Lesson No. 9
SAD (425)

SYSTEM ANALYSIS AND DESIGN **AUTHOR: ANKIT JINDAL**

DATABASE DESIGN

STRUCTURE

9.1 Objectives

9.2 Introduction

9.3 Database Design

9.3.1 Logical and Physical view of Data

9.3.2 Schema

9.3.3 Sub-Schema

- Self-Check Exercise

9.4 Types of Databases

9.4.1 Hierarchal Model

9.4.2 Network Model

9.4.3 Relational Model

- Self-Check Exercise

9.5 Summary

9.6 Keywords

9.7 Practice Questions

9.7.1 Short Answer Questions

9.7.2 Long Answer Questions

9.9 References

9.10 Answer-key

9.1 OBJECTIVES

After reading this chapter student may be able to know

- Understand the fundamental principles of effective Database Design, including the elimination of data redundancy, integration of data files, and implementation of a Database Management System (DBMS) for streamlined data management.
- Gain insight into the types of databases, such as hierarchical, network, and relational models, and grasp the significance of logical and physical views of data, schemas, and sub-schemas in achieving optimal database structures and functionality.

9.2 INTRODUCTION

In this chapter we study how system analyst is responsible for designing the files. We have also studied the various types of files with their advantages and limitations. we will see that what does the database means? Its different views, various types and coding system.

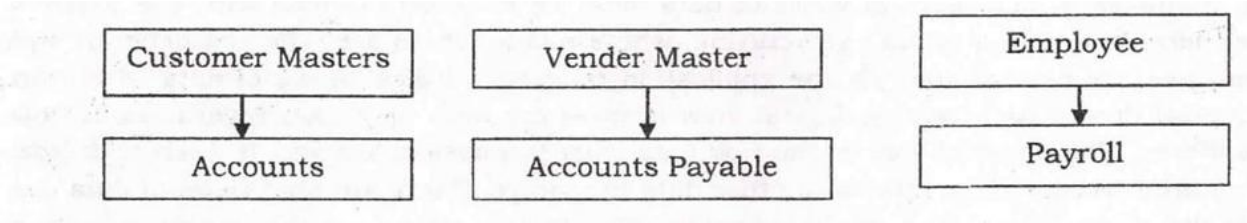
9.3 DATABASE DESIGN

A DATABASE can be thought of as a set of logically related files organized to facilitate access by one or more applications programs and to minimize data redundancy. In other words, a database can be defined as a stored collection of data rather than convenience of storage structures. It is not a replacement of file.Some general objectives in establishing a database are as follows :

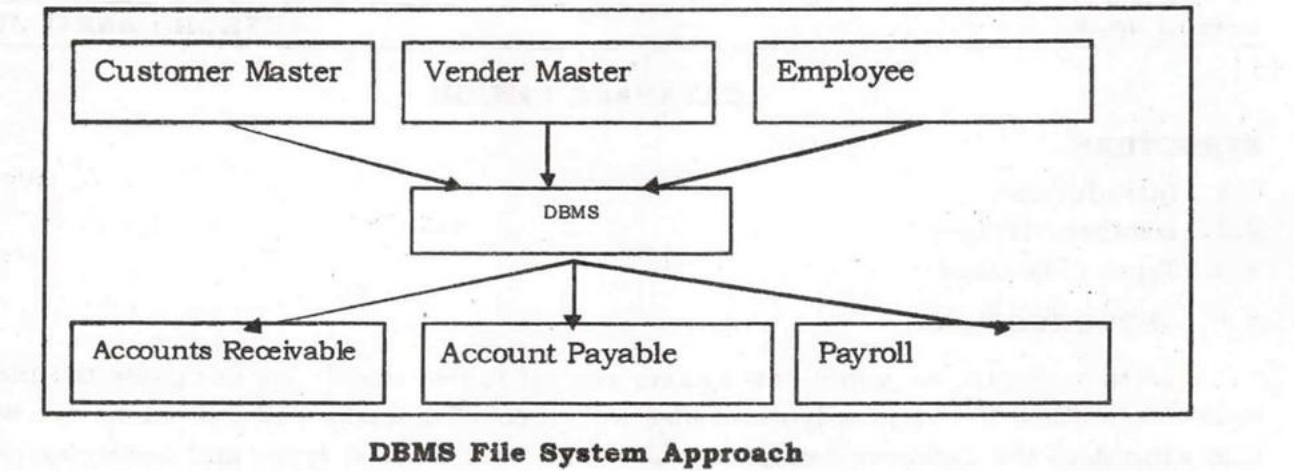
Eliminate redundant data as much as possible.

