail hence there is a need for data dictionary.
- Data dictionaries are lists of all of the names used in the system models.
- Descriptions of the entities, relationships and attributes are also included in data dictionary.
- Typically, the data dictionaries are implemented as a part of structured analysis and design tool
- The data dictionary stores following type of information

Name	Description
Name	The primary name of data or controlthe data store or item.externalentity.
Alias	Other name used for the Name
Where-used or how is It describes where the data or control item is IJsed. It also useddescribes item is used(that means input to the process. outputhow that process) to the	

Data construct	Notation	Meaning
Composition	=	Is composed of
Sequence	+	And
Selection	[I]	Or

Repetition	{ }"	Repetition for n times
	()	Optional data
	* *	Commented Information

For example:

Consider the DFD drawn for reservation system discussed example 1 of section 318.2. The data item "passenger" can be entered in the data dictionary as

```
name:                   passenger
alias:                  none
where used/
how used :              passenger's query (input)
                        ticket (output)
description :
passenger =             passenger_name +
                        passenger_address
passenger_name =        passenger_last_name +
                        passenger_first name +
                        passenger_middle_initial
passenger_address =     local_address +
                        community_address +
                        zip_code
local_address =         house_number +
                        street_name +
                        (apartment_number)
community_address =     city_name +
                        [state_name | province_name]
```

- The data dictionary defines the data items unambiguously
- One can give the detailed description of data items using data dictionary
- For large computer based system the size of data dictionary is very huge. It is also complex to maintain such a data dictionary manually. Hence automated (CASE) tools can be used to maintain the data dictionary.

Advantages:

1. Data dictionary support name management and avoid duplication

2. It is a store of organisational knowledge linking analysis, design and implementation.

Review Questions

1. Explain the term: Requirement engineering.
2. Give different types of requirements.
3. Explain the functional requirements.
4. What is non functional requirement? What are the different types of non functional requirements?
5. What do you mean by user requirements?
6. Explain the requirement engineering process.
7. What is the need for feasibility study? What is the outcome of feasibility study?
8. Write a short note on FAST.
9. What are :he requirement validation techniques?
10. What is software Prototyping? What are its benefits?
11. Compare evolutionary prototyping and throwaway prototyping.
12. What is the necessity of SRS?
13. Describe the structure of SRS?
14. What are the characteristics of SRS?
15. Explain the analysis modeling approach
16. What is Cardinality and modality?
17. Explain the entity Relationship Diagram with some suitable example.

Unit – III

Design concepts and Principles

3.1 Introduction

Design meaningful activity needed to develop a quality product. Design is the only way by which we can accurately translate the customer's requirements into a finished software product or system. Thus design serves as the basis for all the software engineering steps. In this chapter we will get introduced with the systematic approach to design process. We will discuss architectural design, user interface design and real time software design.

3.1.1 Analysis and Design Model

- After analyzing and specifying all the requirements the process of software design begins. Each of the elements of analysis model is used to create the design model.

- The elements of analysis model are

1. Data Dictionary

2. Entity Relationship diagram

3. Da ta flow diagram

4. State transition diagram

5. Control specification

6. Process Specification

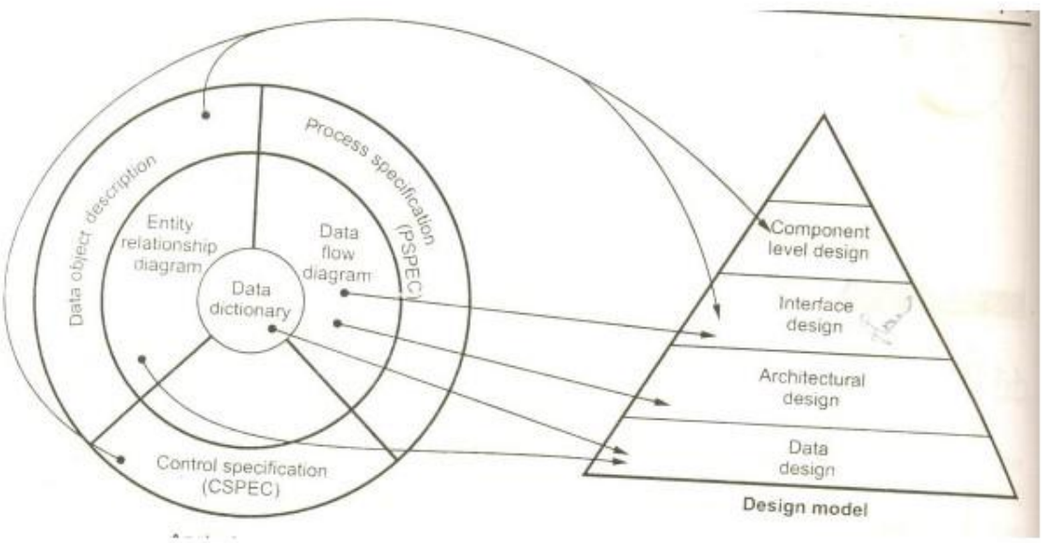
- The elements of design model are

  Data design

  Architectural Design

  Interface design

  Component-level design.



- The data design is used to transform the information domain model of analysis phase into the data structures. These data structures play an important role in software implementation. The entity relationship diagram and data dictionary are used to create the data design model. In entity relationship diagram the relationships among the data objects is defined and in data dictionary detailed data contents are given, Hence EI~D and data dictionary are used to implement the, data design.

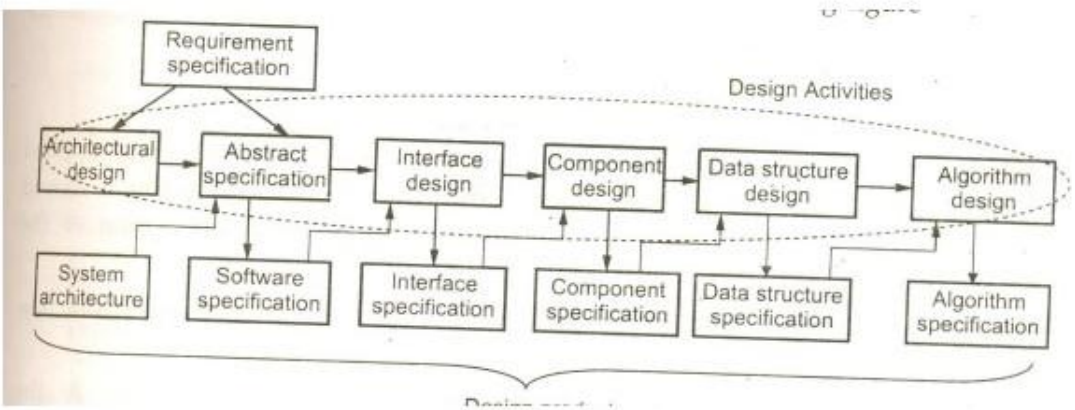
- The architectural design is used to represent the relationship between major structural elements with the help of some “design patterns". Hence data flow diagrams from analysis mod I serve as the basis for architectural design.

- The interface design' describes how software interacts within itself. An interface means flow of information, and specific type of behavior. Hence by using the data flow and control flow diagrams the interface design can be modeled,

- In the 'Component-level design' the structural elements of software architecture into procedural description of software componetes. Hence „component-level design" can be obtained using State Transition Diagram (STD), control Specification (CSPEC) and process Specification (PSPEC).

3.2 Design Process

- Design process is a sequence of steps carried through which the requirement are translated into a system or software model.



Design products

1. In architectural design the subsystem components can be identified.

2. The abstract specification is used to specify the subsystems.

3. The interfaces between the subsystems are designed, which is called interface

4. design.

5. In component design of subsystems components is done ..

6. The data structure is designed to hold the data

7. For performing the required functionality, the appropriate algorithm is designed.

3.2.1 Design Principle

Davis suggested a set of principles for software design as:

- The design process should not suffer from "tunnel vision",
- The design should be traceable to the I1nalysis model,
- The design should not reinvent the wheel.
- The design should "minimize the intellectual distance" between the software and the problem in the real world.
- The design should exhibit uniformity and integration.
- The design should be structured to accommodate change.
- The design should be structured to degrade gently.
- Design is not coding.
- The design should be assessed for quality.
- The design should be reviewed to minimize conceptual errors.

3.2.2 Design concept

The software design concept provides a framework for implementing the right software.

Following issues are considered while designing the software.

1. Abstraction

At each stage of software design process levels of abstractions should be appl to refine the software solution. At the higher lever of abstraction, the solution sho be stated in broad terms and in the lower level more detailed description of solution is given.

While moving through different levels of abstraction the procedural abstract and data abstraction are created.

In procedural abstraction it gives the named sequence of instructions in specific function.

In data abstraction the collection of data objects is represented.

Modularity

2. Modularity

- The software is divided into separately named and addressable component~ that called as modules. Creating such modules bring the modularity ir software.
- Meyer defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system:

Modular decomposability: A design method provides a systematic mechanism f(decomposing the problem into sub-problems. This reduces the complexity of tr problem and the modularity can be achieved.

Modular composability: A design method enables existing design components t be assembled into a new system.

Modular understandability: A module can be understood as a standalone unil Then it will be easier to build and easier to change.

Modular continuity: Small changes to the system requirements result in change~ to individual modules, rather than system-wide changes.

Modular protection: An aberrant condition occurs within a module and its effects are constrained within the module.

3. Refinement

- Refinement is actually a process of elaboration.
- Stepwise refinement is a top-down design strategy proposed by Niklaus WIRTH.
- The architecture of a program is developed by successively refining levels of procedural detail.
- The process of program refinement is analogous to the process of refinement and partitioning that is used during requirements analysis.
- Abstraction and refinement are complementary concepts. The major difference is that - in the abstraction low-level details are suppressed.

Refinement helps the designer to elaborate low-level details.

3.3Modular Design

Modular design reduces complexity and helps in easier 1mplementation. The parallel development of different parts of the system is possible due to modular design.

What is the benefit of modular design?

Changes made during testing and maintenance becomes manageable and they do not affect other modules.

3.3.1 Functional Independence

- The functional independence can be achieved by developing the functional modules with single-minded approach.
- By using functional independence functions may be compartmentalized and interfaces are simplified.
- Independent modules are easier to maintain with reduced error propagation.
- Functional independence is a key to good design and design is the key to software quality.
- The major benefit of functional independence is in achieving effective modularity.

3.3.2 Cohesion

- With the help of cohesion the information hiding can be done.
- A cohesive module performs only "one task" in software procedure with little -interaction with other modules. In other words cohesive module performs only one thing.
- Different types of cohesion are:

1. **Coincidentally cohesive** - The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.

2. **Logically cohesive** - A module that performs the tasks that are logically related with each other is called logically cohesive.

3. **Temporal cohesion** - The module in which the tasks need to be executed in some specific time span is called temporal cohesive.

4. **Procedural cohesion** - When processing elements of a module are related with one another and must be executed in some specific order then such module is called procedural cohesive.

5. **Communicational cohesion** - When the processing elements of a module share the data then such module is communicational cohesive.

- The goal is to achieve high cohesion for modules in the system.

3.3.3 Coupling

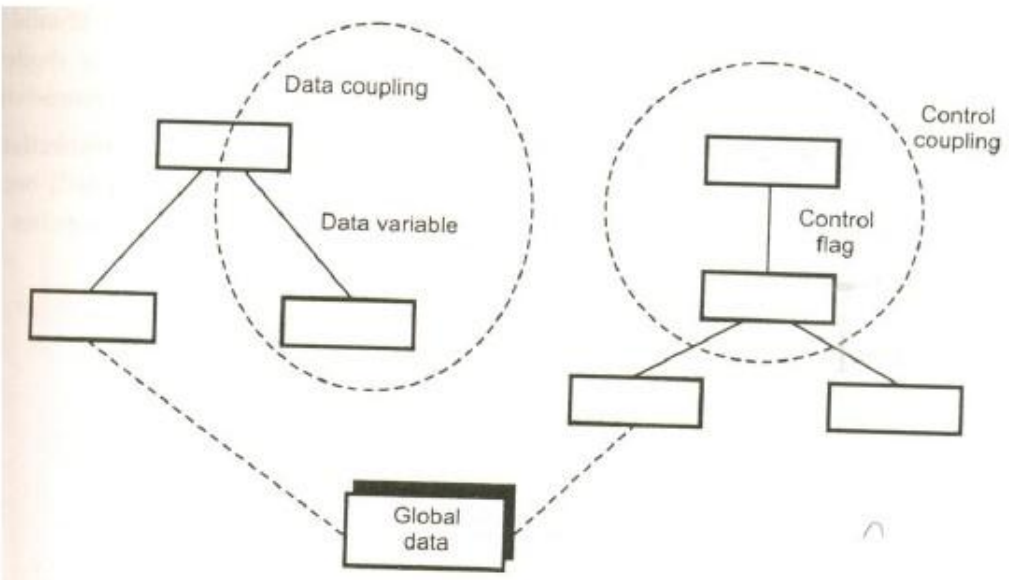
- Coupling effectively represents how the modules can be "connected" with other module or with the outside world.
- Coupling is a measure of interconnection among modules in a program structure.
- Coupling depends on the interface complexity between modules.
- The goal is to strive for lowest possible coupling among modules in software design.
- The property of good coupling is that it should reduce or avoid change impact and ripple effects. It should also reduce the dost in program changes, testing, and maintenance.
- Various types of coupling are:

I. **Data coupling** - The data coupling is possible by parameter passing or dllta interaction.

II. **Control coupling** - The modules share related control data in control coupling.

III. **Common coupling** - In common coupling common data or a global dah\ is shared among the modules,

IV. **Content coupling** – Content coupling occurs when one module makes use of data or control information maintained in another module.



3.4 Design Heuristics

The program structure can be manipulated according the design heuristics as shown below.

1. **Evaluate the first iteration of the program structure to reduce the coupling and improve cohesion** - The module independency can be achieved in the program structure by exploding or imploding the modules. Exploding the modules means obtaining two or more modules in the final stage and imploding the modules means combining the result of different modules.

2. **Attempt to minimize the structures with high fan-out and strive for fan-in as depth increases** - At the higher level of program structure the distribution of control should be made. Fan-out means number of immediate subordinates to the module, Fan-in means number of immediate ancestors the module have.

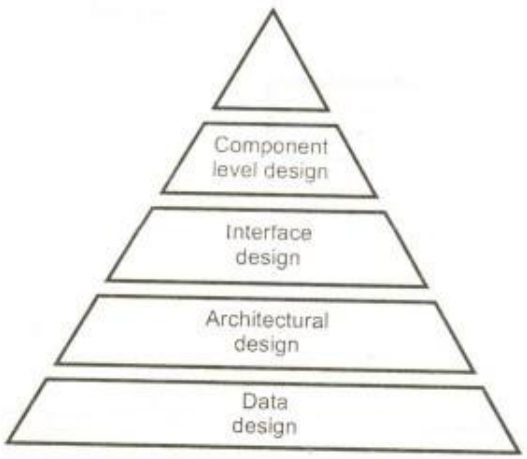
3. **Keep scope of the effect a module within the scope of control of that module** - The decisions made in particular module 'a' should not affect the module 'b' which lies outside the scope of module 'a'.

4. **Evaluate the module interfaces to reduce complexity and redundancy and improve consistency** ~ Mostly the cause of software error is module interfaces. The module interfaces should simply pass the information and should simple pass the information and should be consistent with the module.

5. **Define module whose function is predictable but avoid modules that are too restrictive** – The module should be designed be designed with simplified internal

processing so that expected data can be produced as a result. The modules should no restrict the size of local data structure, options with control flow or modes 0 external interfaces. Being module too much restrictive causes large maintenance.

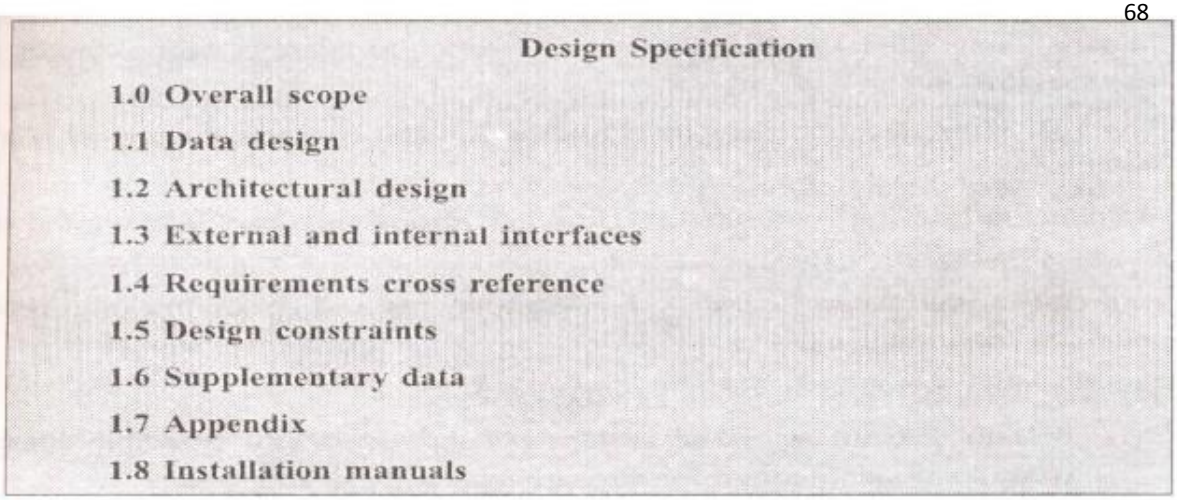
6. **Strive for controlled entry modules by avoiding pathological connections -** Software interfaces should be constrained and controlled so that it will become manageable. Pathological connection means many references or branches into the middle of a module.



- The design model is represented as pyramid. The pyramid is a stable object. Representing design model in this way means that the software design should be stable.
- The design model has broad foundation of data design, stable mid-region with architectural and interface design and the sharp point to for component level design.
- The design model represents that the software which we create should be stable such that any changes should not make it collapsed. And from such a stable design a high quality software should be generated.

3.6 Design Document

The design document can be created as follows



- The design document is used to represent various aspects of design model.
- In this document first of all overall scope of the design effort is described. The information presented here is used from the SRS.
- Then in data design database structure, any external file structure, internal data structure, cross reference of data objects to files is defined.
- The architectural design shows how analysis model builds the program architecture. Sometimes structure charts are used to represent the module hierarchy.
- Then internal and external program interfaces are given. In some cases a detailed prototype of a CUI may be represented.
- The requirement cross reference is given in order to ensure that all requirements are satisfied by the software design. The cross references also indicate which component are critical for implementation. The test documentation is also included in the design document.
- Under design constraints the information such as memory requirements, special requirement for assembling or packaging the software, requirement of virtual memory, high speed requirement is given.
- The final section of design document contains information about supplementary data if any required by the system.
- The appendix includes algorithmic descriptions, tabular data or any other relevant information.
- Finally the design document should contain some user manual or installation manual.

3.7 Architectural Design

- The architectural design is the design process for identifying the subsystems making up the system and framework for subsystem control and communication.
- The goal of architectural design is to establish the overall structure of software system.
- Architectural design represents the link between design specification and actual design process.
- In architectural design logical system components and communication between them are identified.
- The common activities in design process are

1. System structuring - The system is subdivided into principle subsystems components and communications between these subsystems are identified.

2. Control modeling - A model of control relationships between different parts of the system is established.

3. Modular decomposition - The identified subsystems are decomposed into modules.

3.8 Software Architecture

The software architecture gives the hierarchical structure of software components and their interactions. In software architecture the software model is designed and structure of that model is partitioned horizontally or vertically.

Each model represents different perspective on the architecture for instance:

1. Structural model - Represents architecture as an organized collection of components.

2. Framework model - Identifies the repeatable architectural design frameworks and thereby increases the level of abstraction.

3. Dynamic model - Represents the behavior aspects of the program architecture. Behavior means change in functioning of the system on occurrence of external event.