- Integrate existing data files.
- Share data among all users.
- Incorporate changes easily and quickly.
- Simplify the use of data files.
- Lower the cost of storing and retrieving data.
- Improve accuracy and consistency.
- Provide data security from unauthorized use.
- Exercise central control over standards.

In addition to the database itself, a set of programs is necessity to facilitate adding new data as well as modifying and retrieving existing data within a database. This set of programs is referred to as a Data Base Management System (DBMS).. The following figures defined the difference between the traditional file systems and database management system.



Traditional File System Approach



DBMS File System Approach

1. File Consolidation: Pooling data reduces redundancy and inconsistency and promotes cooperation among different users. Since data bases link records together logically, a data change in one system will cascade through all the other system using the data.

2. Program and file independence: This feature separate the definition of the files from their programs, allowing a programmer to concentrate on the logic of the program instead of precisely how to store and retrieve data.

3. Access Versatility: Users can retrieve data in many ways. They enjoy the best of both worlds-sequential access for reporting data in a prescribed order and random access for rapid retrieval of a specific record.

4. Data Security: Usually a DBMS includes a password system that contrails access to sensitive data. By limiting their access to read only. Write-only, or specified records, or even fields in records, passwords can present certain users from retrieving unauthorized data.

5. Program Development: Programmers must use standard names for data items rather than invent their own form program to program. This allows the programmer to focus on desired function.

б. Program Maintenance : Changes and reprints to a system are relatively easy.

7. Special Information: Special-purpose report generators can produce reports with minimum effort.

9.3.1 Logical and Physical view of Data

In database design, several views of data must be considered along with the persons who use them. In addition to data structuring, where relationships are reflected between and within entities, we need to identify the application program's logical views of data within an overall logical data structure. The logical view is what the data look like, regardless of how they are stored. The physical view is the way data exist in physical storage. It deals with how data are stored; accessed, or related to other data in storage. There are four views of data out of which three are logical and one is physical. The logical views are the user's view, the programmer's view and the overall logical view: called a schema.

9.3.2 Schema

Once a database system has been designed, it will be possible to identify each type of data item, data aggregate, and record and set by a name or code. It will be possible to state which data item types go together to make data aggregate types and record types, and to identify which record types are members and owners of set types. A coded set of tables describing this information and stored in the computer system on direct access devices is called a SCHEMA. It is a description of the data structure, which is separate from the data itself.

The schema describes the areas, their identifiers and page sizes, and indicates how these are related to the records and sets. In other systems, a different set of tables is used for this.

The schema therefore, is the view of the data, the overall logical data structure that is held by the DBMS. Each time a program requires data, the DBMS will look up in the sche ma for the details of the structure of the data requested. For example if the program requires itn occurrence of a set, the DBMS will look up in the schema which record types are required. how to find the relevant records given a certain key by the program, and perhaps also which areas the pages containing the relevant data are stored in.

9.3.3 Sub-Schema

In a database system, it is not always possible to allow programmers to write the data division of their choice for reasons of security or control. It is more useful to provide the programmer with a standard description of the logical data to be used in a particular application. All references to data within the program will be for this description, which is called a SUBSCHEMA and is similar to the Schema in structure. The DBMS has the job of matching data requests on a subschema and data requests based on the schema.

- **SELF-CHECK EXERCISE**

I.What is the primary responsibility of a system analyst in database design?

A) Coding programs
B) Designing files
C) Data retrieval
D) Hardware maintenance

II. What is a database in the context of information systems?

A) A collection of files
B) A set of hardware components
C) An organized collection of data
D) A type of programming language

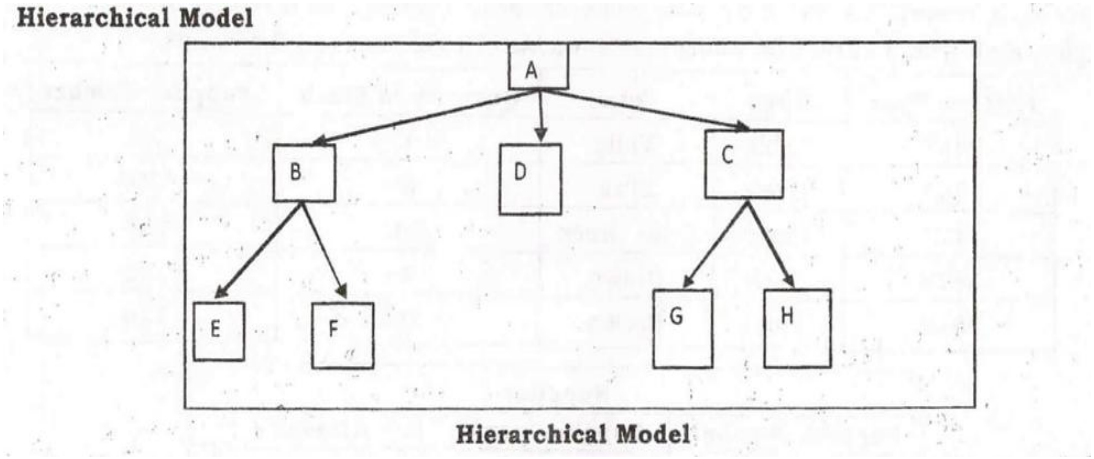
III. What is the role of a Data Base Management System (DBMS) in a database environment?
A) Storing physical files
B) Managing network connections
C) Facilitating data access and manipulation
D) Designing graphical interfaces

IV: What is the primary purpose of file consolidation in a database system?
A) Increasing data redundancy
B) Reducing data inconsistency
C) Slowing down data access
D) Complicating cooperation among users

V: In a database system, what does the term "Schema" refer to?
A) A programming language
B) A physical storage structure
C) An overall logical data structure
D) A report generation tool

9.4 TYPES OF DATABASE

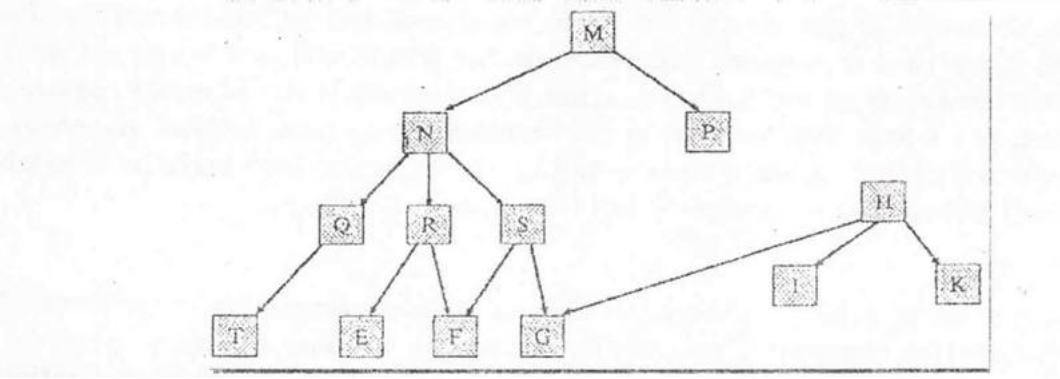
In conventional file systems, groups of bytes constitute a field, one or more fields make a record, and two or more records make a file. In a database environment, a group of bytes constitute a data item or segment, a collection of segments a data entry, and a series of data entries a data set. The complete collection of data sets is the database itself. With traditional processing of files, records are not automatically related, so a programmer must be concerned with record relationships. Often the files are stored and processed by record key, just as we sorted the transaction file. Databases relate data sets in one of three models: hierarchical, network, or relational.**9.4.1**



In a hierarchical structure, sometimes referred to as a tree structure, the stored data get more and more detailed as one branches further and further out on the tree. Each segment, or node, may be subdivided into two or more subordinate nodes, which can be further subdivided into two or more additional nodes, However, each node can have only one "parent" from which it emanates.

9.4.2 Network Model

The network related data sets are similar to hierarchical ones, except that a node may have more than one parent. Thus a hierarchical DBMS is a subset of network DBMS. The trade off between the simplicity of design of a hierarchical structure and the storage efficiency of a network structure is a very important consideration in database implementation.



9.4.3 Relational Model

The relational structure, however, organizes the data in terms of two-dimensional tables.

That is, Relational data sets order data in a table of rows and columns and differ markedly from their hierarchical or network counterparts. There are no parent or node data sets as shown in Fig. E. In relational database management systems, we have the same concept of files, records, and fields. Files are represented by two-dimensional tables, each of which is called a "relation". Records, which can be visualized as rows in the table, are called "Tuples". Fields can be visualized as columns, and are called by attribute names, or domains.