4. Process model - Focuses on the design of the business or technical process.

5. Functional model - It can be used to represent functional hierarchy of the system.

3.8.1 Structural Partitioning

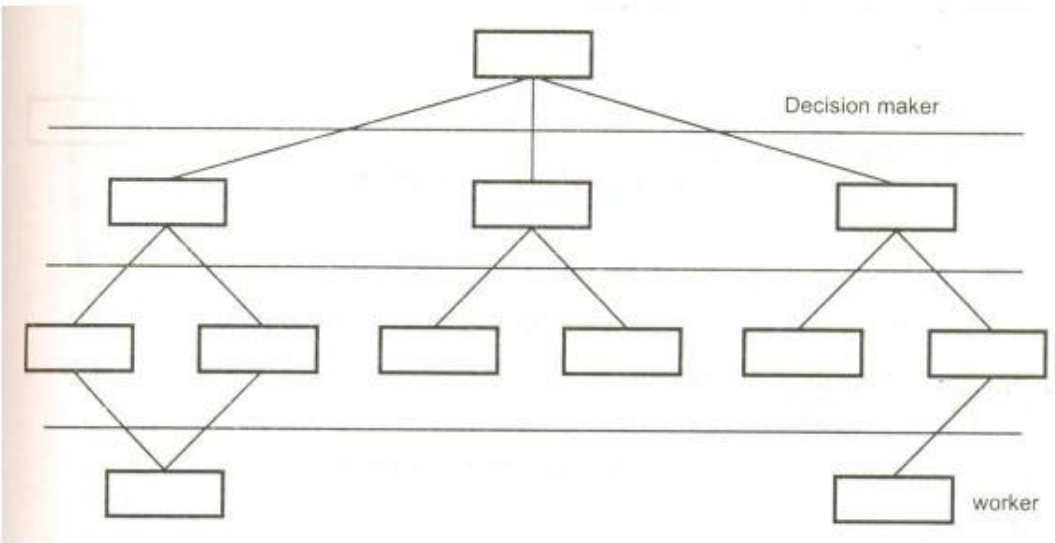
The program structure can be partitioned horizontally or vertically.

Horizontal partitioning

Horizontal partitioning defines separate branches of the modular hierarchy for each major program function.

Horizontal partitioning can be done by partitioning system into: input, data transformation (processing), and output.

In horizontal partitioning the design making modules are at the top of the architecture.



Advantages of horizontal partition are

1. These are easy to test, maintain, and extend

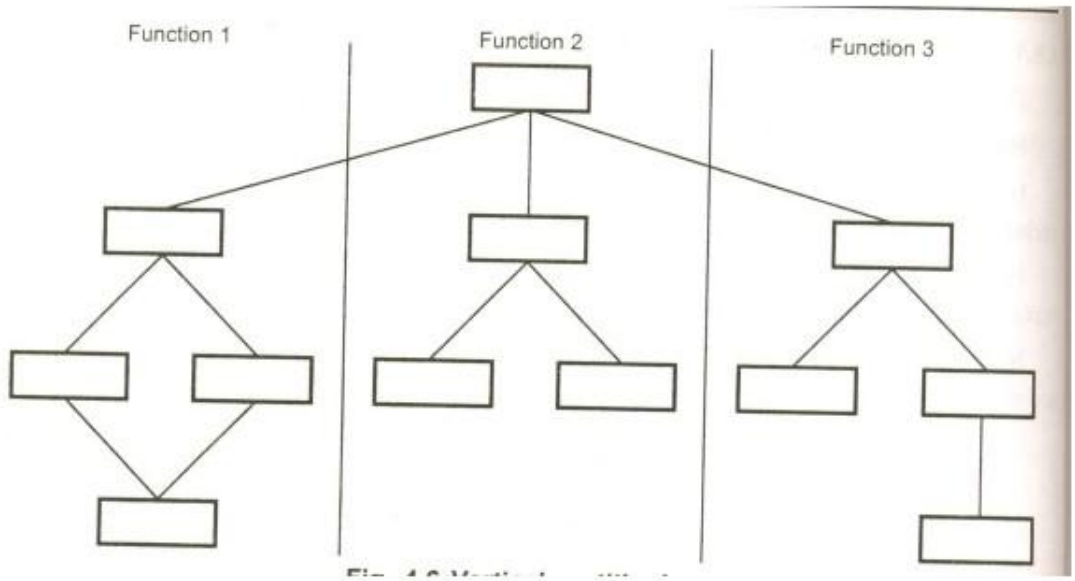
2. They have fewer side effects in change propagation or error propagation

Disadvantage of horizontal partition:

More data has to be passed across module interfaces which complicate the overall control of program flow.

Vertical partitioning

Vertical partitioning suggests the control and work should be distributed top-down in program structure.



In vertical partitioning

- Define separate branches of the module hierarchy for each major function.
- Use control modules to co-ordinate communication between functions.

Advantages of vertical partition:

1. These are easy to maintain the changes.
2. They reduce the change impact and error propagation.

3.9 Data Design

- Data design is basically the model of data that is represented at the high level of abstraction.
- The data design is then progressively refined to create implementation specific representations.
- Various elements of data design are
  - Data object - The data objects are identified and relationship among various data objects can be represented using entity relationship diagrams or data dictionaries.
  - Databases - Using software design model, the data models are translated into data structures and databases at the application level.

- Data warehouses - At the business level useful information is identified from various databases and the data warehouses are created. For extracting or navigating the useful business information stored in the huge data warehouse then data mining techniques are applied.

Guideline for data design

1. Apply systematic analysis on data

   Represent data objects, relationships among them and data flow along with the contents.

2. Identify data structures and related operations

   For the design of efficient data structures all the operations that will be performed on it should be considered.

3. Establish data dictionary

   The data dictionary explicitly represents various data objects, relationships among them and the constraints on the elements of data structures.

4. Defer the low-level design decisions until late in the design process

   Major structural attributes are designed first to establish an architecture of data. And then low-level design attributes are established.

5. Use information hiding in the design of data structures

   The use of information hiding helps' in improving quality of software design. It also helps in separating the logical and physical views.

6. Apply a library of useful data structures and operations

   The data structures can be designed for reusability. A use of library of data structure templates (called as abstract data types) reduces the specification and design efforts for data.

7. Use a software design and programming language to support data specification and abstraction

   The implementation of data structures can be done by effective software design and by choosing suitable programming language.

3.10 Architectural Style

- The architectural model or style is a pattern for creating the system architecture for given problem. However, most of the large systems are heterogeneous and do not follow single architectural style.

- System categories define the architectural style

1. Components: They perform a function.

   For example: Database, simple computational modules, clients, servers and filters.

2. Connectors: Enable communications. They define how the components communicate, co-ordinate and co-operate.

   For example Call, event broadcasting, pipes

3. Constraints: Define how the system can be integrated.

4. Semantic models: Specify how to determine a system's overall properties fJ the properties of its parts.

- The commonly used architectural styles are

   1. Data centered architectures

   2. Data flow architectures

   3. Call and return architecture

   4. Object oriented architectures

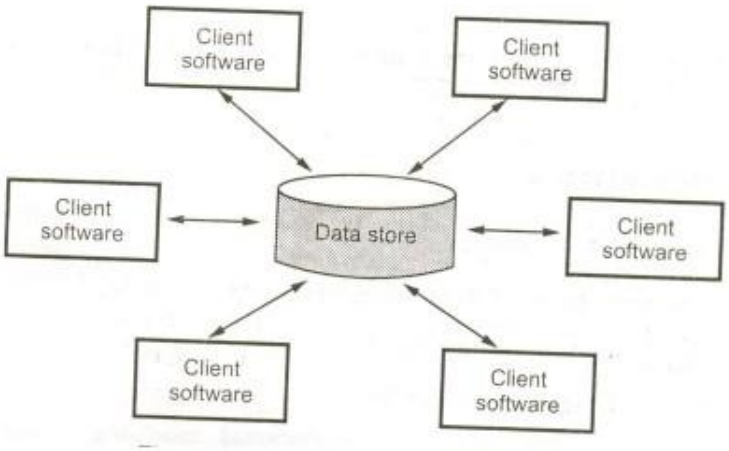
   5. Layered architectures

3.10.1 Data Centered Architectures

In this architecture the data store lies at the centre of the architecture and other components frequently access it by performing add, delete, and modify operatic. The client software requests for the data to central repository. Sometime the client software accesses the data from the central repository without any change in data without any change in actions of software actions.

Data centered architecture posses the property of interchangeability Interchangeability means any component from the architecture can be replaced b; new component without affecting the working of other components.

In data centered architecture the data can be passed among the components.



In data centered architecture

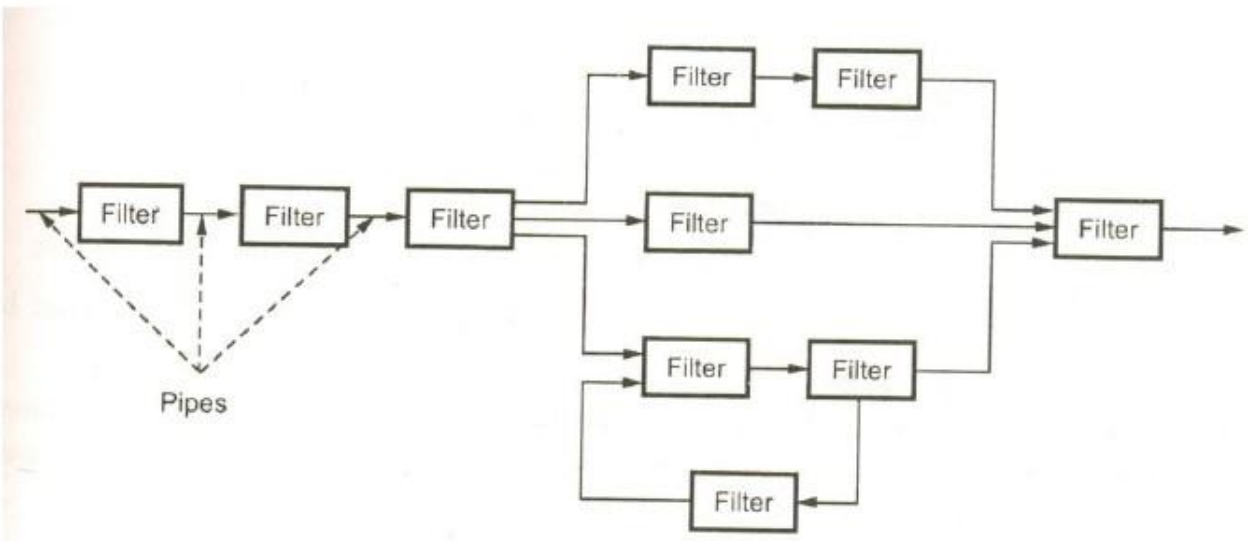
Components are: Database elements such as tables, queries.

Communication are: By relationships

Constraints are: Client software has to request central data store for information.

3.10.2 Data Flow Architectures

In this architecture series of transformations are applied to produce the output data. The set of components called filters are connected by pipes to transform the data from one component to another. These filters work independently without a bothering about the working of neighboring filter.



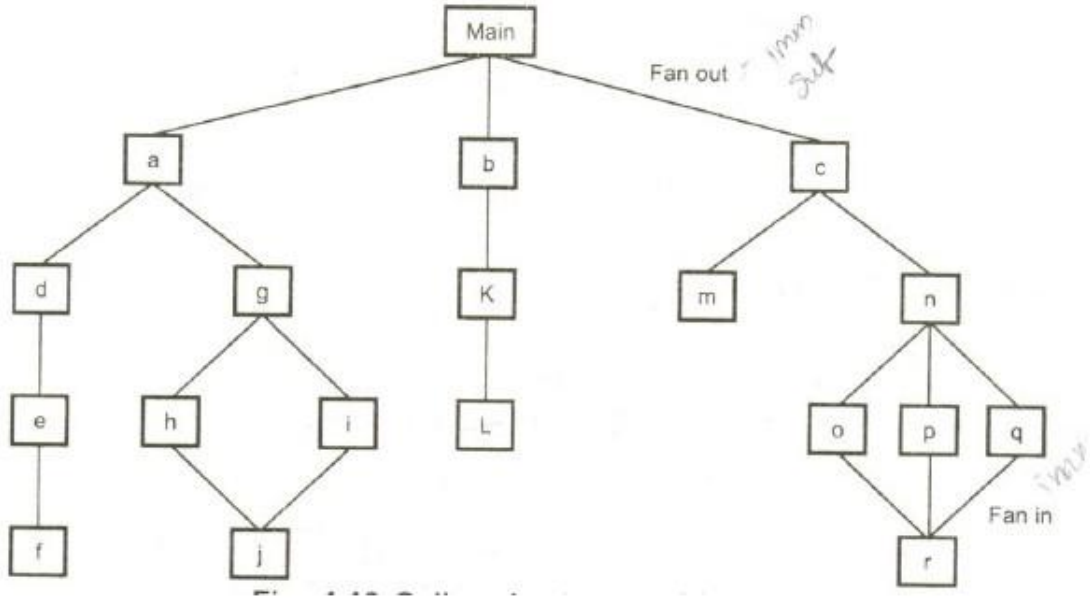
If the data flow degenerates into a single line of transforms, it is termed as batch sequential.

In this pattern the transformation is applied on the batch of data.

4.10.3 Call and Return Architecture

The program structure can be easily modified or scaled. The program structure j organized into modules within the program. In this architecture how modules ca each other. The program structure decomposes the function into control hierarch where a main program invokes number of program components.

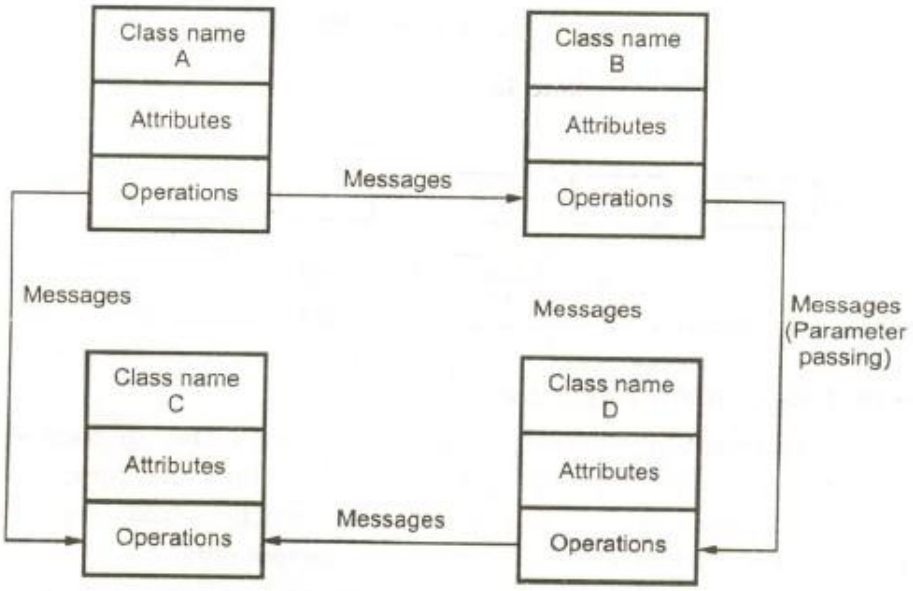
In this architecture the hierarchical control for call and return is represented.



3.10.4 Object Oriented Architecture

In this architecture the system is decomposed into number of interacting objects These objects encapsulate data and the corresponding operations that must applied to manipulate the data.

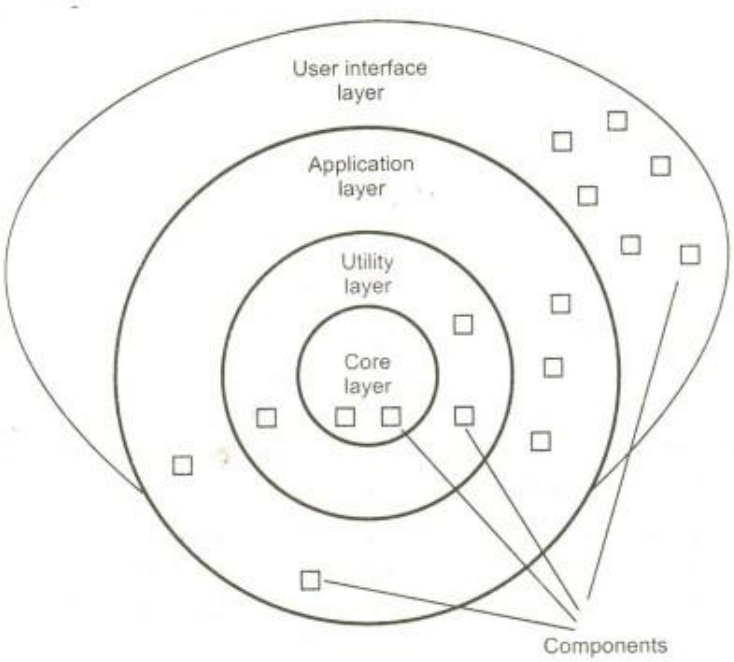
The object oriented decomposition is concerned with identifying objects cla~ their attributes and the corresponding operations. There is some control models 1 to co-ordinate the object operations.



3.10.5 Layered Architecture

- The layered architecture is composed of different layers. Each layer is intended to perform specific operations so machine instruction set can be generated. Various components in each layer perform specific operations.

- The outer layer is responsible for performing the user interface operations while the components in the inner layer perform operating system interfaces.



- The components in intermediate layer perform utility services and application software functions.

3.11 Transform and Transaction Mapping

3.11.1Transform Mapping

The transform mapping is a set of design steps applied on the OFO in order to map the transformed flow characteristics into specific architectural style.

Design steps for transform mapping

We will consider an example of security home system and apply the design steps for performing transform mapping.

Example: Home security system

"Security system software is prepared for the homeowner for home security purpose. After installation of this system it needs to be configured. This system has control panel through which the home owner can interact with it using keypad an functional keys. The sensors are connected to the system and to monitor the status 0 sensors.

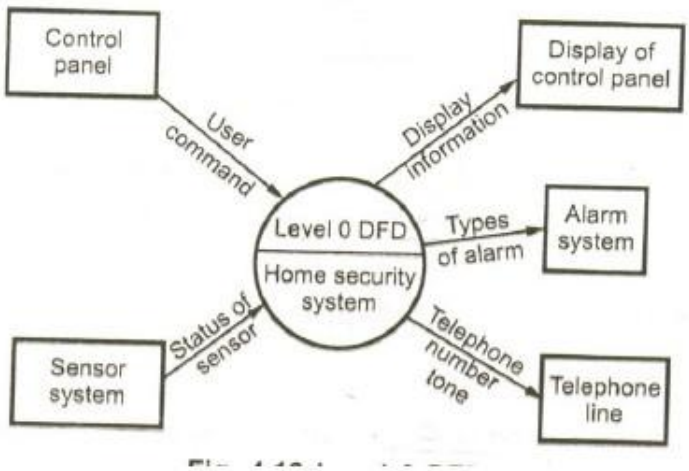
While installing the security system software the control panel is used to progra and configure the system. During configuration each sensor is assigned a number and type, a master password is programmed for alarming and de-alarming the system. The telephone numbers of emergency services are programmed in the system as input for dialing when a sensor event occurs.

On occurrence of sensor event, it is recognized first and then an alarm which ~ attached to the system starts ringing. After a delay time (which is specified by the homeowner during the configuration) the software dials the telephone number, provides the information about the nature of event and its location.

The telephone number will be redialed every 20 seconds until telephone connection is obtained.

Step 1: Review the fundamental system model to identify the information flow

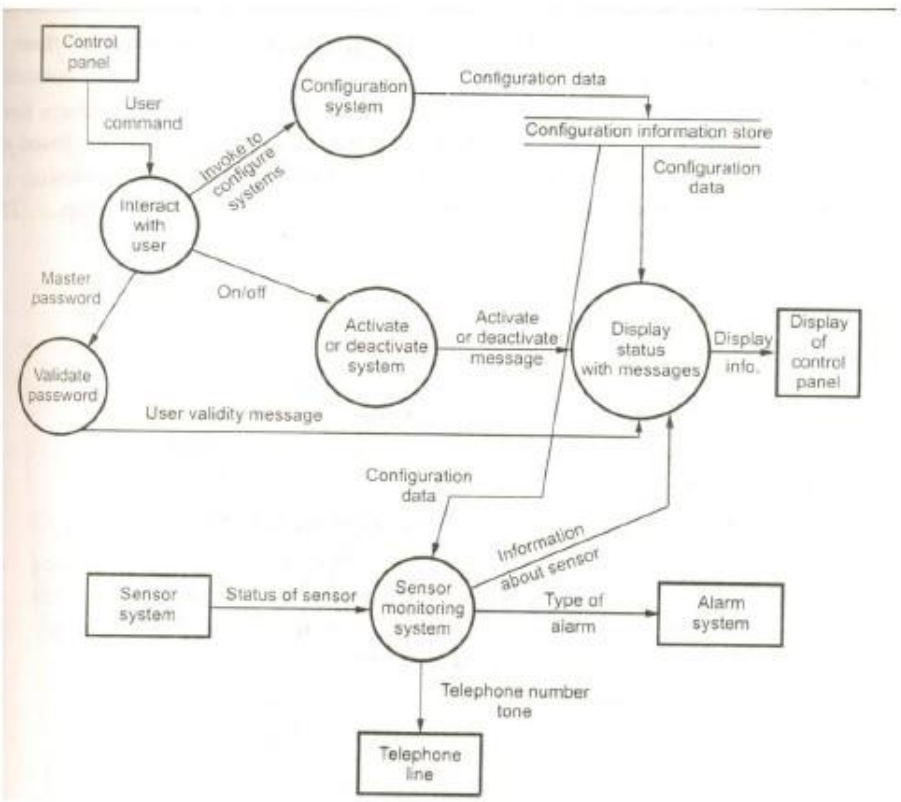
The fundamental system model can be represented by level 0 DFD and supporting information. This supporting information can be obtained from the two important documents called 'system specification' and 'software requirement specifications'. Both of them describe the information flow and structure at software interface.

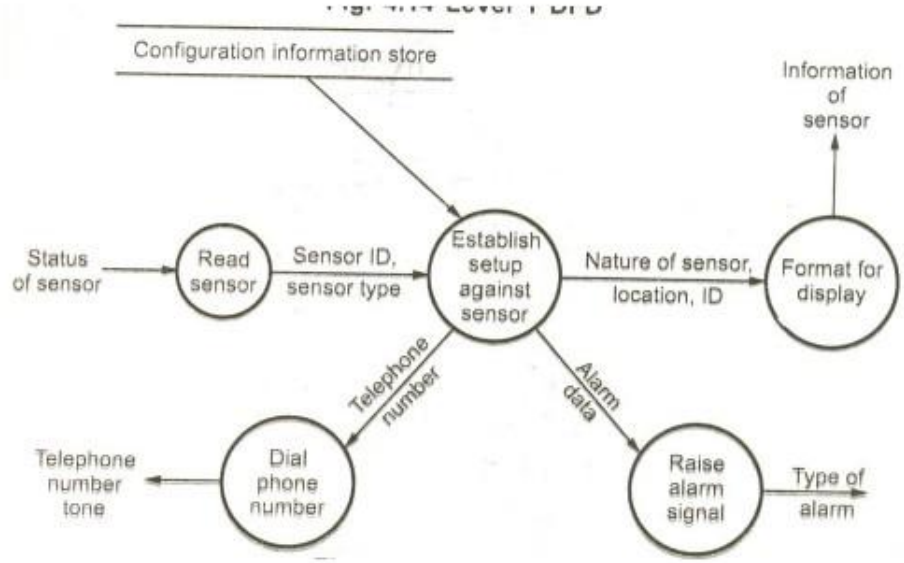


Step 2: Review and refine the data flow diagrams for the software -

The data flow diagrams are analyzed and refined into next higher levels. Each transform in the data flow diagrams impose relatively high level cohesion. That means after applying the certain transformation the process in the DFD performs a single distinct function.

For example the DFD is refined to level 1 to working of the system. Further the level 2 DFD is drawn in which the detailing of sensor monitoring system is done.





Step 3: Determine if the DFD has the transform or transaction flow Characteristics

The information flow within the system is usually represented as transform flow. However, there can be dominance of transaction characteristics in the DFD. Based on characteristics of the DFD the transformation flow or transaction flow is decided.

For example if we draw a level 3 DFD for the process of 'Establish setup ... ' .The transformation flow is identified.

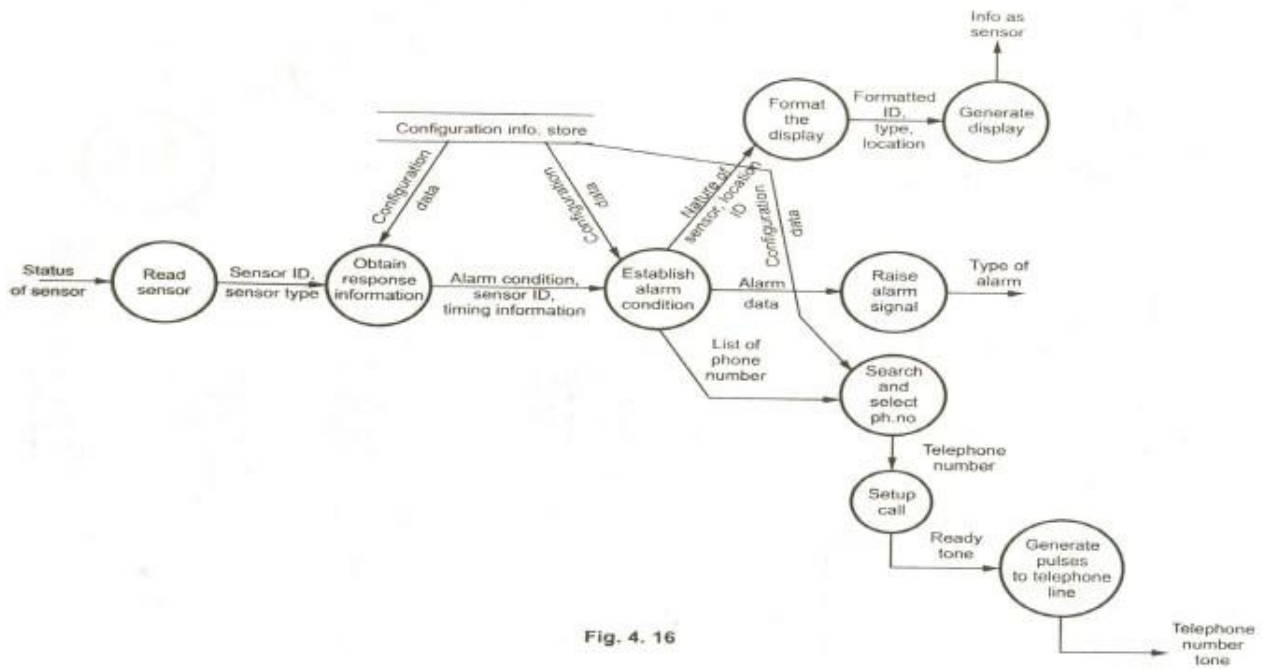


Fig. 4. 16

Step 4 : Isolate the transform centre by specifying incoming and outgoing flow boundaries.

In this design step, the reasonable boundaries are selected and it is avoided to apply lengthy iterations on placement of divisions. The incoming and outgoing flow boundaries are as shown in Fig. 4.17.

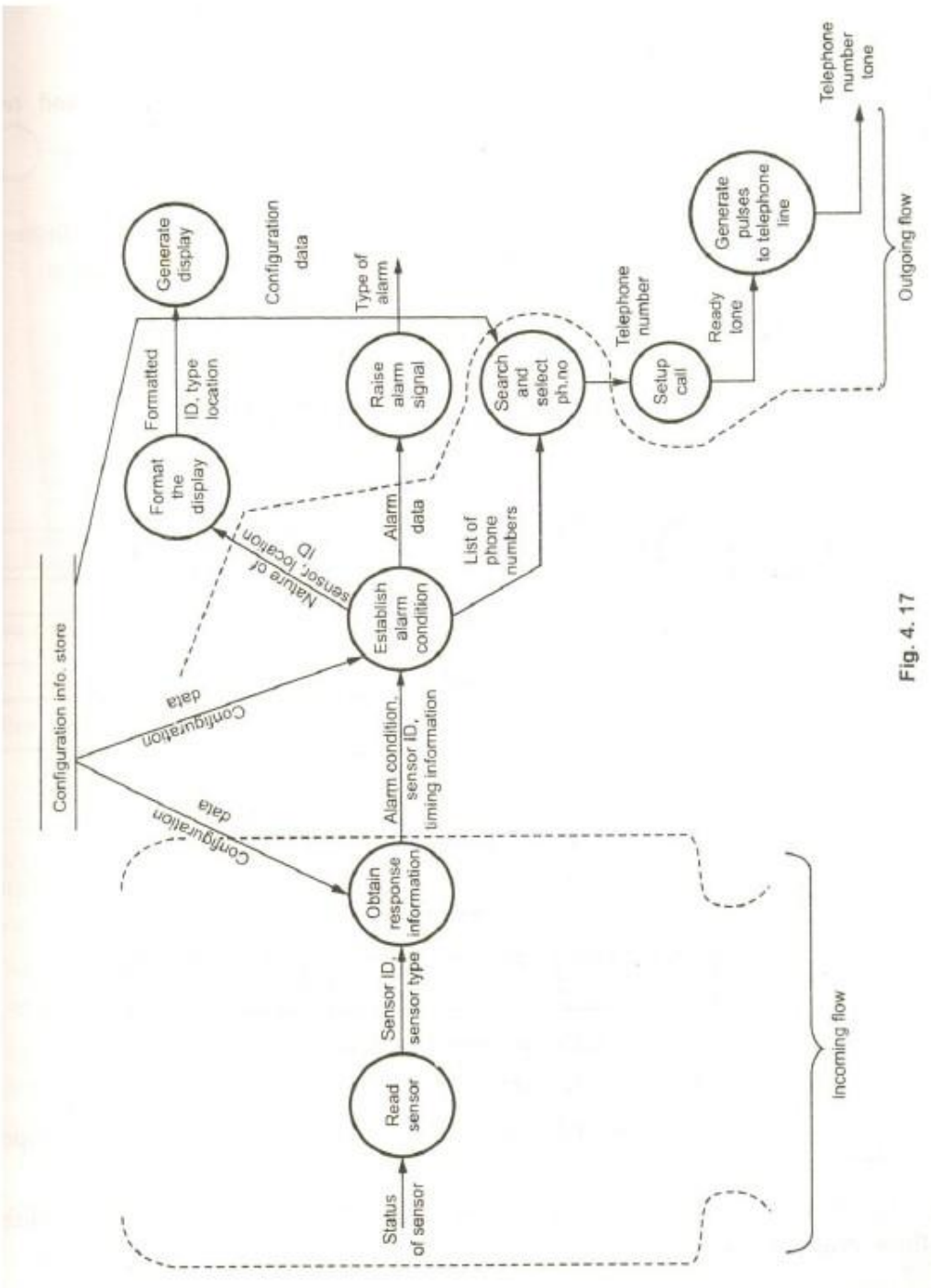

Configuration info. store
Generate display
Configuration data
Type of alarm
Formatted
ID, type location
Format the display
Raise alarm signal
Search and select ph.no
Telephone number
Setup call
Ready tone
Generate pulses to telephone line
Telephone number tone
Outgoing flow
Nature of sensor, location ID
Alarm data
List of phone numbers
Establish alarm condition
Configuration data
Alarm condition, sensor ID, timing information
Configuration data
Obtain response information
Sensor ID, sensor type
Read sensor
Status of sensor
Incoming flow

Fig. 4. 17

Step 5: Perform first level factoring

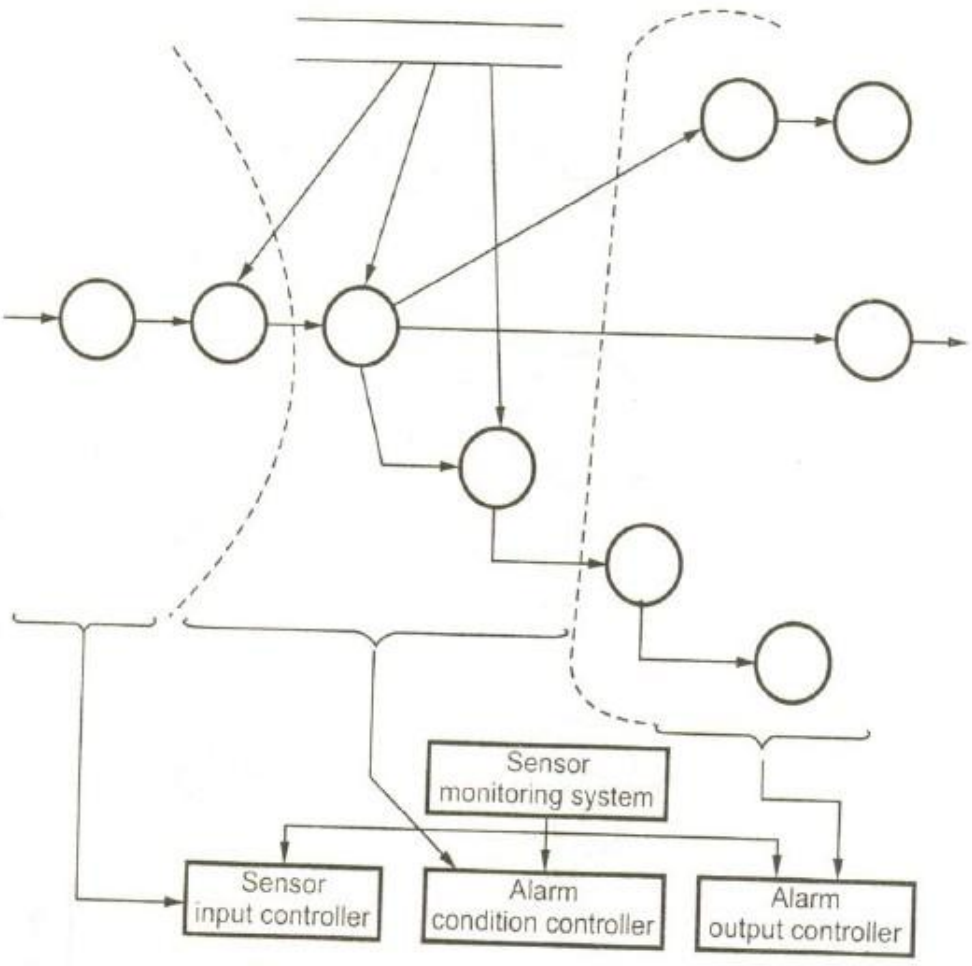
After performing the first level factoring the program structure results in top do architecture where

- Top level components perform decision making
- Low-level components perform most input, computation and output
- Middle-level perform some control and some amount of work

When transform flow is identified the DFD is mapped into call and return architecture.

For example

Sensor monitoring system becomes a top level component which co-ordinates t sensor input controller, alarm conditions controller and alarm output controller.

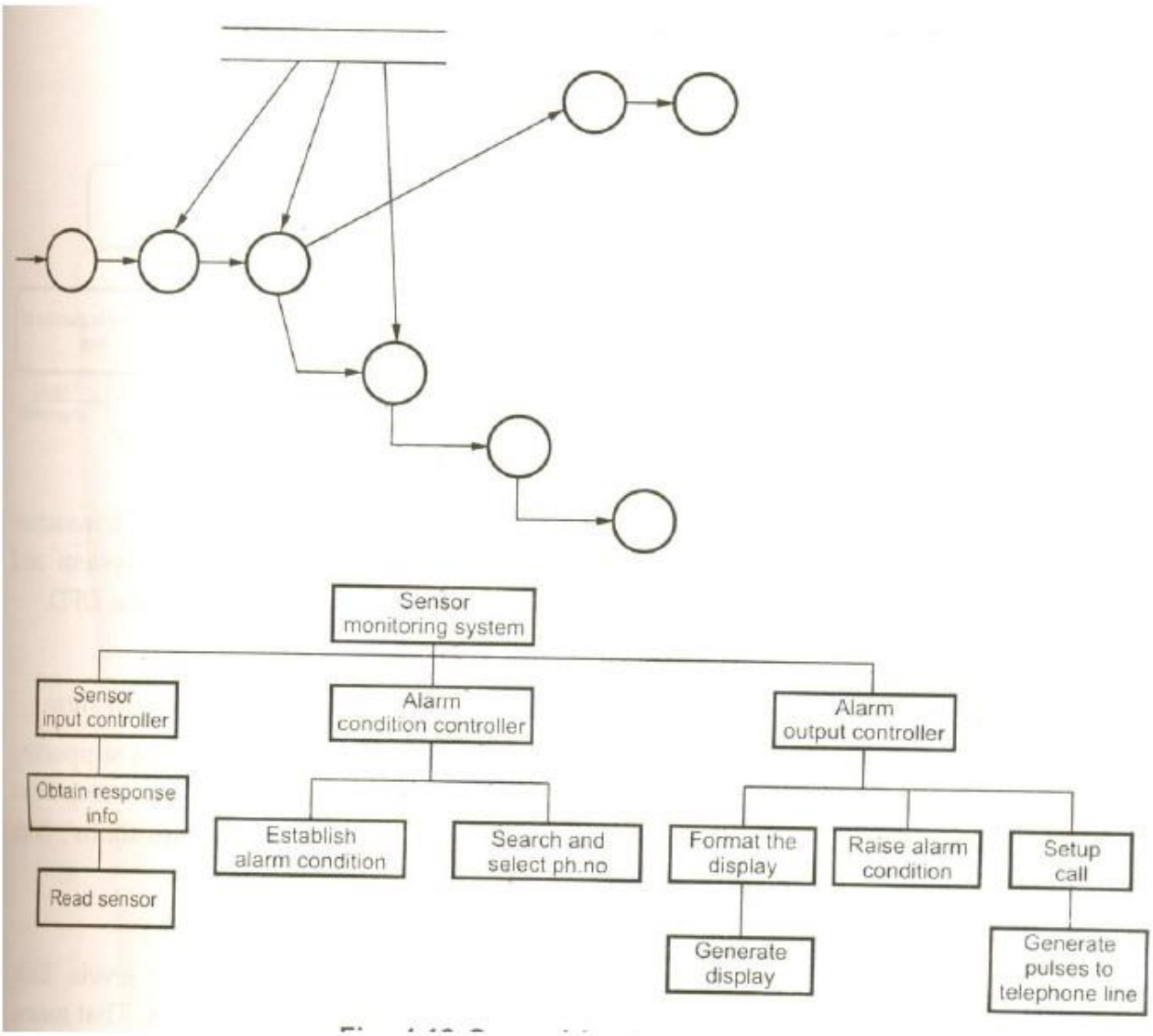


Step 6: Perform second level factoring

In the second level factoring individual bubble of DFD is mapped into appropriate module within architecture.

There could be one-to-one mapping of bubble of DFD into the software module or l" v VI' three bubbles can be combined together to form a single software module.

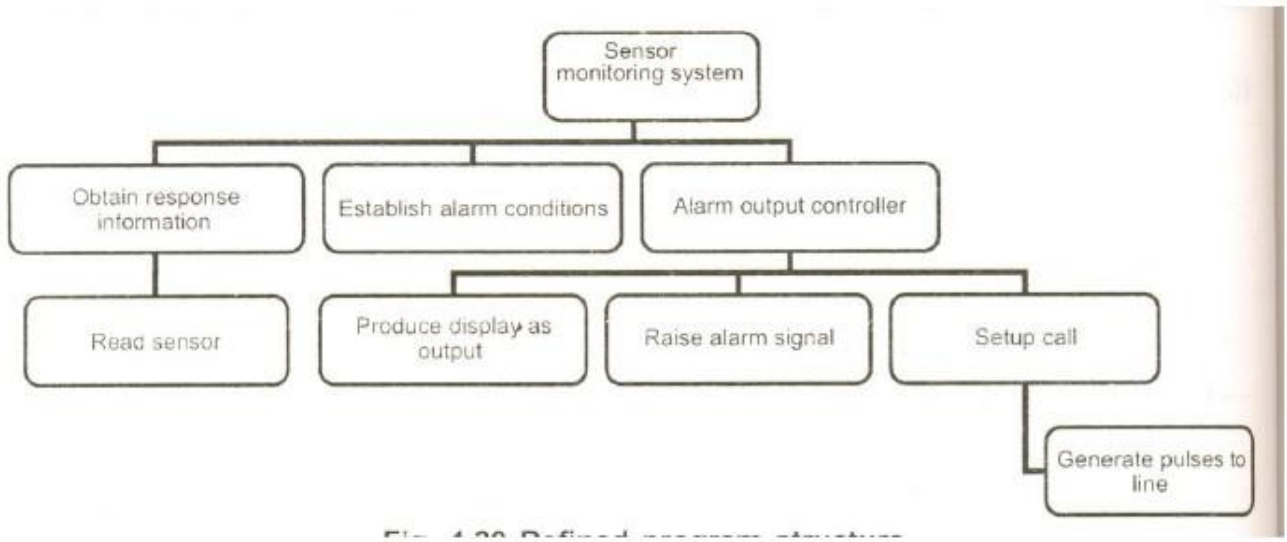
After performing the second level factoring the architecture serves as the first-iteriltion design.