For example, note that in the supplier table. we have three Tuples, or rows, and three attribute names or columns. If we need to know the name of the supplier of blue chairs, the relational DBMS searches the type and color columns of the Furniture, Table and finds supplier number 30, and then it scans the supplier table for number 30, which turns out to be PANKAJ'S. Since each "record" is a row in the table and each "field" a column, an inventory system of 1600 Luples, each with 5 attributes, would create a table of 1600 rows and 5 columns.

Product Type	Type	Color	Quantity in Stock	Supplier Number
2589	Table	White	4	26
2892	Chair	Blue	6	30
3471	Chair	Light Green	20	133
3678	Desk	Brown	9	150
3689	Stool	Brown	25	159

Supplier		
Supplier Number	**Supplier name**	**Amount**
30	PANKAJ	26035
26	DINESH	13960
159	RAJESH	75286

❖ **SELF-CHECK EXERCISE**

VI. What is the primary characteristic of a hierarchical structure in database modeling?
A) Two or more subordinate nodes
B) More than one parent for each node
C) Only one "parent" for each node
D) Two-dimensional tables

VII. How does the network model differ from the hierarchical model?
A) Two or more subordinate nodes
B) Only one "parent" for each node
C) More than one parent for each node
D) Two-dimensional tables

VIII. What is the organizational structure of the relational model based on?
A) Three-dimensional tables
B) Hierarchical tree structure
C) Two-dimensional tables
D) Network connections

IX. What are rows called in a relational database table?
A) Tuples
B) Domains
C) Nodes
D) Fields

X. An inventory system of 1600 Luples, each with 5 attributes, would create a table of _____ rows and _____ columns.

9.5 SUMMARY

In this Chapter, the focus is on Database Design, exploring its types and structures. The chapter begins with an introduction to the responsibilities of a system analyst in designing files, emphasizing the importance of eliminating data redundancy and facilitating data access. The objectives of establishing a database are outlined, including the integration of existing data files, data sharing among users, and ensuring data security. The chapter introduces a Database Management System (DBMS) and highlights

the differences between traditional file systems and DBMS. It delves into logical and physical views of data, introducing the concept of a schema to describe the overall logical data structure. The sub-schema is also discussed, providing programmers with a standard description of logical data for a specific application. The chapter concludes by presenting three types of databases: hierarchical, network, and relational.

9.6 KEYWORDS

1.Database Design: The process of defining the structure that will organize and store data in a database system, considering factors such as data integrity, security, and efficiency.

2.Data Redundancy: The repetition of data within a database, leading to inefficiency and inconsistency, which database design aims to eliminate or minimize.

3.Schema: Meaning: A coded set of tables that describes the overall logical data structure in a database, specifying relationships between data items, aggregates, records, and sets.

4.DBMS (Database Management System): A set of programs and tools that manage the storage, retrieval, and manipulation of data.

9.7 PRACTICE QUESTIONS

9.7.1 Short Answer type Questions

Q1- Define Data Base Management System ?

Q2- What characterizes the relational model in terms of data organization?

Q3- What is a schema, and what role does it play in a database system?

Q4- What are the general objectives in establishing a database?

9.7.2 Long Answer type Questions

Q1- Elaborate on the logical and physical views of data in the context of database design. What are the four views of data, and how do they contribute to the overall structure?

Q2- Define and describe the concept of a schema in a database system. How does it facilitate data organization, and what role does it play in program execution?

Q3- Explore the various types of databases mentioned in the text (hierarchical, network, and relational). Provide insights into their structures and trade-offs in implementation.

Q4- Explain the difference between file systems and database management systems, citing specific features.

9.9 References

Elmasri, R., & Navathe, S. B. (2016). "Fundamentals of Database Systems." Pearson.

Connolly, T., & Begg, C. (2014). "Database Systems: A Practical Approach to Design, Implementation, and Management." Pearson.

Kroenke, D. M., & Auer, D. J. (2018). "Database Processing: Fundamentals, Design, and Implementation." Pearson.

Coronel, C., & Morris, S. (2016). "Database Systems: Design, Implementation, & Management." Cengage Learning.

Rob, P., & Coronel, C. (2019). "Database Systems: Design, Implementation, and Management." Cengage Learning.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). "Database System Concepts." McGraw-Hill Education.

Hoffer, J. A., Venkataraman, R., & Topi, H. (2017). "Modern Database Management." Pearson.

Ramakrishnan, R., & Gehrke, J. (2002). "Database Management Systems." McGraw-Hill.

Sipser, M. (2012). "Introduction to the Theory of Computation." Cengage Learning.