Step 7: Refine the first-iteration architecture using design heuristics for improved software quality

The first-iteration architecture· can be refined by applying the module independency.

The modules can be exploded or imploded with high cohesion and minimum coupling. The refinement should be such that the structure can be implemented without difficulty, tested without confusion and can be easily maintained.



4.11.2 Transaction Mapping

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

Design steps for transaction mapping

Step 1: Review the fundamental system model to identify the information flow

The fundamental system model can be represented by level 0 DFD and supporting information. This supporting information can be obtained from the two important documents called 'system specification' and 'software requirement specifications'. Both of them describe the information flow and structure at software interface.

Step 2: Review and refine the data flow diagrams for the software

The data flow diagrams are analyzed and refined into next higher levels. Each transform in the data flow diagrams impose relatively high level cohesion. That means after applying the certain transformation the process in the DFD performs a single distinct function.

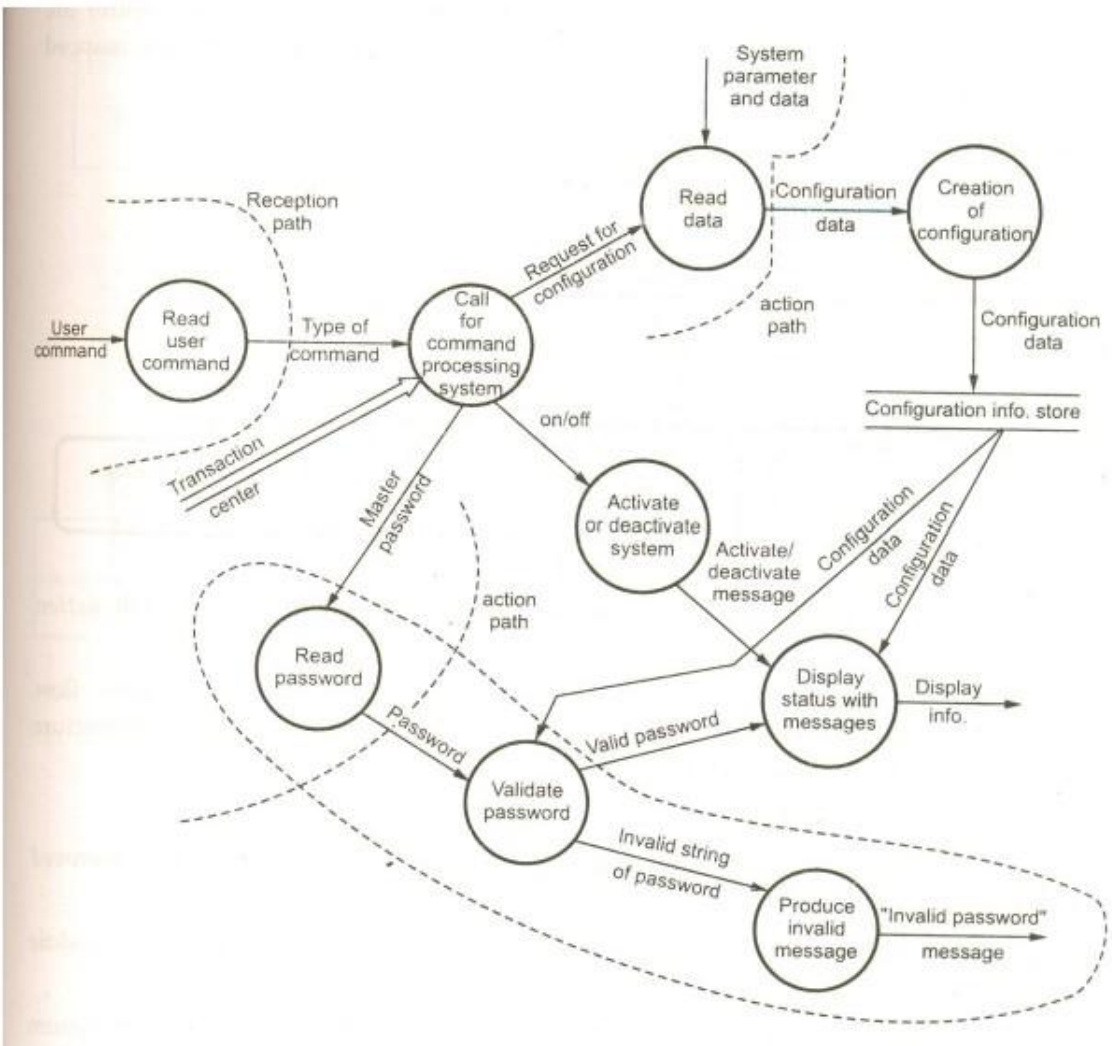
Step 3: Determine if the DFD has the transform or transaction flow Characteristics

The information flow within the system is usually represented as transform floW· However, there can be dominance of transaction characteristics in the DFD. Based 011 characteristics of the DFD the transformation flow or transaction flow is decided.

Step 4: Identify the transaction centre and flow characteristics along each of tire action paths

In transaction mapping we have to identify the, location of transaction centre. FrOlJ1 the transaction centre many action paths flow radially from it.

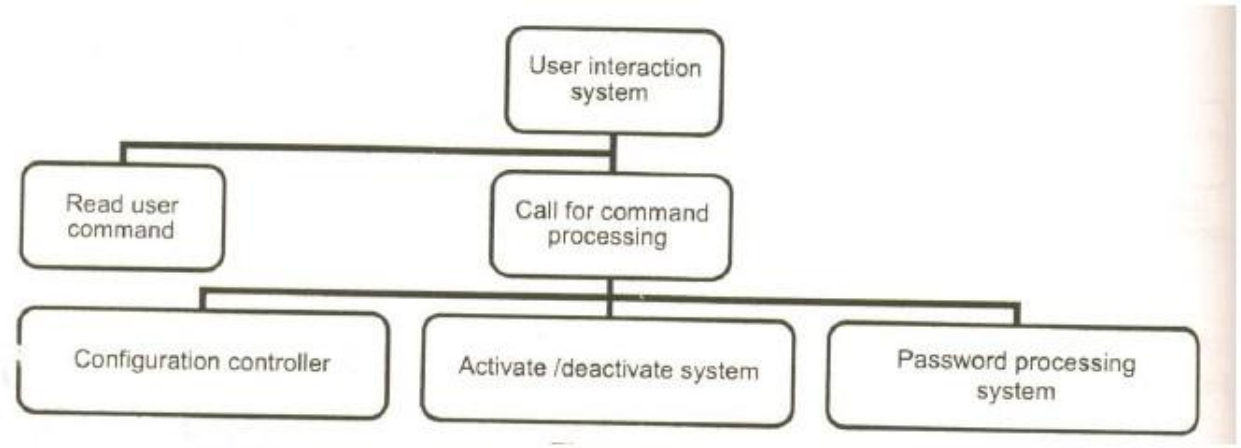
For example: As shown in following level 2 DFD the command processing centre serves as the transaction centre. The reception path and action paths are also shown.



Step 5 : Map DFD into transaction processing structure

The identified transaction flow is mapped into an architecture that contains an incoming branch and a dispatcher branch. Starting from the transaction centre the corresponding bubbles on incoming path (path corning to transaction centre) are mapped into the appropriate modules. The bubbles along the action path are mapped into the action modules.

For example



Step 6: Factor and refine the transaction structure and structure of each action path

Each action path of the data flow diagram has its own information flow characteristics. The action path related substructure is developed. This substructure serves as first iteration architecture.

See Fig. 4.23 on next page.

Step 7: Refine the first-iteration architecture using design heuristics for improved software quality

The first-iteration architecture can be refined by applying the module independency.

The modules can be exploded or imploded with high cohesion and minimum coupling. The refinement should be such that the structure can be implemented without difficulty, tested without confusion and can be easily maintained.

3.12 User Interface Design

- In user interface design the effective interaction of system with user is provided.
- Typically' system user judges a system by its interface rather than its functionality. A poorly designed interface leads improper use of the system.

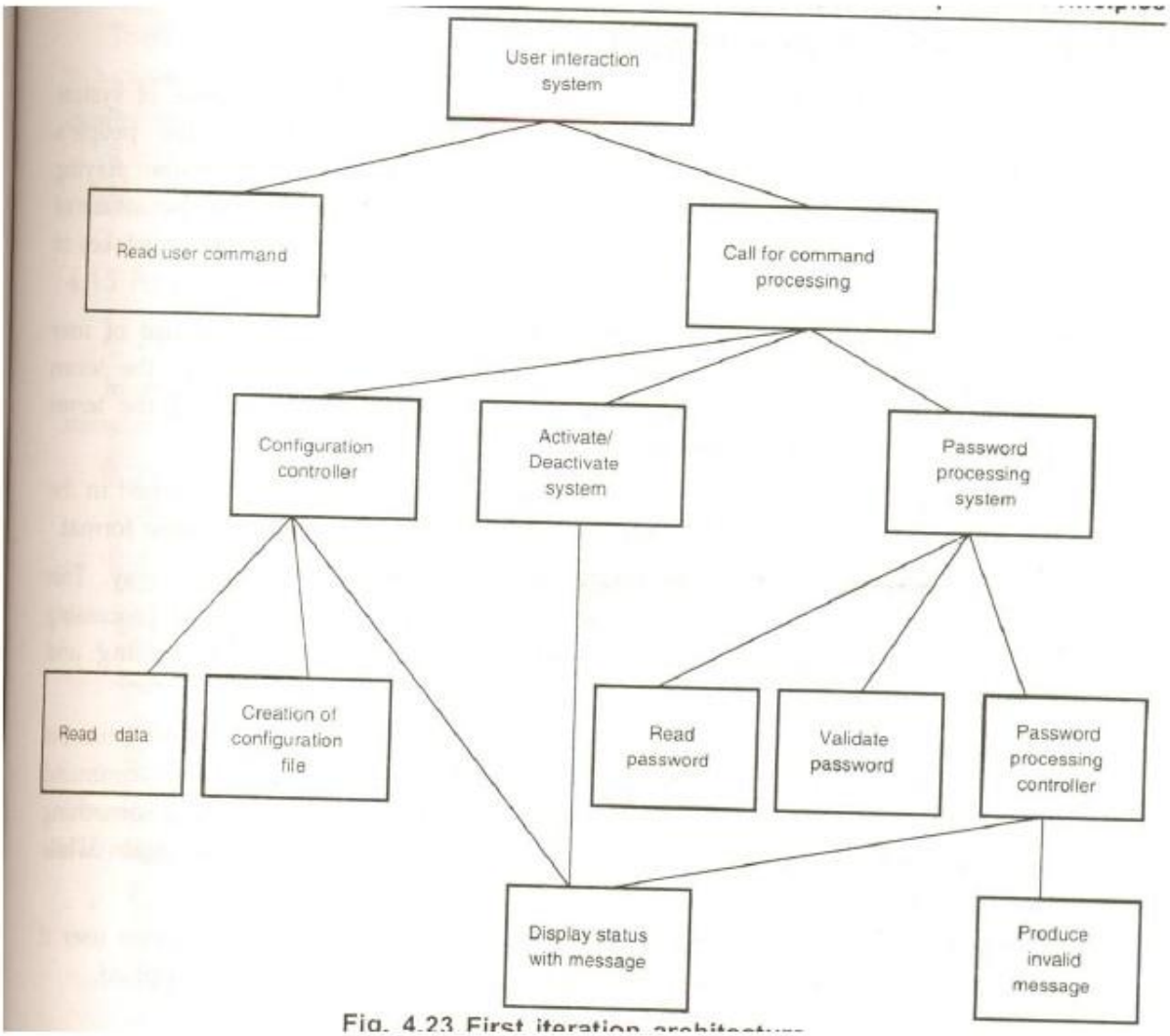


Fig. 4.23 First iteration architecture

Many software systems can not be used because of poorly designed user interface.

- Most of the users in business systems interact with the systems using graphical user interfaces. However text based interfaces is also used.

Advantages of Graphical User Interfaces (GUI)

1. GUTs are easy to use. An inexperienced user can use the system easily with the graphical user interface.

2. The user can switch from one task to another very easily. He can also interact with many applications simultaneously. The application information remains visible in its own window.

3. Fast and full screen interaction is possible with user.

3.13 User Interface Design Principles

While designing the user interface the capability, need and experience of system user should be considered. The designer should take into account the people's physical and mental limitations. For instance a short memory or in game playing software a child can recognize pictures rather than text. He should also be aware of the fact that people make mistakes and the interface should tolerate these mistakes in a friendly manner. Following design principles are used.

1. **User familiarity** - Instead of using computer terminology make use of user oriented terminologies. For example in 'Microsoft Office' software, the terms such as document, spreadsheet, letter, folder are used and use of the terms directory, file identities is avoided.

2. **Consistency** - The appropriate level of consistency should be maintained in the user interface. For example the commands or menus should be of same format.

3. **Minimal surprise** - The commands should operate in a known way. This makes user to easily predict the interface. For example the in word processing document under the tool menu there should be a facility of spelling and grammar checking.

4. **Recoverability** - The system should provide recovering facility to user from his errors so that user can correct those error. For example an undo command should be given so that user can correct his errors, or while deleting something the confirmation action must be provided, so that user can think again while deleting something.

5. **User guidance** - The user interface can be effectively used by a novice user if some user guidance such as help systems, online manuals etc are supplied.

3.14 Real Time Systems

- Behavior of real time systems is based on timing constraints. Real time systems are those systems which monitor and control their environment.
- These systems are usually associated with some hardware devices.
- There are two important elements of the real time system.

Sensors - They are responsible for collecting the data from the system.

Actuators - These are responsible to change the environment of the system. • Real time systems must respond within specified time.

Real time system is a software system in which the correct functionalities of the system are dependant upon results produced by the system and the time at which these results are produced.

There are two types of real time systems soft real time and hard time systems.

A soft real time systems are those systems in which the operations are degraded if results are not produced in specified timing requirements.

A soft real time systems are those systems in which the operations are incorrect if the results are not produced in specified timing requirements.

3.15 Real Time Software Design

- Stimulus response system -

In real time systems, there is one class of the system in which on giving the stimulus the system produces the response. The stimuli can be of two types

I. Periodic stimuli - These are the stimuli that occur at periodic time interval. For example - pressure sensor may be activated after every 20 seconds.

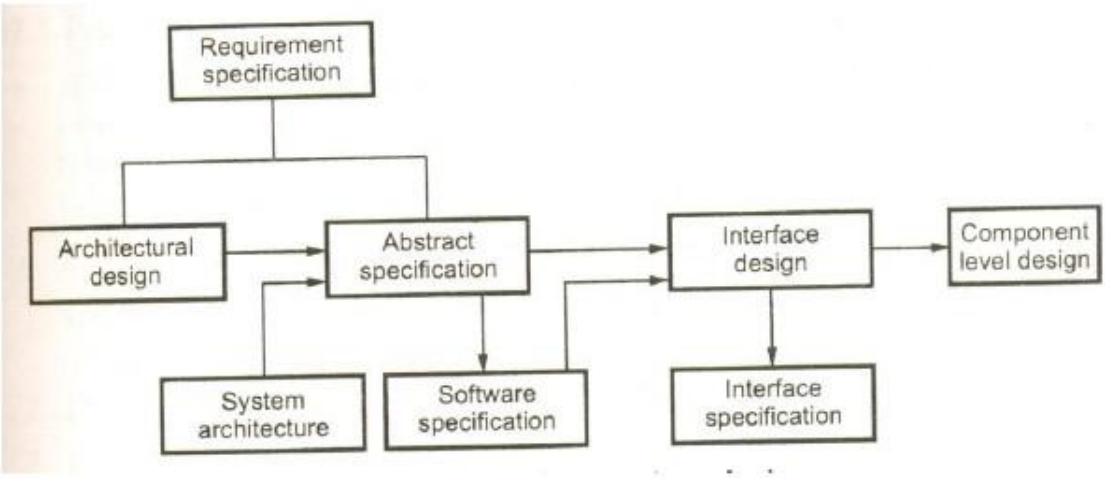
II. Aperiodic stimuli - These are stimuli that occur at unpredictable time interval. For example - Power failure in the system.

- Different stimuli require different time intervals. Hence the real time system architecture should be such that there fast switching between the stimuli handler should be allowed.

- Real time systems are designed with the help of many co-operating processes.

3.16 System Design

- In real time system design both the hardware and software systems are designed .Then functionalities are identified and classified in hardware or software. The system design decisions can be made based on non functional requirements.



- Timing constraints -The timing constraints are applied to various class of stimulus/response systems. It is checked that whether these timing constraints are being satisfied by the system. The timing constraints suggest that certain design strategies can not be applied to avoid additional overhead involved. The low level programming features can be used in system design for improving the performance.

- State machine modeling - On triggering some stimulus the system may transition from one state to another state. Hence to model the real time systems the Finite State Machines are u~ed. But drawback of this design is that it gives complex model even for the simple systems. Some UML notations can be used in FSM for modeling the real time systems.

- Real time programming - For programming hard real time systems the assembly language is used. The assembly languages are used when strict deadlines has to be met. The C language is also efficient for real time programming. But C does not support concurrency and shared resource management.

  Language ADA can be used for real time programming. It supports the concurrency mechanism.

JAVA is the most effective language for real time programming. JAVA supports thread and synchronized methods. Java is used in soft real time programming. JAVA 2.0 is not supporting hard real time programming.

3.17 Real Time Executives

- Real time executives are specialized operating systems which are used in process management. The real time executives also perform process management and resource allocation. The real time executive kernel can be modified or unmodified for particular application.

- The real time executive does not support file management.
- The executive components are

  1. Real time clock - It is used while scheduling the processes.

  2. Interrupt handler - It manages the aperiodic requests.

  3. Scheduler - Schedules various processes.

  4. Resource manager - It allocates processor resources and memory.

  5. Dispatcher - It is responsible for start of process execution.

- Non stop system components are -

1. Configuration manager - The configuration manager reconfigures the system dynamically. In dynamic reconfiguration the hardware can be completely replaced and software can be upgraded without stopping the systems.

2. Fault manager - The fault manager detects the hardware and software fault. On detecting the faults the fault manager is responsible to take necessary actions in order to continue working of the system.

- Real time executive components process various types of stimuli. Sometimes this processing may be based on priorities.

1. Interrupt level priority - The processes which require quick response should be assigned highest priority.

2. Clock level priority - Periodic process priority may be aSSigned.

- Interrupt servicing - When a current process is interrupted for processing a high priority process, the control is first transferred to a predetermined memory location. There an instruction jump to interrupt service routine is given. Hence the control is passed to interrupt service routine for processing. All the other interrupts are disabled and the interrupt is serviced. Finally the control returns to interrupted process. The necessary thing is that the interrupt service routine should be small, simple and fast.

- Periodic Process servicing - The periodic processes are assigned with different periods of execution time, deadlines. The process manager schedules the periodic processes when an interrupt is caused by clock-tick of real time clock. The process manager selects the process which is ready for execution.

3.17.1 Process Management

- The process management is responsible for managing the concurrent processes. The periodic processes execute at specific time interval. The real time executes make use of real time clock to determine which process has to be processed at what time.

- The processing of process can be done as follows-

1. The scheduler chooses the process that is ready for execution. The scheduling strategy considers the process priority while choosing the process.

2. The resource manager then allocates the memory and processor for processing.

3. The despatcher selects the process from ready queue, loads it on the processor and starts executing it.

3.17.2 Scheduling Strategies

- The scheduling can be done as

1. Non preemptive scheduling - Once a process is scheduled for execution it runs to its completion. It can not be interrupted or stopped in between.

2. Preemptive scheduling - The execution of executing process may be stopped in between if a high priority process requires a service.

- Various scheduling algorithms that can be used are shortest job fist, round robin, priority based scheduling.

3.18 Data Acquisition System

- The data acquisition system is responsible to acquire the data for processing.
- In data acquisition systems there are two major activities - collection of data from sensors and processing of collected data.
- The producer process is responsible for collecting the data and consumer process is responsible for processing of process.
- The circular or ring buffer is used to store the collected data.
- There must be a proper synchronization between producer and consumer processes.

- Producer and consumer processes must be mutually excluded from accessing the same element.

- The buffer must stop the producer process to place the data onto it to make it full. Similarly, it should stop the consumer process to access the buffer when it is empty.

3.19 Monitoring and control System

In this class of real time system, monitoring and control systems are designed for the purpose of monitoring and controlling the activities.

Monitoring systems are used to monitor the sensors. The results of these sensors are then reported to the real time system. Using the monitoring system it is possible to check the values of sensors continuously and take the necessary actions depending upon the sensors values.

Control systems take the sensor values and signal the necessary hardware.

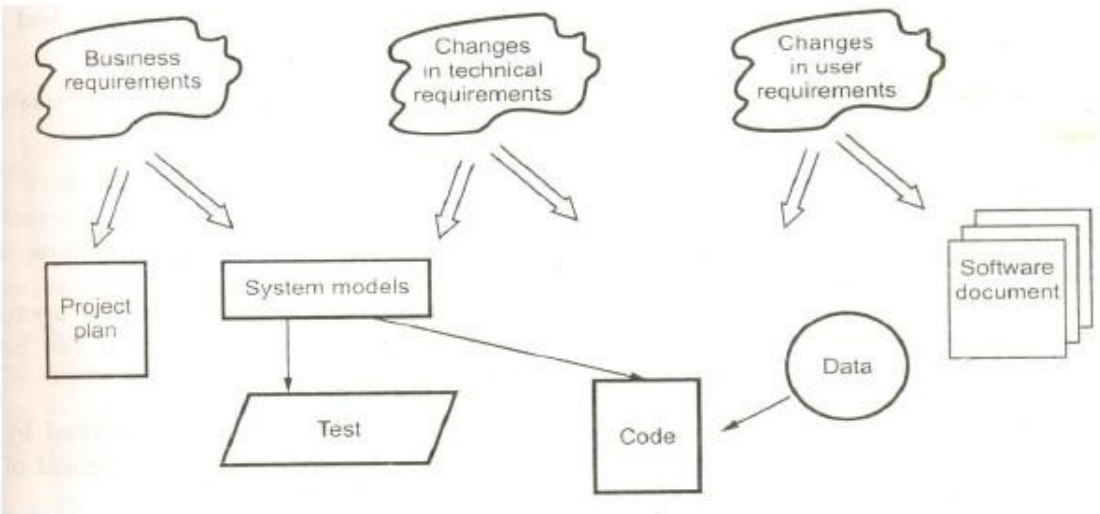
For example - In some real time systems, the temperature can be monitored continuously and if crosses the threshold value the signal is sent to corresponding hardware and the heater can be set off.

3.20 Software Configuration Management (SCM)

Software configuration management is a set of activities carried out for identifying, orgalli:il1g and controlling changes throughout the life cycle of computer software.

During the development of software change must be managed and controlled in order to improve quality and reduce error. Hence Software Configuration Management is a quality assurance activity that is applied throughout the software process.

Why the changes occur?



New n:rsions of the software system are created due to

- Different m,1Chines or different operating systems being used.
- Providing different functionalities.
- Changes in user requirements

The software configuration management is concerned with managing evolving software systems.

SCM Activities

1. Identify change
2. Control change
3. Ensure that change is being properly implemented
4. Report change to others who may have an interest

Software configuration management is a set of tracking and control activities that begin when a software development project begins and terminates when the software is taken out of operation.

3.20.1 Need for SCM

The software configuration management is concerned with managing the changes in the evolving software. If the changes are not controlled at all then this stream 0 uncontrolled change can cause the well-running software project into chaos. Hence it is essential to i) identify these

changes ii)control the changes iii) ensure that the changes are properly implemented and iv) Report these changes to others. And to carry out this set of activities the software configuration is essential.

The software configuration management may be seen as part of quality management process.

3.21 Baseline

- The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as:
- A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.
- A baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of them is obtained through formal technical review
- The baseline is the shared project data base. It is an SCM task to maintain the integrity of the set of artifacts.
- The elements of a design model have been first documented and reviewed. From this design model errors are identified and corrected. Once all parts of the model have been reviewed, corrected and then approved, the design model becomes a baseline.
- Further changes to the program architecture (which is actually documented in the design model) can be made only after each has been evaluated and approved.

3.22 Software Configuration Item

A software configuration Item (SCI) is information that is created as part of the software engineering process.

Examples of Software Configuration Items are

- Computer programs

  Source programs

  Executable programs
- Documents describing the programs

  Technical manual

Users manual

- Data

Program components or functions

External data

File structure

For each type of item, there may be a large number of different individual items produced. For instance there may be many documents for a software specification such as project plan, quality plan, test plan, design documents, programs, test reports, review reports.

These SCI or items will be produced during the project, stored, retrieved, changed, stored again, and so on.

Each configuration item must have a unique name, and a description or specification which distinguishes it from other items of the same type.

3.23 Introduction to SCM Process

The software configuration management is an activity required for software quality assurance.

Various tasks that are carried out in SCM process are -

1. Identification

- Each SCI must be named and identified as objects.
- Software configuration item can be organized to form a database or repository of configuration objects or basic objects
- The configuration object consists of

1. Name and Description

2. Attributes and reference pointer to the object in the database

3. Relationships to other configuration objects

- If a change is made to one Configuration Object it is possible to determine which other Configuration Objects in the repository are affected by the change

2. Version control

- The configuration objects become the baseline, at baseline it becomes version 1.0. Subsequent changes in baseline become new versions.

- The version control combines procedures and tools to manage different versions of configuration objects

3. Change control

Change control is an essential step in software lifecycle. The change control can be carried out using following steps

1. A change request initiates a change

2. The configuration object is 'checked out' of the database.

3. The changes are applied to the object

4. The object is then 'checked in' to the database where automatic version control is applied.

4. Configuration audit

In order to ensure change has been properly implemented or not two activities are carried out

1. Formal Technical Review

2. Software Configuration Audit

In the formal technical review, t~e correctness of configuration object identified and corrected. It is conducted by technical reviewer.

The software configuration audit assesses the configuration object for the characteristics that are not reviewed in formal technical review. It is conducted by the Software Quality Assurance group.

5. Status reporting

The status reporting focuses on communication of the changes to all people in an organization involved with the changes.

In this task of SCM following type of questions are asked

1. What happened?

2. Who did it?

3. When did it happen?

4. What else will be affected?

3.24 Version Control

Version is an instance of a system which is functionally distinct in some way from other system instances.

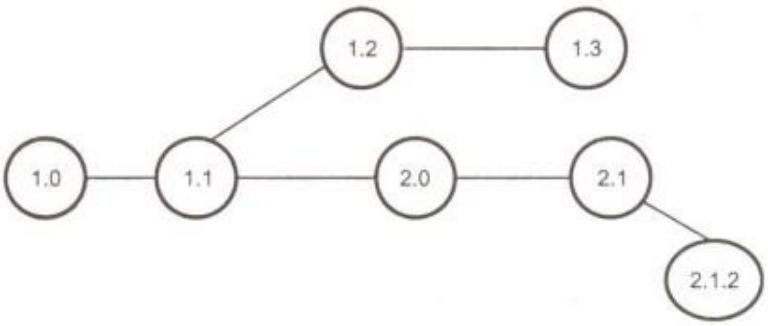
Version control works to help manage different versions of configuration items during the development process.

The configuration management allows a user to specify the alternative configurations of the software system by selecting appropriate version.

Certain attributes are associated with each software version. These attributes are useful in identifying the version. For example: The attribute can be 'date', 'creator', 'customer', 'status'.

In practice the version needs an associated name for easy reference.

Different versions of a system can be shown by an evolution graph as



Each version of software system is a collection of software configuration items.

Review Questions

1. Explain the Design model.

2. What are the design principles?

3. What are the different issues that need to be considered during design of the software?

4. What do you mean by the modular design?

5. Explain the different design Heuristics.

6. What is the significance of design document?

7. Explain Horizontal partitioning and vertical partitioning.

8. Give the guideline for data design.

9. What is call and return architecture?

10. Explain the steps involved in transform mapping.

11. Explain the steps involved in transaction mapping.

12. Give the advantages of GUI

13. Explain the Real time executives.

14. What is SCM?

15. What are Software configuration it('m7

16. Explain the activities that are carried out in software configuration management.

Unit – IV

Testing

4.1 Introduction

People are not perfect. We make errors in design and code. Hence testing is an essential activity in software life cycle. The goal of testing is to uncover as many errors as possible. The software testing is an important activity carried out in order to improve the quality of the software. For finding out all possible errors the testing must be conducted systematically and test cases must be designed using disciplined techniques.

Definition: Software testing is a critical element of software quality assurance. and represents the ultimate review of specification, design, and coding.

The purpose of software testing is to ensure whether. the software functions appear to be working according to specifications and performance requirements.

Testing objective

According to Glen Myers the testing objectives are

1. Testing is a process of executing a program with the intend of finding an error.
2. A good test case is one that has high probability of finding an undiscovered error.
3. A successful test is one that uncovers an as-yet undiscovered error.

The major testing objective is to design tests that systematically uncover types of errors with minimum time and effort.

Testing Principles

Every software engineer must apply following testing principles while performing the software testing.

1. All tests should be traceable to customer requirements.
2. Tests should be planned long before testing begins.
3. The Pareto principle can be applied to software testing - 80'10 of all errOTl uncovered during testing will likely be traceable to 20% of all program modules.
4. Testing should begin "in the small" and progress toward testing "in the large".
5. Exhaustive testing is not possible.

6. To be most effective, testing should be conducted by an independent third party.

4.2 Taxonomy of software Testing

There are two general approaches for the software testing. 1. Black box testing

1. Black box testing

The black box testing is used to demonstrate that the software functions are operational. As the name suggests in black box testing it is tested whether the input is i1ccepted properly and output is correctly produced.

The Major focus of black box testing is on functions, operations, external interfaces, external data and information.

2. White box testing

In white box testing the procedural details are closely examined. In this testing the intern, of software are tested to make sure that they operate according to specifications and designs. Thus major focus of white box testing is on internal structures, logic paths, control flows, data flows, internal data structures, conditions, loops, etc.

4.3 Levels

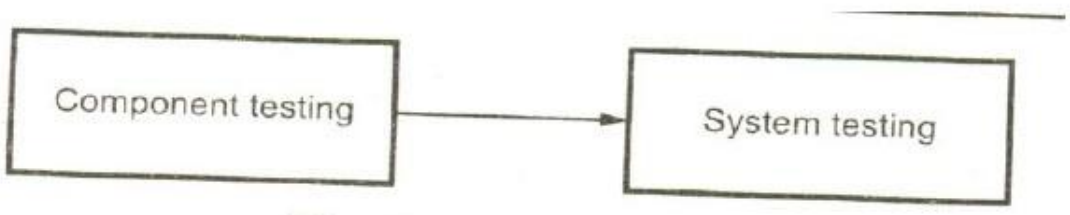
The testing can be typically carried out into two levels.

1. **Component testing**

In Component testing individual components are tested. It is the responsibility of component developer to carry out this kind of testing. These tests are derived from developer's experience.

2. **System testing**

In system testing the testing of groups of components integrated to create a system or sub-system is done. It is the responsibility of an independent testing team to carry out this kind of testing. These tests are based on a system specification.



4.4 Test Activities

Various testing activities are

1. Test planning

The test plan or test script is prepared. These are generated from requirements analysis document (for black box) and program code (for white box).

2. Test case design

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

3. Test execution

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

4. Data collection

The test results are 'collected and verified.

5. Effective evaluation

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

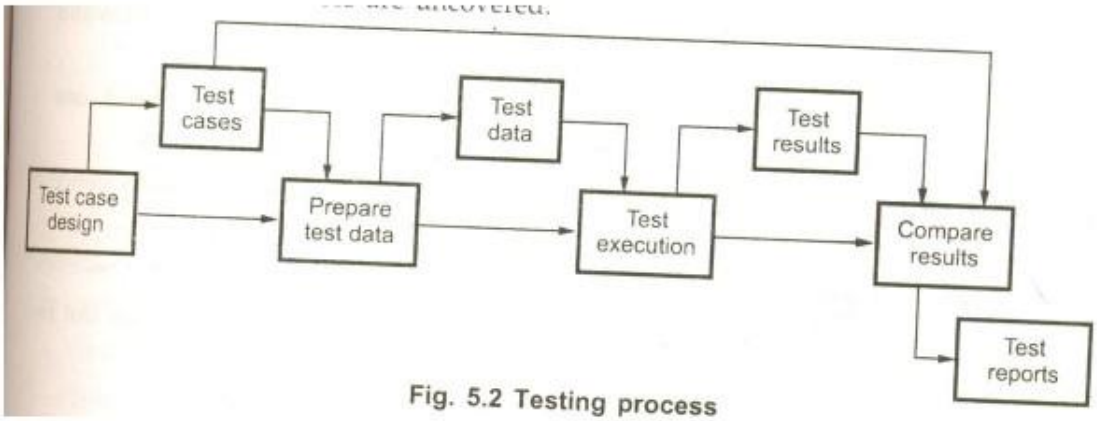


Fig. 5.2 Testing process

4.5 Black Box Test

- The black box testing is also called as behavioral testing.
- Black box testing methods focus on the functional requirements of the software. Tests sets are derived that fully exercise all functional requirements.
- The black box testing is not an alternative to white box testing and it uncover different class of errors than white box testing.
- Black box testing uncovers following types of errors

1. Incorrect or missing functions

2. Interface errors

3. Errors in data structures

4. Performance errors

5. Initialization or termination errors

4.5.1 Equivalence Partitioning

- It is a black-box technique that divides the input domain into classes of data. From this data test cases can be derived.
- An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is ·observed.
- In equivalence partitioning the equivalence classes are evaluated for given input condition. Equivalence class represents a set of valid or invalid states for input conditions.
- Equivalence class guidelines can be as given below:
  - If input condition specifies a range, one valid and two invalid equivalence classes are defined.
  - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
  - If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined.
  - If an input condition is Boolean, one valid and one invalid equivalence class is defined.

For example

Area code: input condition. Boolean - the area code mayor may not be present.

Input condition, range - value defined between 200 and 700.

Password: input condition, Boolean - a password mayor may not be present.

Input condition, value - seven character string.

Command: input condition, set - containing commands noted before.

4.5.2 Boundary Value Analysis

- A boundary value analysis is a testing technique in which the elements at the edge of the domain are selected and tested.
- Instead of focusing on input conditions only, the test cases from output domain are also derived.
- Boundary value analysis is a test case design technique that complements equivalence partitioning technique.
- Guidelines for boundary value analysis technique are

1. If the input condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.

2. If input condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as wit~ the values that are just above and below the maximum and minimum should be tested.

3. If the output condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.

4. If output condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as with the values that are just above and below the maximum and minimum should be tested.

5. If the internal program data structures specify such boundaries then the test cases must be designed such that the values at the boundaries of data structure can be tested.

For example

Integer D with input condition [-2, 10l,

test values: -2, 10, 11, -I, 0

If input condition specifies a number values, test cases should developed to exercise the minimum and maximum .numbers. Values just above and below this min and max should be tested.

Enumerate data E with input condition: {2, 7, 100, 102}

Test values: 2, 102, -I, 200, 7

4.6 White Box Testing

- The white box testing is a testing method which is based on close examination of procedural details. Hence it is also called as glass box testing.

- In white box testing the test cases are derived for

1. Examining all the independent paths within a module.

2. Exercising all the logical paths with their true and false sides.

3. Executing all the loops within their boundaries and within operational bounds.

4. Exercising internal data structures to ensure their validity.

Why to perform white box testing?

There are three main reasons behind performing the white box testing.

1. Programmers may have some incorrect assumptions while designing or implementing some functions. Due to this there are chances of having logical errors in the program. To detect and correct such logical errors procedural details need to be examined.

2. Certain assumptions on flow of control and data may lead programmer to make design errors. To uncover the errors on logical path, white box testing is must.

3. There may be certain typographical errors that remain undetected even after syntax and type checking mechanisms. Such errors can be uncovered during white box testing.

4.6.1 Cyclomatic Complexity

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

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

The cyclomatic complexity can be computed by one of the following ways.

1. The number of regions of the flow graph correspond to the cyclomatic complexity.

2. Cyclomatic complexity, V(G), for a flow graph, G, is defined as:

   V(G) = E - N + 2 ,

   E - number of flow graph edges,

   N - number of flow graph nodes

3. V(G) = P + 1

where P is the number of predicate nodes contained in the flow graph G.

4.7 Structural Testing

- The structural testing is sometime called as white-box testing.
- In structural testing derivation of test cases is according to program structure. Hence knowledge of the program is used to identify additional test cases.
- Objective of structural testing is to exercise all program statements.

4.7.1 Condition Testing

- To test the logical conditions in the program module the condition testing is used. This condition can be a Boolean condition or a relational expression.
- The condition is incorrect in following situations.

i) Boolean operator is incorrect, missing or extra.

ii) Boolean variable is incorrect.

iii) Boolean parenthesis may be missing, incorrect or extra.

iv) Error in relational operator.

v) Error in arithmetic expression.

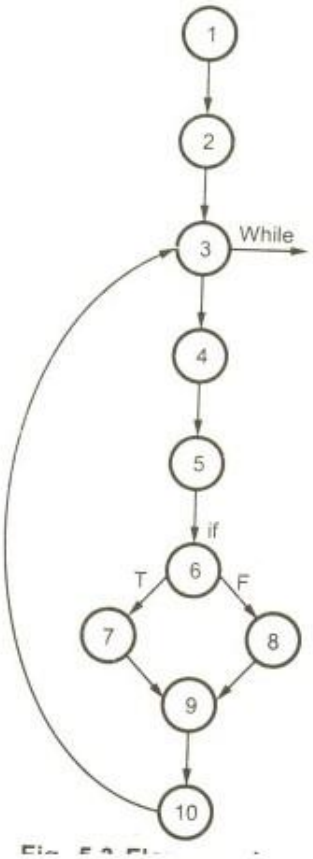
- The condition testing focuses on each testing condition in the program.
- The branch testing is a condition testing strategy in which for a compound condition each and every true or false branches are tested.
- The domain testing is a testing strategy in which relational expression can be tested using three or four tests.

4.8 Test Coverage Criteria Based on Data Flow Mechanisms

- The testing based on data flow mechanism performs testing on definitions and .uses of variables in the program.
- In this method of testing, definition and use chain (DU-chain) is required. The DU chain is obtained by identifying the def and use pairs from the program structure. This strategy of testing is also called as DU testing strategy.
- Set DEF(n) contains variables that are defined at node n.
- Set USE(n) contains variables that are read or used at node n.

For example

```
1.  s: =0;
2.  a: =0;
3.  3.- while (a<b) {
4.  a:=a+2;
5.  b=b-4;
6.  if (a+b<20);
7.  s:=s+a+b;
    else
8.  s:=s+a+b;
}
```



For above given programming lines the DU chain will be -

DEF(1) = {s} USE(1) = {φ}

DEF(2) = {a} USE(2) = {φ}

DEF(3) = {φ} USE(3) = {a, b}

DEF(4) = {a} USE(4) = {a}

DEF(5) = {b} USE(5) = {b}

DEF(6) = {φ} USE(6) = {a, b}

DEF(7) = {s} USE(7) = {s, a, b}

DEF(8) = {s} USE(8) = {s, a, b}

DEF(9) = {φ} USE(9) = {φ}

DEF(10) = {φ} USE(10) = {φ}

- Identify all DU pairs and construct test cases that cover these pairs.
- Identify all DU paths: That means for each DU pair (n1,n2) for variable a, exercise all possible paths n1. .. n2 that are clear of definitions of a.
- Identify all Uses: That means for each DU pair (n1,n2) for a, exercise atleast one path n1.. .n2 that is clear of definitions of a

4.9 The Strategic Approach

- A testing strategy provides a process that describes for the developer, Quality analysts, and the customer the steps conducted as part of testing. The testing strategy includes
  - Test planning
  - Test case design
  - Test execution
  - Data collection
  - Effectiveness evaluation
- The strategic approach for software testing can be

1. The process of testing begins at the component level and works outward toward the integration of the entire computer-based system.
2. Different testing techniques can be applied at different point of time.
3. The developer of the software conducts testing and may be assisted by independent test groups for large projects.
4. Testing and debugging are different activities.
5. Debugging must be accommodated in any testing strategy.