Kent, W. (1978). "Data and Reality: Timeless Perspective on Perceiving and Managing Information in Our Imprecise World." Elsevier.

Hernandez, M. J. (2013). "Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design." Addison-Wesley.

Rosenfeld, L., Morville, P., & Arango, J. (2015). "Information Architecture: For the Web and Beyond." O'Reilly Media.

9.10 Answer key (Self- Check Exercise)

I. B, II. C , III. C ,IV. B, V. C, VI. C, VII. C, VIII.C, IX. A , X 1600,5.

Semester-IV

Lesson No. 10
SAD (425)

SYSTEM ANALYSIS AND DESIGN **AUTHOR: ANKIT JINDAL**

DOCUMENTATION TOOLS

STRUCTURE

10.0 Objective

10.1 Introduction

10.2 Characteristics of a good design

10.3 Types of Documentation

10.3.1 Program Documentation

10.3.2 Operation Documentation

10.3.3 User Documentation

10.3.4 Management Documentation

10.3.5 Systems Documentation

- Self-Check Exercise

10.4 Software Design and Documentation Tools

10.4.1 Structured Flowchart

10.4.2 HIPO diagrams

10.4.3 Warnier/ Orr diagrams

- Self-Check Exercise

10.5 Need for Documentation

10.6 Guidelines for preparing Documentation Package

10.7 Elements that Comprise A Documentation Package

- Self-Check Exercise

10.8 Summary

10.9 Keywords

10.10 Practice Questions

10.10.1 Short Answer type Questions

10.10.2 Long Answer type Questions

10.11 References

10.12 Answer key

10.0 OBJECTIVE

After reading this chapter student may :

Understand the importance of documentation in the system development process and recognize the characteristics of good design documentation.

Identify the types of documentation and their specific purposes and learn about software design and documentation tools.

Explore the need for documentation in system development and gain insights into guidelines for preparing a documentation package.

10.1 INTRODUCTION

A system is not completely effective unless it is adequately documented. It should be documented as it is being created. That is, at various stages of the system development. Status report should be prepared for those management personnel for whom the system is being designed. Such reports could include flowcharts, decision tables, output forms and other documents thus far developed. It also includes various problems encountered, suggested solutions and resulting schedule revisions. In this way, management remains full aware of system's progress so that they can offer criticisms or suggest change while it being necessary to revise the entire system. These progress reports provide an excellent basis on which to build additional information.

Instructions and narrative descriptions must be prepared for every phase and part of the system, including system logic, tunings, user instructions, guidelines for operations staff in the data processing center and instructions relating to transmission of data and results.

Much of this can be incorporated into procedural manual. This manual stipulates the relationship between personnel in the application areas affected by the system and the data processing center. It should relate in details, exactly what procedures must be employed by the user to operate the system efficiently and effectively.

10.2 CHARACTERISTICS OF A GOOD DESIGN

Documentation is considered to be good if it has the following qualities :

(a) Availability: It should be accessible to those for whom it is intended.

(b) Objective: It must be clearly defined in a language that is easily understood.

(c) Cross-referable : It should be possible to refer to other documents.

(d) Easy to maintain: When the system gets modified, it should be easy to update the documentation.

(e)Completeness: It should contains everything needed, so that those who have gone through it carefully can understand the system.

10.3TYPES OF DOCUMENTATION

There are five major types of documentation. They are

(i) Program Documentation

(ii) Operation Documentation

(iii) User Documentation

(iv) Management Documentation

(V) Systems Documentation

10.3.1 Program Documentation

Many companies discuss about programming documentation but fail to provide it adequately. Before a program is developed, the systems analyst provides the programmer with the required documentation.

The logic in some programs is best described by a flowchart.Sometimes, decision tables are mostappropriate for explaining the logic of a program.
Programmers should insist on proper documentation before starting a job.
Four items constitute normal documentation required for each program.

- Copying in final form of all input/output documents affecting the program.
- Statement of standards for coding structures and input/output layouts.
- Clarification of the program's interface with other related programs.
- General flowchart for decision table.

The programmer's responsibility in documentation is to provide information to enable future programmers to make necessary changes. Personnel turnover is normal feature in any business and turnover is particularly high among programmers. A company can never think that a programmer assigned to a specific program will be available in two years, when some modifications to that program are required. For continuity of information a company must insist on complete and meaningful documentation. Typically, a Documentation folder is provided for each program, which contains all input/output form, associated with the program, a detailed flowchart for the program use a set of operator and user instructions.