- The software strategy for software testing must perform low-level tests th?t are necessary to verify that small source code segment has been correctly implemented. Similarly the high-level tests should be conducted that validate major system functions against customer requirements.

- Who are involving software testing?
  - Developers

- Testers (test engineers) in Independent Test Group (ITG)
- SQA group

4.9.1 Verification and Validation

- Verification refers to the set of activities that ensure that software correctly implements a specific function.
- Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.
- According to Boehm

  Verification: "Are we building the product right?"

  Validation: "Are we building the right product?"
- Software testing is only one element of Software Quality Assurance (SQA).
- Quality must be built into the development process, you can't use testing to add quality after the fact.
- Verification and validation involve large number of Software Quality Assurance activities such as
  - Formal technical reviews
  - Quality and configuration audits a Performance monitoring
  - Feasibility study
  - Documentation review
  - Database review
  - Algorithmic analysis
  - Development testing
  - Installation testing

4.10 The Software Testing Strategy

- We begin by 'testing-in-the-small' and move toward 'testing-in-the-large'.
- Various testing strategies for conventional software are

1. Unit testing

2. Integration testing

3. Validation testing

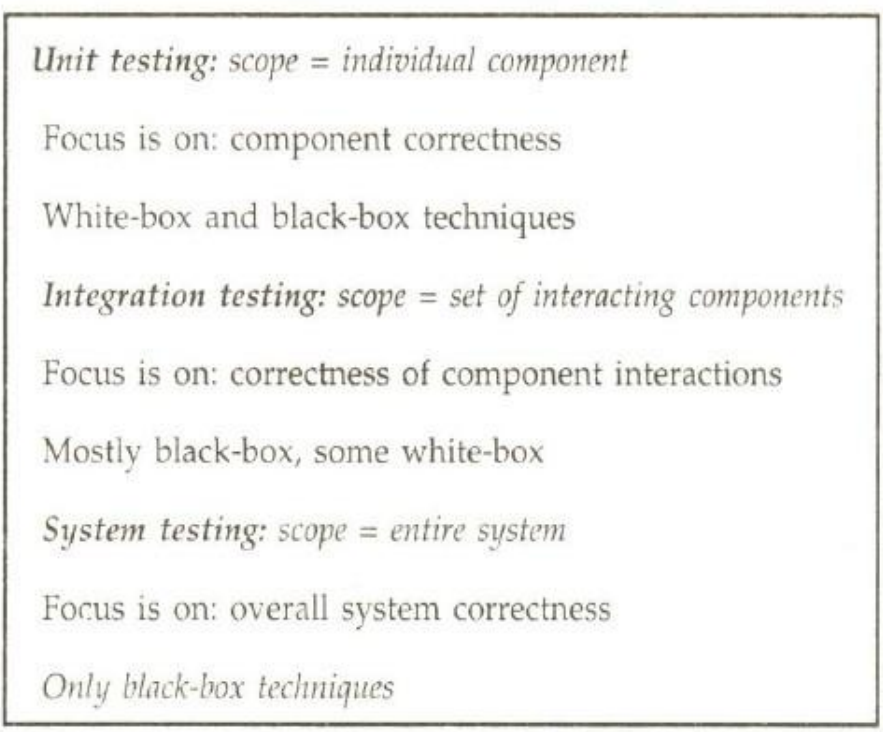
4. System testing

- Unit testing - In this type of testing techniques are applied to detect the errors from each software component individually.

- Integration testing - It focuses on issues associated with verification and program construction as components begin interacting with one another.

- Validation testing -It provides assurance that the software validation criteria (established during requirements analysis) meets all functional, behavioral, and performance requirements.

- System testing – In system testing all system elements forming the system is tested as a whole.



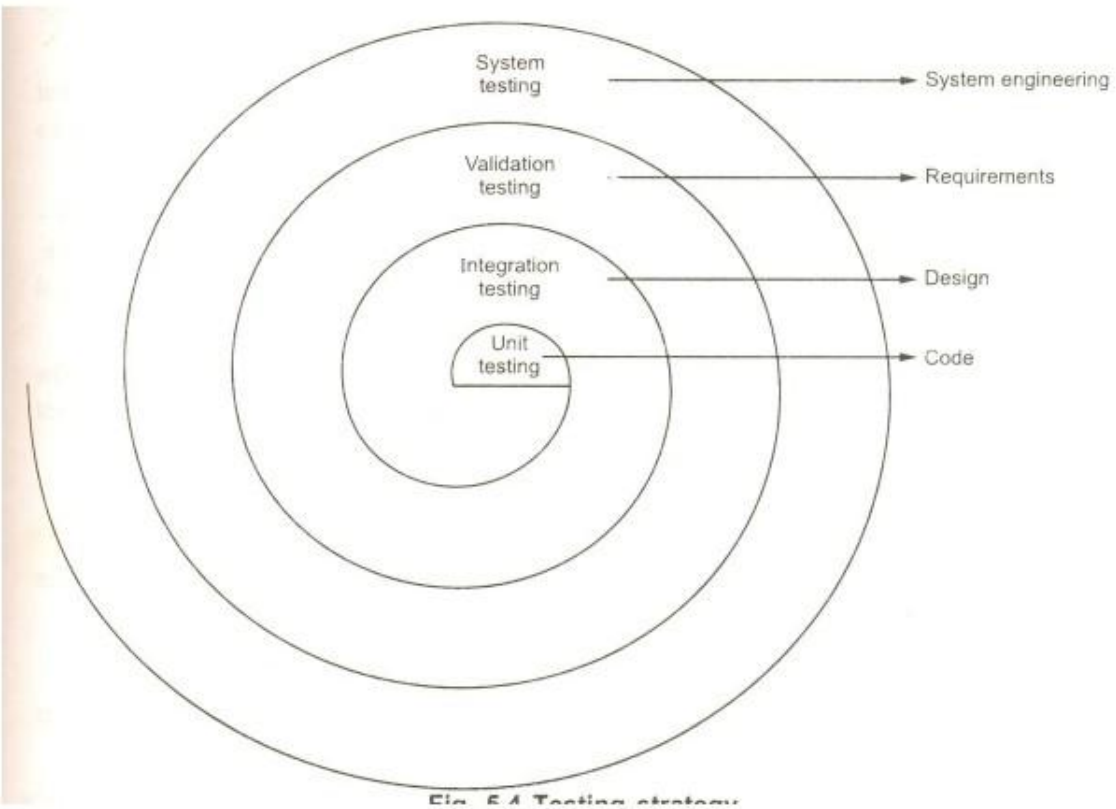


Fig. 5.4 Testing strategy

4.11 Strategic issues

- ***Specify product requirements in a quantifiable manner before testing starts -*** Certain quality characteristics of the software such as maintainability, portability and usability should be specified in order to obtain the unambiguous test results.

- ***Specify testing objectives explicitly*** - Testing objectives such as effectiveness, mean time to failure, and cost of defects should be stated clearly in the test plan.

- ***Identify categories of users for the software and develop a profile for each-*** Use cases describe the interactions among different class of users and thereby testing can focus on the actual use of the product.

- ***Develop a test plan that emphasizes rapid cycle testing*** - Test plan is an important document which helps the tester to perform rapid cycle testing (2 percent of project effort)

- ***Build robust software that is designed to test itself*** - The software should be capable of detecting certain classes of errors. Moreover, the software design should allow automated testing and regression testing.

- ***Use effective formal reviews as a filter prior to testing*** - Formal technical reviews need to be conducted to uncover errors. The effective technical reviews conducted before testing, reduce significant amount of testing efforts.

- ***Conduct formal technical reviews to assess the test strategy and test cases*** - The formal technical review helps to detect any the lacuna in testing approach. Hence it is necessary to assess the test strategy and test cases by technical reviewers to improve the quality of software.

- ***Develop a continuous improvement approach for the testing process*** - The measured test strategy should be used as part of statistical process control approach for software testing.

4.12 Unit Testing

- In unit testing the individual components are tested independently to ensure their quality.
- The focus is to uncover the errors in design and implementation.
- The various tests that are conducted during the unit test are described as below -

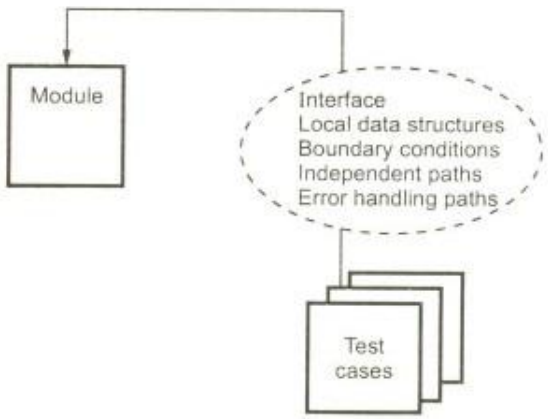
1. Module interfaces are tested for proper information flow in and out of the program.

2. Local data are examined to ensure that integrity is maintained. ~~

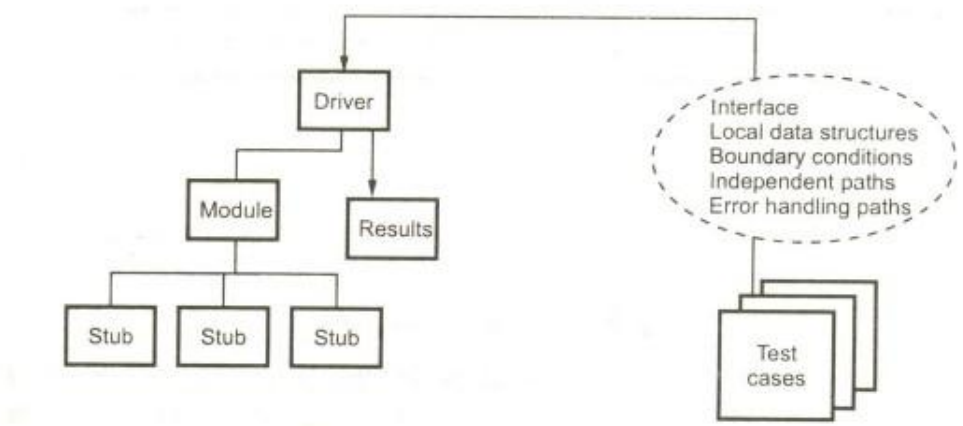
3. Boundary conditions are tested to e~ that the module operates properly at boundaries established to limit or restrict processing.

4. All the basis (independent) paths are tested for ensuring that all statements in the module have been executed only once.

5. All error handling paths should be tested.



6. Drivers and stub software need to be developed to test incomplete software. The "driver" is a program that accepts the test data and prints the relevant results. And the "stub" is a Subprogram that uses the module interfaces and performs the minimal data manipulation if required. This is illustrated by following figure.



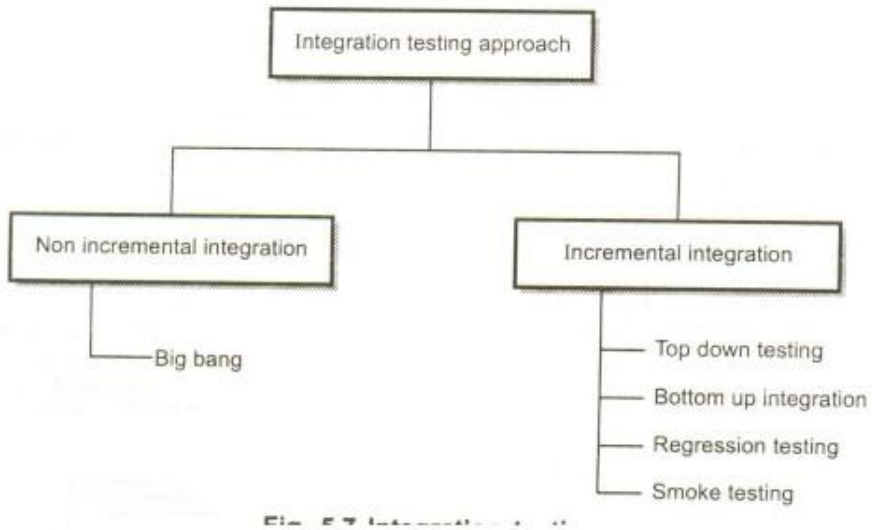
7. The unit testing is simplified when a component with high cohesion (with one function) is designed. In such a design the number of test cases are less and one can easily predict or uncover errors.

4.13 Integration Testing

- A group of dependent components are tested together to ensure their quality of their integration unit.
- The objective is to take unit tested components and build a program structure that has been dictated by software design.
- The focus of integration testing is to uncover errors in:
  - Design and construction of software architecture.
  - Integrated functions or operations at sub-system level.
  - Interfaces and interactions between them.
  - Resource integration and/or between them.
- The integration testing can be carried out using two approaches

1. The non incremental integration

2. Incremental integration



- The non incremental integration is given by the "big bang" approach. All components are combined in advance. The entire program is tested as a whole. And chaos usually results. A set of errors is tested as a whole. Correction is difficult because isolation of causes is complicated by the size of the entire program. Once these errors are corrected new ones appear. This process continues infinitely.

Advantages of big-bang: This approach is simple.

Disadvantages:

1. It is hard to debug.
2. It is not easy to isolate errors while testing.
3. In this approach it is not easy to validate test results.
4. After performing testing, it is impossible to form an integrated system.

- An incremental construction strategy includes

  Top-down integration

  Bottom-up integration

  Regression testing

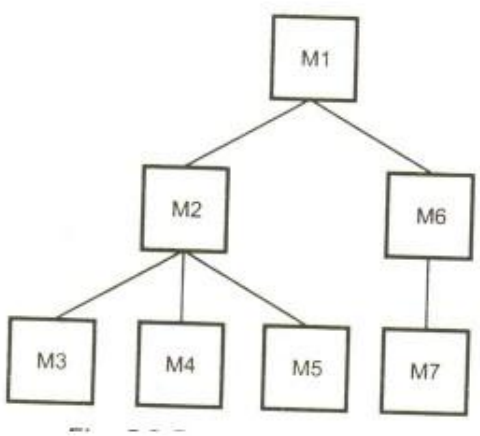
  Smoke testing

4.13.1 Top Down Integration Testing

- Top down testing is an incremental approach in which modules are integrated by moving down through the control structure.
- Modules subordinate to the main control module are incorporated into the system in either a depth-first or breadth-first manner.
- Integration process can be performed using following steps

1. The, main control module is used as a test driver, and the stubs are substituted for all modules directly subordinate to the main control module.
2. Subordinate stubs are replaced one at a time with actual modules using either depth first or breadth first method.
3. Tests are conducted as each module is integrated.
4. On completion of each set of tests, another stub is replaced with the real module.
5. Regression testing is conducted to prevent the introduction of new errors.

For example



In top down integration if the depth first approach is adopted then we will start integration from module Ml then we will integrate M2 then M3, M4, M5, M6 and then M7.

If breadth first approach is adopted then we will integrate module Ml first then M2, M6. Then we will integrate module M3, M4, M5 and finally M7.

4.13.2 Bottom Up Integration Testing

In bottom up integration the modules at the lowest levels are integrated at first, then integration is done by moving upward through the control structure.

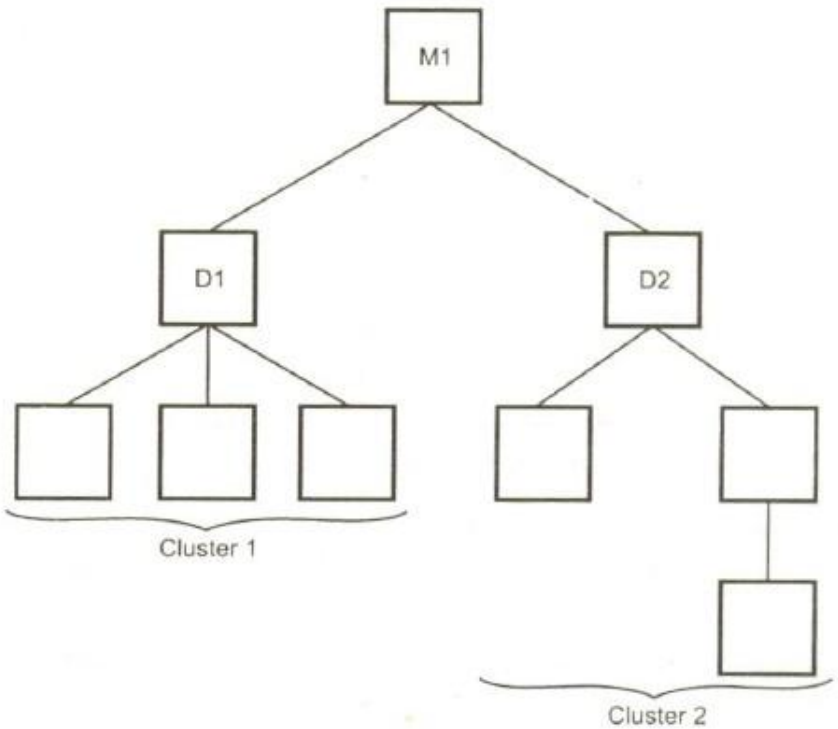
The bottom up integration process can be carried out using following steps.

1. Low-level modules are combined into clusters that perform a specific software sub-function.

2. A driver program is written to coordinate test case input and output.

3. The whole cluster is tested.

4. Drivers are removed and clusters are combined moving upward in the program structure.



First components are collected together to form cluster 1 and cluster 2. Then each cluster is tested using a driver program. The clusters subordinate the driver module. After testing the driver is removed and clusters are directly interfaced to the modules.

4.13.3 Regression Testing

- Regression testing is used to check for defects propagated to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.

- There are three different classes of test cases involved in regression testing -
    - Representative sample of existing test cases is used to exercise all software functions.
    - Additional test cases focusing software functions likely to be affected by the change.
    - Tests cases that focus on the changed sottware components.

- After product had been deployed, regression testing would be necessary because after a change has been made to the product an error that can be discovered and it should be corrected. Similarly for deployed product addition of new feature may be requested and implemented. For that reason regression testing is essential.

4.13.4 Smoke Testing

- The smoke testing is a kind of integration testing technique used for time critical projects wherein the project needs to be assesed on frequent basis.

- Following activities need to be carried out in smoke testing -

1. Software components already translated into code are integrated into a "build". The "build" can be data files, libraries, reusable modules, or program components.

2. A series of tests are designed to expose errors from build so that the "build" performs its functioning correctly.

3. The "build" is integrated with the other builds and the entire product is smoke tested daily.

Smoke Testing Benefits

1. Integration risk is minimized.

2. The quality of the end-product is improved.

3. Error diagnosis and correction are simplified.

4. Assessment of progress is easy.

4.14 Validation Testing

- The integrated software is tested based on requirements to ensure that the desired product is obtained.
- In validation testing the main focus is to uncover errors in

  - System input/output

  - System Junctions and information data

  - System interfaces with external parts

  - User interfaces

  - System behavior and performance
- Software validation can be performed through a series of black box tests.
- After performing the validation tests there exists two conditions

  1. The function or performance characteristics are according to the specifications and are accepted.

  2. The requirement specifications are derived and the deficiency list is created. The deficiencies then can be resolved by establishing the proper communication with the customer.
- Finally in validation testing a review is taken to ensure that all the elements of software configuration are developed as per requirements. This review is called configuration review or audit.

Acceptance Testing

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

The acceptance testing can be conducted over a period of weeks or months.

The types of acceptance testing are

1. Alpha test - The alpha testing is a testing in which the version of complete software is tested by the customer under the supervision of developer. This testing is performed at developer's site.

2. Beta test - The beta testing is a testing in which the version of software is tested by the customer without the developer being present. This testing is performed at customer's site.