Maintaining the type of Documentation is costly and time consuming, for, programmers do not take interest in spending time for this type of work. Routine changes occur frequently in a program and all changes must be covered in the Documentation folder. But the very changes, which require the updating of existing documentation, are the reasons for maintaining accurate documentation

10.3.2 Operations Documentation

A well-designed system may run for a long time with little or no assistance from thesystems department. This can happen only when the system has been documented in a proper way. Forsmooth running of the system, the console operator must have complete knowledge about the job. Providing the computer center with the set of operating instructions will not serve the purpose. The instructions must be in a form readily accessible to the console operator and written in simple and understandable style. A system analyst must thoroughly discuss all the requirements of new jobs with the operations staff before the job can be properly transferred.

10.3.3 User Documentation

Systems users require proper documentation to prepare a developing system and to smoothly carry out existing ones. To meet this requirement, each system should have a manual that spells everything the users must know to do their job correctly. Users require two general types of information; complete details to handle each case the system processes, and overall picture of the system so that they can see their role in the total operations of the company.

The manual should supply the following information :

- General flowchart of the system
- Assignment of responsibility for specific tasks
- Standards for work flow including target dates and deadlines for specific tasks
- Simple input and output documents
- Detailed procedures
- Anticipated exceptions and instructions on how to handle them
- Accuracy standards for data in the system

The systems department must write a thoroughly detailed narrative of each system, including the proper handling of routine cases, as well as exception handling. A staff member in the user department must have an authority to consult when faced with a case not handled before. Properly prepared manual, which is always, can provide the information needed by the user. This requires documentation, in the form of charts, graphs and illustrations, so that the supervising staffs have a clear grasp of their department's role in the total system.

10.3.4 Management Documentation

The documentation required by corporate management differs quite a lot that required by the users. The systems designer must know the requirements of the management and provide documentation to enable management to perform three functions:

Evaluate progress on system development

• Monitor existing systems

• Understand the objectives and methods of new and existing system

Management is primarily interested to know in general the system's overall and basic operations. A brief manual highlighting the key steps in each system may be prepared for management. Good managers have an exceptional ability to get to the root of the system and, their experience should enable them to retrieve information from a systems summary or chart, which may not be apparent to the system analysts.

10.3.5 Systems Documentation

Each phase in the systems development cycle is accompanied by appropriate documentation. The systems request, even if it is initially mark verbally, eventually must be written. It is desirable for the client and a systems analyst to work jointly in writing the request since each can contribute knowledge the other does not have. The written system'srequest is merely a statement of the user's problem.

In documenting the results of its deliberations, the selection committee must specify the following :

(i) The objective of the impending feasibility study.
(ii) The extent of the authority of the feasibility team.
(iii) The individual or group responsible for completing the study.

A Feasibility report is probably the most important form of documentation in the system development life cycle. It accomplishes the following two purposes:

(i) It defines the objectives of the proposed system's change in reasonable detail after a sufficiently detailed study.

(ii) Itgives a plan to attain these objectives.

The documentation of this plan must be thorough enough file in the system designers could produce a complete and effective system. At various points during systems design, the designing team produces the following additional forms of documentation :

(i) File Specification: Detailed definitions of each in the system, best done in graphic form.

(ii)Transaction Specifications : Detailed descriptions of each type of input in thesystem, including a layout of each transaction and narrative description of how it is used.

(iii) Output Specifications: Detailed descriptions of all output anticipated from thesystem.

Documentation also includes plans to test the system and convert from the old to the new one. The systems analyst must also provide a plan to train the personnel affected by the Changes. During the life cycle of the completed system, the system itself must provide documentation of how well it is operating and consequently should be designed to yield data about itself as a normal by-product.

❖ **SELF-CHECK EXERCISE**

1. What is the primary purpose of documenting a system during its development stages?
a. To impress management personnel
b. To satisfy auditors
c. To create a historical record
d. To ensure system effectiveness

2. Which characteristic is essential for good documentation?

a. Redundancy
b. Ambiguity
c. Accessibility
d. Complexity

3. Which type of documentation focuses on the logic of a program using graphical representation?
a. Program Documentation
b. Operation Documentation
c. User Documentation
d. Systems Documentation

4. Programmers should insist on proper documentation before starting a job to ensure _____ information for future modifications.

5. A Feasibility report is an important form of documentation in the system development life cycle as it defines the _____ of the proposed system's change.