4.15 System Testing

The system test is a series of tests conducted to fully the computer based system.

Various types of system tests are

1. Recovery testing
2. Security testing
3. Stress testing
4. Performance testing

The main focus of such testing is to test

- System functions and performance
- System reliability and recoverability (recovery test)
- System installation (installation test)
- System behavior in the special conditions (stress test)
- System user operations (acceptance test/alpha test)
- Hardware and software integration and collaboration
- Integration of external software and the system.

4.15.1 Recovery Testing

- Recovery testing is intended to check the system's ability to recover from failures.
- In this type of testing the software is forced to fail and then it is verified whether the system recovers properly or not.
- For automated recovery then reinitialization, checkpoint mechanisms, data recovery and restart are verified.

4.15.2 Security Testing

- Security testing verifies that system protection mechanism prevent improper penetration or data alteration.
- It also verifies that protection mechanisms built into the system prevent intrusion such as unauthorized internal or external access or willful damage.
- System design goal is to make the penetration attempt more costly than the value of the information that will be obtained.

4.15.3 Stress Testing

- Determines breakpoint of a system to establish maximum service level
- In stress testing the system is executed in a manner that demands resources in abnormal quantity, frequency or volume.
- A variation of stress testing is a technique called sensitivity testing.
- The sensitive testing is a testing in which it is tried to uncover data from a large class of valid data that may cause instability or improper processing.

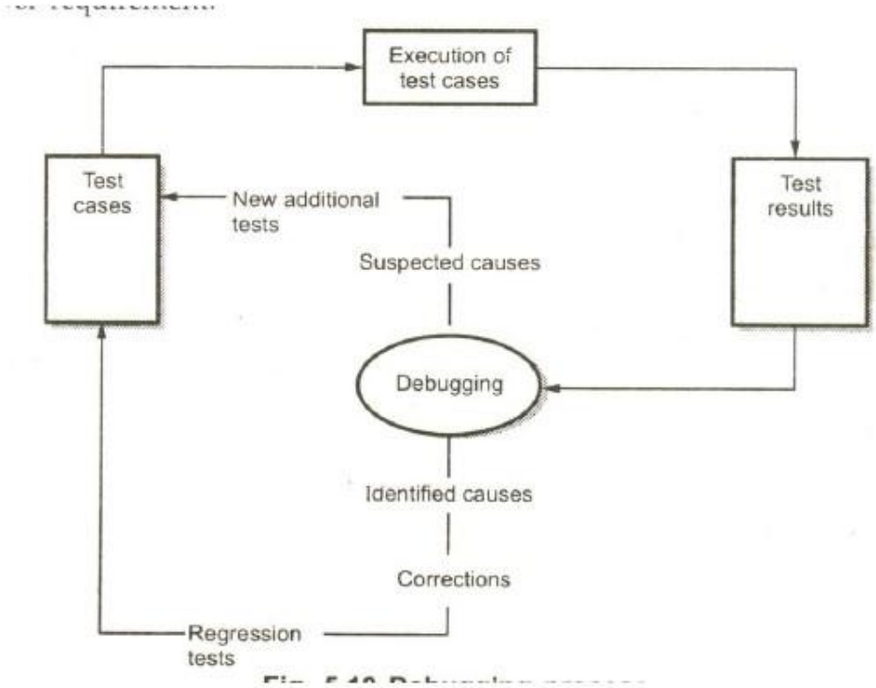
4.15.4 Performance Testing

- Performance testing evaluates the run time performance of the software, especially real-time software.
- In performance testing resource utilization such as CPU load, throughput, response time, memory usage can be measured.
- For big systems (e.g. banking systems) involving many users connecting to servers (e.g. using internet) performance testing is very difficult.
- Beta testing is useful for performance testing.

4.16 Debugging

Debugging is a process of removal of a defect. It occurs as a consequence of successful testing.

Debugging process starts with execution of test cases. The actual test results are compared with the expected results. The debugging process attempts to find the lack of correspondence between actual and expected results. The suspected causes are identified and additional tests or regression tests are performed to make the system to work as per requirement.



Common approaches in debugging are :

Brute' force method -The memory dumps and run-time traces are examined and program with write statements is loaded to obtain clues to error causes.

In this method "Let computer find the error" approach is used. \

This is the least efficient method of debugging.

Backtracking Method -This method is applicable to small programs.

In this method, the source code is examined by looking backwards from 'symptom to potential causes of errors.

Cause elimination Method - This method uses binary partitioning to reduce the number of locations where errors can exist.

Thus testing is an essential activity carried out during software development process for improving quality of the product.

Review Questions

1. What is testing? Give the taxonomy of testing.
2. Enlist the testing objectives.
3. What is equivalence partitioning?
4. Compare white box testing and black box testing.
5. What is cyclomatic complexity?

6. What is verification and validation?

7. Explain the software testing strategy.

8. What is top down integration testing?

9. What is bottom lip integration testing?

10. Explain the regression testing.

11. Explain the debugging process.

12. What are the common approaches in debugging?

Unit – V

Software Project Management

5.1 Introduction

Software project management is an activity of organizing, pla~ning and scheduling software projects. The goal of software project management is to deliver the software on time and on schedule. It is also concerned with the fact that software should get developed in accordance with the requirements of organization. The software project management is needed because it performs all the activities related to the budget and schedule constraints that are set by the organization. Activities in software project management:

1. Project planning
2. Project scheduling
3. Risk management
4. Managing people.

Project planning

Under this activity various different types of plan may be developed to support the main software project plan that is concerned with schedule and budget.

This is the continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.

Types of project plan

- Quality plan-This plan describes the quality procedures and standards that - will be used in a project.
- Validation plan-This plan describes the approach, resources and schedule required for system validation.
- Configuration management plan-This plan focuses on the configuration management procedures and structures to be used.
- Maintenance plan-The purpose of maintenance plan is to predict the maintenance requirements of the system, maintenance cost and effort required.
- Staff development plan-This plan describes how to develop the skills and experience of the project team members.

Activities in project planning

The project planning activities can be carried out in following steps.

1. Identify the constraints in the project

2. Make initial assessment of the project

3. Define project milestones and deadlines.

4. While developing the project

I. Prepare project schedule.

II. Initiate project activities as per the project schedule. iii) Review progress of the project.

III. Update project schedule if required.

IV. Revise project constraints and deliverables.

V. If any problem arises then perform technical review.

5. Repeat the step 4 until the project is ready

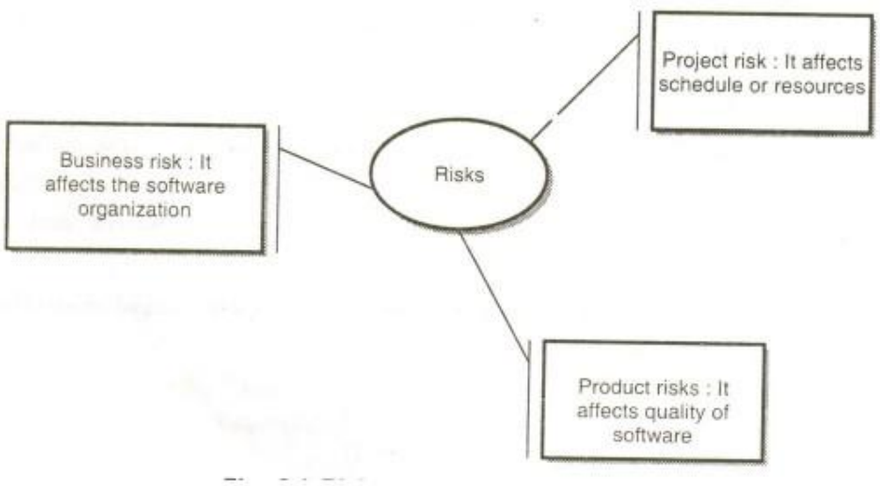
Project scheduling

While scheduling the project the project is split into number of tasks and time and resources required for each task to accomplish is estimated.

Risk management

Risk management is an activity in which risks in the software project are identified. Under risk management plans are prepared to minimize their effects on a project.

A risk is a probability that some adverse circumstance will occur.



The risk management process

The risk management process can be carried out in following stages.

1. Risk identification-In this phase the project, product and business risks are identified.

2. Risk analysis-Under risk analysis, the analysis is made on probabilities and consequences of project, product and business risks.
3. Risk planning-In risk planning the plans are prepared to avoid or minimize the effects of the risk.
4. Risk monitoring-Monitoring of the risks is an actj.vrty which needs to be carried out throughout the project.

Managing people

People is an important asset of any software project. Staff selection factors include education, domain experience, adaptability and personality. People in the software project are motivated by interaction, recognition and personal development.

In Software development groups, the leaders should be competent, dynamic and should have administrative and technical support.

Good project management cannot guarantee success, but poor management on significant projects always leads to failure.

5.2 Measures and Measurement

Software measurement means deriving a numeric value for an attribute of a software product or process.

Measure:

It is a quantitative indication of the extent, amount, dimension, or size of some attribute of a product or process.

Metrics:

It is the degree to which a system, component, or process possesses a given attribute. The software metrics relate several measures. For example - average number of errors found per review.

Indicators:

Indicators mean combination of metrics that provides insight into the software process, project or product.

Direct Metrics:

It refers to immediately measurable attributes. For example - line 'of code, execution speed

Indirect Metrics:

It refers to the aspects that are not immediately quantifiable or measurable. For example-functionality of the program.

Faults:

Faults are of two types.

Errors:

These are type of faults that are found by the practitioners during software development.

Defects

Defects mean faults found by the customers after release.

5.2.1 Software Complexity and Science Measure

The software complexity and science measure is a type of measurement which related to software system process or related documentation.

The software metrics allows the software and software process to be quantified.

This measure is used to predict product attributes or to control the software process.

Product metrics is used to identify the anomalous components.

Normalization for Metrics

The process of combining of metrics coming from different projects depends on the size and the complexity of the software.

Hence the normalization for metrics can be performed. The normalization compensate for complexity aspects particular to a product.

There are two approaches for normalization.

- Size oriented (lines of code approach)
- Function oriented (function point approach).

5.2.2 Size Measure

- Size oriented measure is derived by considering the size of software that has been produced.
- The organization builds a simple record of size measure for the software projects. It is built on past experiences of organizations.
- It is a direct measure of software·

Project	LOC	Effort	Cost($)	Doc.(pgs.)	Errors	Defects	People
ABC	10,000	20	170	400	100	12	4
PQR	20,000	60	300	1000	129	32	6
XYZ	35,000	65	522	1290	280	87	7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

- A simple set of size measure that can be developed is as given below.
  - Size = Kilo Lines of Code (KLOC)
  - Effort = person / month
  - Productivity = KLOC/person-month
  - Quality = number of faults/KLOC
  - Cost = $/KLOC
  - Documentation = pages of documentation/KLOC
- The size measure is based on the lines of code computation. The lines of code is defined as one line of text in a source file.
- While counting the Jines of code the Simplest Standard is
  - Don't count blank lines
  - Don"t count comments
  - Count everything else
- The size oriented measure is not universally accepted method.

Advantages

(1) Artifact of software development which is easily counted.

(2) Many existing methods use LOC as a key input.

(3) A large body of literature and data based on LOC already exists.

Disadvantages

(1) This measure is dependent upon the programming language.

(2) This method is well designed but shorter program may get suffered.

(3) It does not accommodate non procedural languages.

(4) In early stage of development it is difficult to estimate LOC.

5.3 Software Cost Estimation

- The software cost estimation is the process of predicting the resources required for software development process.
- Fundamental questions that are asked to judge the estimation are

1. How much effort is required to complete the project or an activity?
2. How much calendar time is needed to complete an activity?
3. What is the total cost computed for an activity?

- The software cost component are

1. Hardware and software costs
2. Travel and software or technology training costs
3. Effort costs (the dominant factor in most projects) which involves
   - Salaries of employees involved in the project

- Social and insurance costs

4. Costs of building, heating, and lighting.
5. Costs of networking and communications.
6. Costs of shared facilities such as library, staff.

Estimation techniques

- It is not possible to make the accurate estimate of efforts required to develop a software system because

1. The initial estimates are based on inadequate information in a user requirements definition.
2. The software may run on unfamiliar computers or uses new technology.
3. The people in the project may be unknown.

- Project cost estimates may be self-fulfilling. The estimate defines the budget and the product is adjusted to meet the budget.
- Various estimation techniques are

1. Algorithmic cost modeling - The cost estimation is based on the size of the software.
2. Expert judgement - The experts from software development and the application domain use their experience to predict software costs.
3. Estimation by analogy - The cost of a project is computed by comparing the project to a similar project in the same application domain and then cost can be computed.
4. Parkinson's Law - The cost is determined by available resources rather than by objective assessment.
5. Pricing to win - The project costs whatever the customer ready to spend on it. •
    - There are two approaches used in cost estimation.

1. Top-down - In this approach we start estimation at the system level and assess the overall system functionality and focuses on how this is delivered through sub-systems.
2. Bottom-up - In this approach we start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.

5.4 Function Point Model

- The function point model is based on functionality of the delivered application.
- These are generally independent of the programming language used.
- This method is developed by Albrecht in 1979 for IBM.
- Function points are derived using

(1) countable measures of the software requirements domain

(2) assessments of the software complexity.

How to calculate function point?

- The data for following information domain characteristics are collected

1. Number of user inputs - Each user input which provides distinct application data to the software is counted.
2. Number of user outputs - Each user output that provides application data to the user is counted, e.g. screens, reports, error messages.
3. Number of user inquiries - An on-line input that results in the generation of some immediate software response in the form of an output.
4. Number of files - Each logical master file, i.e. a logical grouping of data that may be part of a database or a separate file.
5. Number of external interfaces - All machine-readable interfaces that are used to transmit information to another system are counted.

- The organization needs to develop criteria which determine whether a particular entry is simple, average or complex.
- The weighting factors should be determined by observations or by experiments.

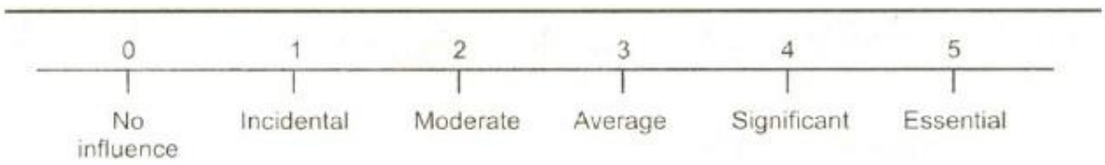
Domain Characteristics	Count		Weighting factor			Count
			Simple	Average	Complex	
Number of user input		X	3	4	6	
Number of user output		X	4	5	7	
Number of user inquiries		X	3	4	6	
Number of files		X	7	10	15	
Number of external interfaces		X	5	7	10	
Count Total						

- The count table can be computed with the help of above given table.
- Now the software complexity can be computed by answering following questions. These are complexity adjustment values.

1. Does the system need reliabie backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance of the system critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files or inquiries complex?
10. Is the internal processing complex?
11. Is the code which is designed being reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?

14. Is the application designed to facilitate change and ease of use by the user?

- Rate each of the above factors according to the following scale:



- Function Points (FP) = Count total x (0.65 + (0.01 x Sum(Fj»
- Once the functional point is calculated then we can compute various measures as follows.
  - Productivity = FP /person-month
  - Quality = number of faults/FP
  - Cost = $/FP
  - Documentation = pages of documentation/FP.

Advantages

1. This method is independent of programming languages.
2. It is based on the data which can be obtained in early stdge of project .

Disadvantages

1. This method is more suitable for business systems and can be developed for that domain.
2. Many aspects of this method are not validated.
3. The functional point has no significant meaning. It is just a numerical value.

5.5 COCOMO Model

- Boehm designed a cost model called COCOMO (Constructive Cost Model) to give an estimate of number of man-months it will take to develop the software product.
- The original COCOMO model was first published in 1981. Boehm and his colleagues have updated the original COCOMO, called COCOMO II, that accounts for recent changes in software engineering technology.
- COCOMO I1 is useful for a much wider collection of techniques and technologies. COCOMO II is applicable for business software, object-oriented software, software created by using spiral or evolutionary development models.
- COCOMO II includes

1. Application Composition Model
2. Early Design Model
3. Post-Architecture Models.

- The COCOMO IT model requires the sizing information such as object point function point, and lines of source code.

Application Composition Model

This model is useful for resolving potential high risk issues such as system pCrfOrl11ilnCe, interfaces or technology maturity.

In this model, the object points are used for sizing rather than the traditional lines of code measure.

An initial size measure is determined by counting the number of screens, reports, and third-generation components that will be used in the application. Each object ~ classified as simple, medium, or difficult as shown in following table.

Complexity weights

Object type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3 GL component	–	–	10

The object point count is calculated by multiplying the original number of object instances by the weighting factor as given in above table. Then these values are :,llllll11cd lip to obtain the total object point count.

Reuse is then taken into account. Assuming that r % of the objects will be reused from previous projects, the number of new object points (NOP) is calculated to be :

NOP = (Object points) x (100 - r) / 100

A productivity rate (PROD) is determined for estimating the efforts. This computation is based on the value of NOP. Hence the productivity rate is calculated as

PROD = NOP/person-month

The productivity rate is as shown below.

Developer's experience and Environment maturity and capability PROD4	Very low Very low	Low capability	Low	Nominal Nominal	High	Very high Very high
		7	13		25	50

The effort can be estimated using following formula

E = NOP / PROD

5.6 Delphi Method

The Delphi technique is an estimation technique intended to achieve a common agreement for estimating efforts. This method is developed by BaryBohem and John Farquhar in 1970. This method involves more interactions and communications between those who are participating. The procedure is as follows.

1. The co-ordinator presents a specification and estimation form to each e~
2. Co-ordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
3. Experts fill out forms anonymously.
4. Co-ordinator prepares and distributes a summary of the estimates.
5. The co-ordinator then calls a group meeting. In this meeting the experts mainly discuss the points where their estimates vary widely.
6. The experts again fill out terms anonymously.
7. Again co-ordinator edits and summarizes the forms, repeating the steps 5 and 6 until the co-ordinator is satisfied with overall prediction synthesized from experts.

The key to this technique is in expert co-ordinator. The co-ordinator must be talented enough to synthesize the diverse and wide ranging statements.

This method is successful for technical forecasting.

The first meeting is the kickoff meeting, during which the estimation team creates it work breakdown structure (WBS) and discusses assumptions. After the meeting, each team member creates an effort estimate for each task.

The second meeting is the estimate session, in which the team revises the estimates as a group and achieves consensus.

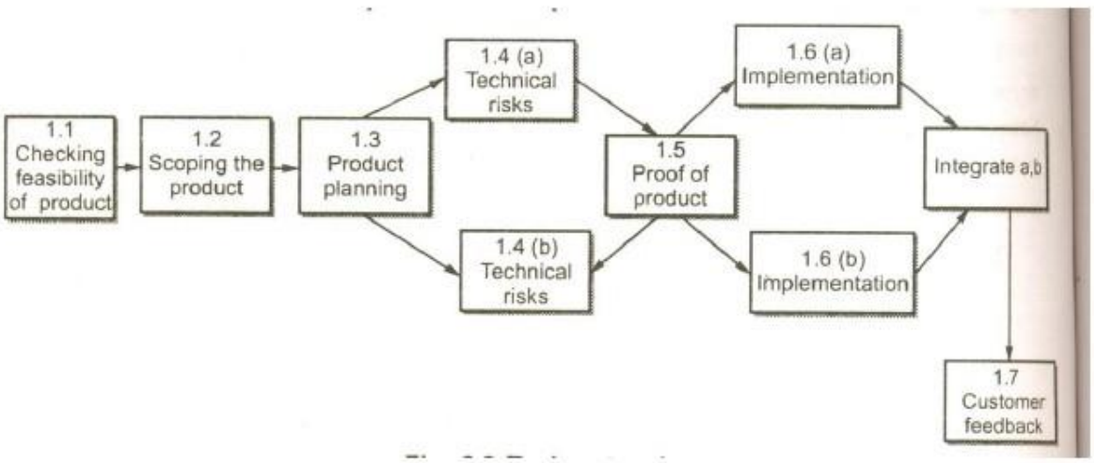
After the estimation session, the project manager summarizes the results and reviews them with the team, at which point they are ready to be u~ed as the basis for planning the software project.