10.4 SOFTWARE DESIGN AND DOCUMENTATION TOOLS

Well-designed, modular software is more likely to meet the maintenance, reliability-and testing requirement. Three specific tools are described below:

- Structured Flowchart
- HIPO diagrams
- Warnier/ Orr diagrams

10.4.1 Structured Flowchart

Structured Flowcharts is a graphic tool that forces the designer to structure software in modular as well as top-down form. They provide a proper structure that can be retained by the programmer fordeveloping the application software. The programmer should be expert in using the structured flowcharts are:

The basic elements are used in developing structured flowcharts are :

- Process
- Decision
- Iteration

Process: Simple processes or a rectangular box, the process symbol, shows steps in a program. This symbol represents initialization of values, input and output operations andcalls to execute other procedures.

Decision : The decision symbol represents alternative conditions that can occur and that the program must have a manner of handling. The decision symbol may show actions for more than two alternatives at the same time.

Iteration: The iteration symbol represents looping and repetition of operations while a certain condition exists or until a condition exists.

Structured flowchart uses no arrows or continuations on separate pages, each structured flowchart is shown on a single sheet of paper. When designing a structured flowchart, the system analyst specifies the logic in a top down fashion. The first consideration in a process is the top element. The second in sequence is next one shown and so forth.Similarly, there is a single exists from the process.

10.4.2 HIPO DIAGRAM

The HIPO documentation uses a structure that is similar to an organization chart. This type of structure allows the enforcement of major principles to HIPO, a top-to-bottom approach to design. The emphasis is made on forcing the flow of data down through the system, not in the opposite direction. The main idea behind the top-to-bottom approach is the elimination of

"output-oriented" systems solutions. An output-oriented system is concerned with providing output and does not bother about the sound principles of system design. In essence, an output-oriented system often gets the job done without delay.

Unfortunately, many data processing organizations try to employ this type of rationale in their system designs. Output-oriented systems are often fragmentary, with large gaps evident in the logic and flow of data throughout the system. Programs written for this type of system often duplicate each other in part. The net effect is that more programming efforts are required, with a resultant, loss of manpower and time.

The HIPO concept, with its highly ordered structure and top-to-bottom approach, tries to eliminate piecemeal system design. A view of HIPO chart structure is shown in the below fig.1.