The advantage of this method is that direct inter personnel relations are avoided. In this method the strong willed member cannot dominate the group.

5.7 Defining Task Network

- The task is a unit of work.
- The task network or an activity network is a graphical representation, with:
  - nodes corresponding to activities
  - tasks or activities are linked if there is a dependency between them.

The task network for the product development is as shown below.



- The task network definition helps project manager to understand the project work breakdown structure.
- The project manager should be aware of interdependencies among various tasks. It should be aware of all those tasks which lie on the critical path.

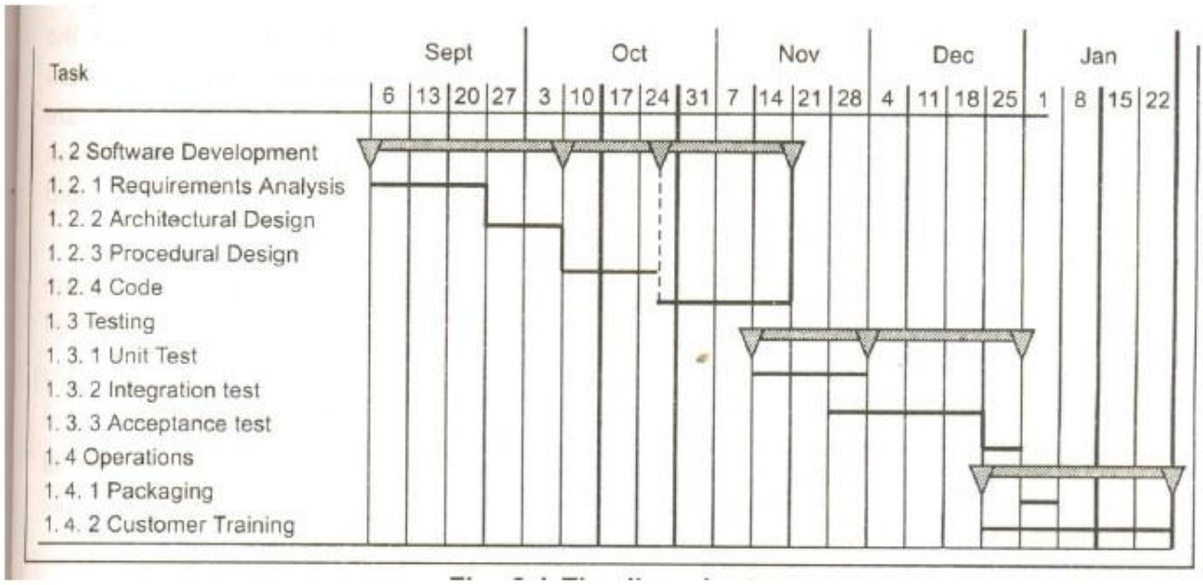
5.8 Scheduling

- While scheduling the project the project is split into number of tasks and time and resources required for each task to accomplish is estimated.
- While scheduling the project, organize tasks concurrently to make optimal use of workforce.
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete. The tasks are also called as work breakdown structure (WBS).
- Program evaluation and review technique (PERT) and critical path method (CPM) are two project scheduling methods used in software development.
- With the help of PERT and CPM technique the software planner can determine
  i) Critical Path - The critical path can be defined as a chain of tasks that determines the duration of the project.
  ii) Time estimates for individual tasks by applying statistical model.
  iii) A boundary time - The boundary time defines the time required by particular task to accomplish.
- Boundary time calculation helps in determining critical path. It also provides the project manager to evaluate the progress of the individual tasks.

5.8.1 Time Line Chart

- In software project scheduling the timeline chart is created. The purpose of time line chart is to emphasize the scope of individual task. Hence set of tasks are given as input to the timeline chart.
- The timeline chart is also called as Gantt chart.

- The time line chart can be developed for entire project or it can be developed for individual functions.
- In time line chart
    1. All the tasks are listed at the leftmost column.
    2. The horizontal bars indicate the time required by the corresponding task.
    3. When multiple horizontal bars occur at the same time on the calendar, then that means concurrency can be applied for performing the tasks.
    4. The diamonds indicate the milestones.
- In most of the projects, after generation of timeline chart the project tables are prepared. In project tables all the tasks are listed along with actual start and end dates and related information.



Project Table

Tasks	Planned Start	Actual start	Planned end	Actual End	Effort Assignment
Requirement analysis	6th Sept'05	6th Sept'05	20th Sept'05	22nd Sept'05	Jayashree, Padma, Lucky
Architectural design	27th Sept'05	27th Sept'05	3rd Oct'05	7th Oct'05	Trupti, Varsha
Procedural design	10th Oct'05	12th Oct'05	24th Oct'05	25th Oct'05	Varsha, Sachin, Devendra
⋮	⋮	⋮	⋮	⋮	⋮
Customer training	1st Jan'06	4th Jan'06	22nd Jan'06	25th Jan'06	Smita, Yogita

5.9 Earned Value Analysis

- Earned Value Analysis (EVA) is technique of performing quantitative analysis of the software project.
- Earned value system provides a common value scale for every task of software project.

- The EVA acts as a measure for software project progress.
- With the help of quantitative analysis made in EVA, we can know how much percentage of the project is completed.
- The earned value analysis can be made using following steps.
    1. The budgeted cost of work scheduled (BCWS) is an estimated cost for the work that has been scheduled. This value is obtained for every individual task in the software project. In this activity the work of each software engineering task is planned. The $BCWS_i$ is the effort planned for work task i. Hence at every point in the progress of project the $BCWS_i$ are calculated c1l1d the total cost is the summation of all the $BCWS_i$•
    2. At the completion of the project the BCWS values for all work tasks are summed to derive the budget of the project. The calculation of budgeted actual cost is $BAC = \Sigma BCWS_i$ for all the tasks i.
    3. Then budgeted cost of work performed (BCWP) is computed. The value of BCWP is the sum of all the BCWS values of all the corresponding tasks that have actually been completed by a point in time on the project schedule.

- The difference between BCWS and BCWP is that BCWS represents values for the project activities that are planned and BCWP represents the values of the project activities that are completed.
- Various types of computations in EVA are given as follows 1. SPI = BCWP/BCWS

1. SPI = BCWP/BCWS

Where SPI is the software performance index. It represents the project efficiency. If the value of SPI is 1.0 then it represents that execution of project is very efficient.

2. SV= BCWP-BCWS

Where SV indicates the scheduled variance.

3. Project scheduled for completion = BCWS/BAC

where project scheduled for completion indicates the percentage of work which should be completed by time t :

Percent complete = BCWP /BAC

Percent complete represents the percent of project which is actually completed by time t.

4. ACWP = Σ efforts expended on work task that have been completed by time t.

where ACWP refers to Actual Cost Work Performance. This value helps in computing the cost factor of the project.

CPI = BCWP/ACWP

Where CPI indicates the cost performance index. This value represents whether the performance of project is within the defined budget or not. The value 1.0 indicates that the project is within the defined budget.

Thus EVA helps in identifying the project performance, cost of performance and project scheduling difficulties. This ultimately helps the project manager to take the appropriate corrective actions.

5.10 Error Tracking

- While developing the software project many work products such as SRS, design document, source code are being created. Along with these work products many errors may get generated. Project manager has to identify all these errors to bring quality software.
- Error tracking is a process of assessing the status of the software project.
- The software team performs the formal technical reviews to test the software developed. In this review various errors are identified and corrected. Any errors that remain uncovered and are found in later tasks are called defects.
- The defect removal efficiency can be defined as

DRE = E/(E+D)

Where ORE is the defect removal efficiency,

E is the error

and D is defect.

- The ORE represents the effectiveness of quality assurance activities. The ORE also helps the project manager to assess the progress of software project as it gets developed through its scheduled work task.
- During error tracking activity following metrics are computed

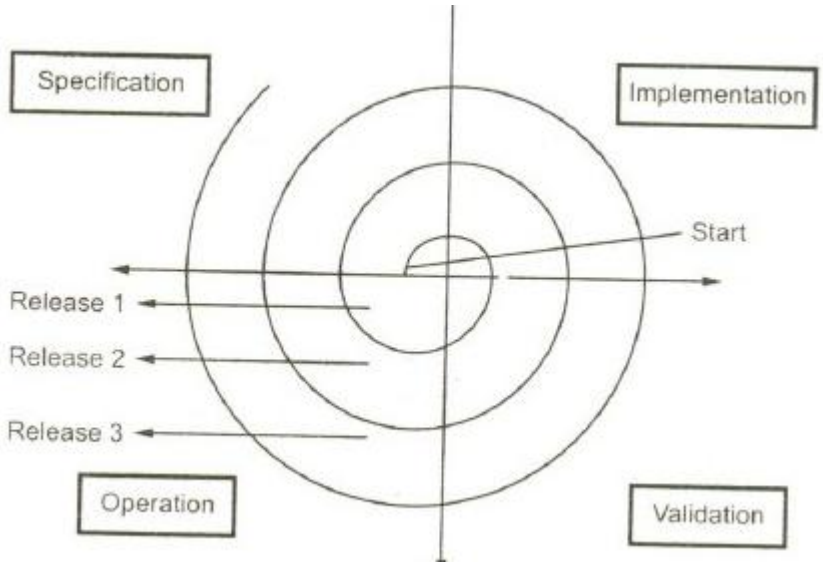
1. Errors per requirements specification page: denoted by E_{req}
2. Errors per component - design level: denoted by E_{design}
3. Errors per component - code level: denoted by E_{code}
4. ORE - requirement analysis
5. ORE - architectural design
6. ORE - component level design
7. ORE - coding

The project manager calculates current values for E_{req}, E_{design} and E_{code}These values are then compared with past projects. If the current result differs more than 20% from the average, then there may be cause for concern and investigation needs to be made in this regard.

- These error tracking metrics can also be used for better target review and testing resources.

5.11 Software Changes

- Software change is an unavoidable process.
- Software change occurs because of following reasons.
    1. New requirements emerge when the software is used.
    2. The business environment changes.
    3. Errors needs to be repaired.
    4. New equipment must be accommodated.
    5. The performance or reliability may have to be improved.
- A key problem for organisations is implementing and managing change to their legacy systems.
- Organizations have huge investments in their software systems - they are critical business assets. To maintain the value of these business assets these software systems must be changed and updated at appropriate time.



The software change strategies that could be applied separately or together are as given below –

1. Software maintenance - The changes are made in the software due to requirements.
2. Architectural transformation - It is the process of changing one architecture in to another form.
3. Software re-engineering - New features can be added to existing system and then the system is reconstructed for better use of it in future.

5.12 Program Evaluation Dynamics

Program evolution dynamics is the study of the processes of system change.

Lehman and Belady proposed a number of 'laws' which can be applied to all systems as they evolved.

These laws are as given below.

Law	Description
Continuing change	A program which is used in a real-world environment needs to be changed as it become progressively less useful in that environment.
Increasing complexity	After making changes in an evolving program, its structure tends to become more complex. This complexity may be used of additional resources in the system for preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases and the number of reported errors is approximately invariant for each system release.
Organisational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	The incremental change in each release during the software development process is approximately constant.
Continuing growth	The user requires more and more features and to satisfy these needs the additional functionalities are offered by the system.
Declining quality	The changes should be adapted by the system in its existing operational environment otherwise the quality of systems will appear to be declining.
Feedback system	The feedback system adopted in the system is useful for product improvement.

The Lehman's l:1w is also applicable to large, tailored system .

5.13 Software Maintenance

Software maintenance is an activity in which program is modified after it has been put into use.

In software maintenance usually it is not preferred to apply major software changes to system's architecture.

Maintenance is a process in which changes are implemented by either modifying the existing system's architecture or by adding new components to the system.

Need for software maintenance

The software maintenance is essential because of following reasons

1. Usually the system requirements are changing and to meet these requirements some changes are incorporated in the system.

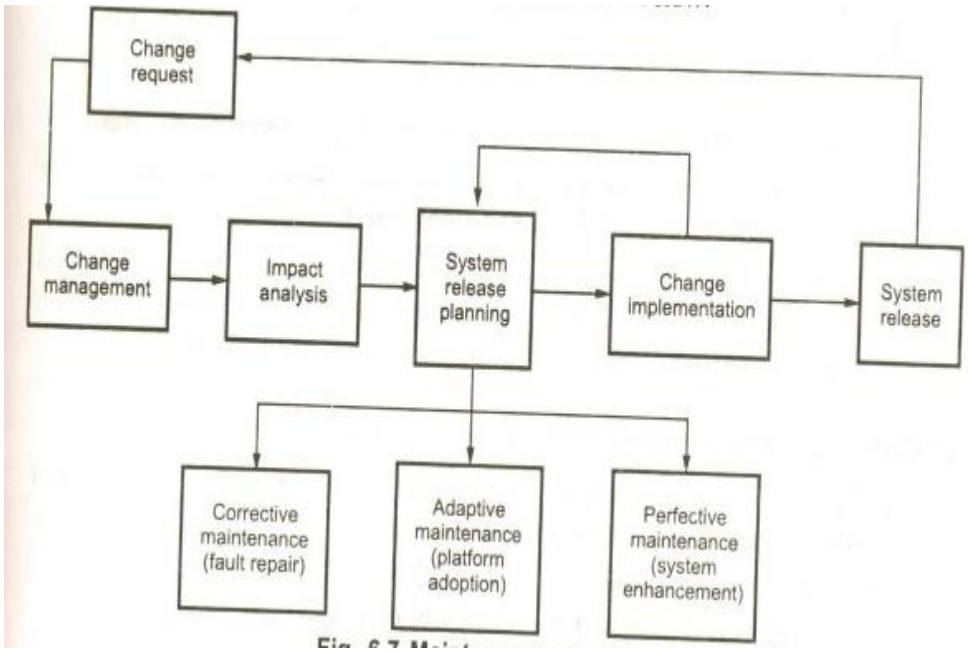
2. There is a strong relationship getween as system and its environment when a system is installed in an environment it changes that environment. This ultimately changes the system requirements.
3. The maintained system remains useful in their working environment.

5.13.1 Types of Software Maintenance

Various types of software maintenance are

1. Corrective maintenance - Means the maintenance for correcting the software faults.
2. Adaptive maintenance - Means maintenance for adapting the change in environment (different computers or different operating systems).
3. Perfective maintenance - Means modifying or enhancing the system to meet the new requirements.
4. Preventive maintenance Means changes made to improve future maintainability.

The software maintenance process can be as shown below.



1. In the maintenance process initially the request for change is made.
2. Change Management - In this phase the status of all the change requests is identified, described.
3. Impact Analysis -Following activities are performed in this phase.
   i. Identify all systems and system products affected by a change request.
   ii. Make an estimate of the resources needed to effect the change.

iii. Analyze the benefits of the change.

4. System Release Planning - In this phase the schedule and contents of software release is planned. The changes can be made to all types of software maintenance.
5. Change Implementation - The implementation of changes can be done by first designing the changes, then coding for these changes and finally testing the changes. Preferably the regression testing must be performed while testing the changes.
6. System Release - During the software release i) documentation ii) software iii) training iv) hardware changes v) data conversion should be described.

5.14 Architectural Evoultion

Sometimes many legacy systems need to be changed from a centralised architecture to a distributed architecture like client server. This process is called architectural evolution. This causes changes to

- Hardware costs. This is because servers are cheaper than mainframes.
- The distributed system requires a fine User interface. In this type of system users want graphical user interfaces (GUI).
- The purpose of distributed system is to enable different users distributed access to systems. Users can access the system from different, geographically separated computers or from remote computers.

5.15 CASE TOOLS:

- The computer aided software engineering tools automate the project management activities, manage all the work products. The CASE tools assist to perform various activities such as analysis, design, coding and testing.

  Software engineers and project managers make use of CASE tools.

  The use of CASE tools in the software development process reduces the significant amount of efforts.

  CASE is used in conjunction with the process model that is chosen.

  CASE tools help software engineer to produce the high quality software efficiently.

5.15.1 Taxonomy of CASE Tools

To create an effective CASE environment, various categories of tools can be developed.

CASE tools can be classified by

1. by function or use
2. by user type (e.g. manager, tester), or
3. by stage in software engineering process (e.g. requirements, test)

The taxonomy of CASE tools is as given below.

(1) Business process engineering tools

This tool is used to model the business information flow. It represents business data objects, their relationships and how data objects flow between different business areas within a company.

(2) Process modeling and management tools

It models software processes. First the processes need to be understood then only it could be modeled. This tool represents the key elements of the processes. Hence it is pOSSible to carry out work tasks efficiently.

(3) Project planning tools

These tools help to plan and schedule projects. Examples are PERT and CPM. The objective of this tool is finding parallelism and eliminating bottlenecks in the projects.

(4) Risk analysis tools

It helps in identifying potential risks. These tools are useful for building the risk table and thereby providing detailed guidance in identification and analysis of risks. Using this tool one can categorize the risks as catastrophic, critical, marginal, or negligible. A cost is associated with each risk which can be calculated at each stage of development.

(5) Project management tools

These track the progress of the project. These tools are extension to the project planning tools and the use of these tools is to update plans and schedules.

(6) Requirements tracing tools

The objective of these tools is to provide a systematic approach to isolate customer requirements and then to trace these requirements in each stage of development.

(7) Metrics and management tools

These tools assist to capture specific metrics that provide an overall measure of quality. These tools are indetended to focus on process and product characteristics. For example "defects per function point", "LOC/person-month"

(8) Documentation tools

Most of the software development organizations spend lot of time in developing the documents. For this reason the documentation tools provide a way to develop documents efficiently. For example - word processors that give templates for the organization process documents.

(9) System software tools

These tools provide services to network system software, object management and distributed component support. For example - e-mail, bulletin boards, and www access.

(10) Quality assurance tools

These are actually metrics tools that audit source code to insure compliance with language standards.

(11) Database management tool

It provides consistent interfaces for the project for all data, In particular the configuration objects are primary repository elements.

(12) Software configuration management tools

It assists w:th identification, version control, change control, auditing, and status accounting.

(13) Analysis and design tools

It creates models of the system. Some create formal models. Others construct data flow models. These models contain representation of data, function and behavior. Such tools helps in architectural, component level and interface design.

(14) PRO/SIM tools

These are prototyping and simulation tools. They can help predict real time system response and allow mockups of such systems to be fashioned.

(15) Interface design and development tools

These tools are used in developing user interface. It includes various components such as menu, icons, buttons, scrolling mechanisms etc. For example - JAVA, Visual studio.

(16) Prototyping tools

These tools support to define screen layout rapidly for interactive applications.

(17) Programming tools

The programming tool category include the programs that support most of the (O!1\ cntion,11 progri1mming languages. For example - compilers, debuggers, editors, database query languages.

(18) Web development tools

These tools help in developing the web based applications. The various components of these tools are text, graphics, form, scripts, and applets.

(19) Integration and testing tools

These tools include various category of tools such as data acquisition tools, static measurement, dynamic measurement, simulation, cross functional tools.

(20) Static analysis tools

The static testing tools are used for deriving the test cases. There are three types of static testing tools.

I. Code based testing tools - These tools take source code as input and generate test cases.
II. Specialized testing language - Using this language the detailed test specification can be written for each test case.

III. Requirement-based testing tools - These tools help in designing the test cases as per user requirements.

(21) Dynamic analysis tools

These interact with an executing program, checking path coverage, and testing assertions. Intrusive tools insert code in the tested program. Non intrusive tools use a separate hardware processor.

(22) Test management tools

These tools manage and co-ordinate regression testing performs comparisons of output, and act as test drivers.

(23) Client/server testing tools

The client server tools are used in client server environment to exercise the GUI and network communication requirements for client and server.

(24) Reengineering tools

These tools performs a set of maintenance activities. These tools perform various functions such as

reverse engineering to specification tools.

coderestruchlring.

on-line system reengineering.

Review Questions

1. what are the different activities involved in project planning?
2. Explain the concept of risk management.
3. What is size measure?
4. Explain the function point mode.
5. What is COCOMO Model?
6. Explain the Delphi technique.
7. What is Earned Value Analysis?
8. Give the Lehman's Laws used in program evolution dynamics.
9. What is software maintenance? Explain the different types of software maintenance.
10. What is the use of CAS£7
11. Give the taxonomy of CASE tools.