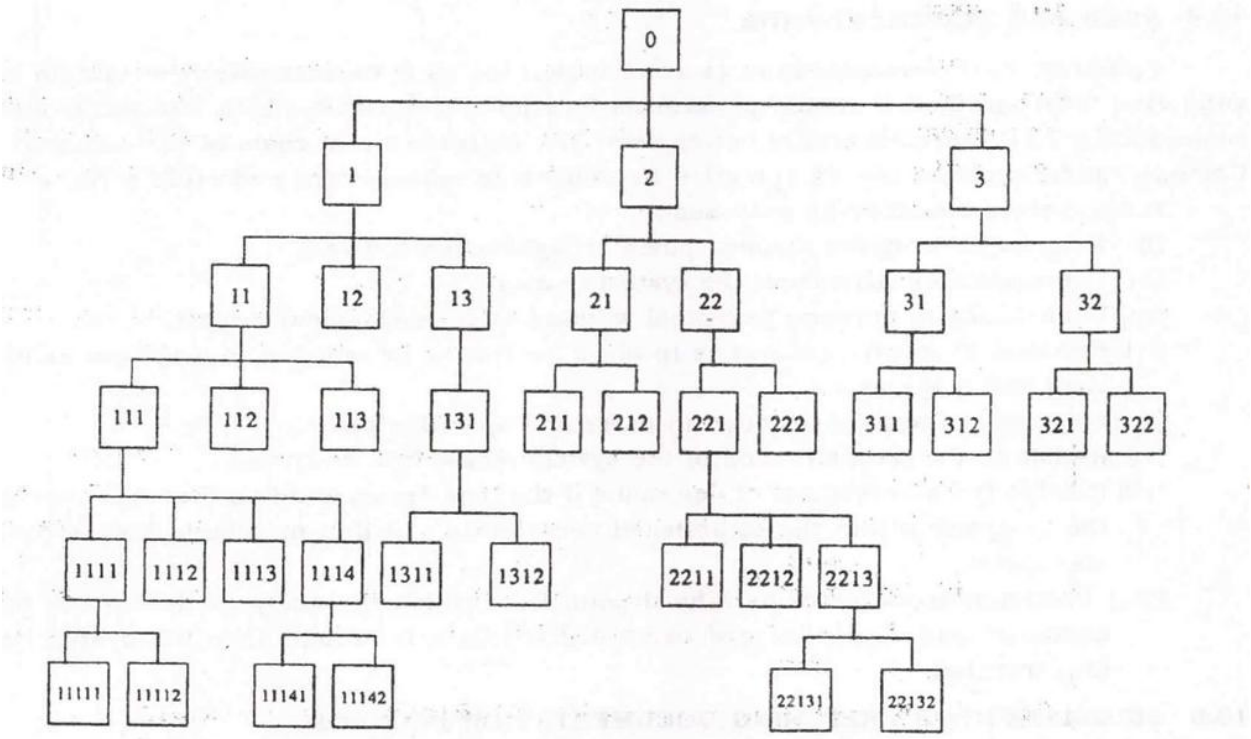


Figure 1: Structure of HIPO chart identifying all of the components within the chart by at each of the sub-level.

As you can observe that HIPO chart is quite similar to that of the manager's organizational chart. The numbers are shown in various boxes of HIPO chart provides a means of identifying each of its sub-levels and components blocks on the chart. The rationale of subdividing and identifying the components blocks within the HIPO design is extremely

important. Applying this concept, the analyst is capable of defining and completely laying out the overall structure of the entire system under study. The HIPO approach is mainly designed to accommodate the development of a system

10.4.3 Warnier/Orr Diagram

Warnier/Orr Diagram, also known as logical construction of programs/logical construction of systems are powerful tools aimed at designing of program structures by identifying the output and processing results and then working backwards to determine the steps and combinations of input needed to produce them. The simple graphic methods used in Warnier/Orr diagrams make the levels in the system evident and movement of the data between them vivid.

Warnier/Orr diagrams clearly show the various processes and sequences in which they are performed. Each process is defined in a hierarchical way. At each level, the process is shown in a bracket that groups its components. Since a process consists of different sub processes, a Warnier/Orr diagram employs a set of brackets to indicate each

Warnier/Orr diagrams are very powerful design tools and offer some distinct advantages to systems experts. They are quite simple in appearance and easy to understand.

- **SELF- CHECK EXERCISE**

  VI. What is the emphasis of the HIPO (Hierarchy plus Input-Process-Output) approach in system design?
  a. Bottom-up approach
  b. Output-oriented solutions
  c. Chaotic system flow
  d. Elimination of process blocks

  VII. HIPO charts are structured similarly to which organizational chart?
  a. Circular chart
  b. Manager's chart
  c. Horizontal chart
  d. Random chart

  VIII. Warnier/Orr diagrams aim to design program structures by identifying:
  a. Future technologies
  b. Past failures
  c. Output and processing results
  d. Random symbols

  IX. The HIPO chart structure is similar to a manager's organizational chart, providing a means of identifying each sub-level and component block within the chart by using _____.

10.5 NEED FOR DOCUMENTATION

Preparation of documentation is quite important as it depicts what the system is supported to be and how it should perform its functions. It illustrates both technically and economically how a system would better serve the objectives and goals of the company.

Documentation improves overall operation in addition to management and audit control.

It also serves the following purposes

(i) Reviews the progress or development of application software.
(ii) Communicates facts about the system to users.
(iii) Communicates between personnel working on a development project.
(iv) Provides necessary guidelines to allow correction or revision of a system or itscomputer programs.
(v) Provides operating instruction to users and operating staff.
(vi) Assists in the reconstruction of the system incase it is destroyed.
(vii) It helps the management to determine if the new design archives the objectives of the company within the established constraints and if is justifiable from a cost standpoint.
(viii) Documentation serves as a focal point from which the analysts' design can be assessed and as a standard to be utilized as a reference once the system is implemented.

10.6 GUIDELINES FOR PREPARING DOCUMENTATION PACKAGE

There are, as yet, no universal documentation standards, since systems vary greatly inform, contents and requirements. The format of each documentation package will be based on the following points:

(1) Characteristics of system : Some designs require descriptive while others can be explained with the help of diagrams.

(ii) Managements' attitude towards documentation: The analyst must prepare the documentation package within the limitations established by the management.

(iii) Equipment restraints: A company with large and integrated computer system having teleprocessing facilities will require more formalized and technical documentation than a company with a more conservative and small computers system.

10.7 ELEMENTS THAT COMPRISE A DOCUMENTATION PACKAGE

A documentation package consists of the following elements :

- Cover Letter
- Table of Contents
- Narrative
- Flow Charts
- File Specification
- Program Specification
- Cost of the proposed system and of its alternatives
- Test brochures

Cover Letter: The cover letter is a correspondence primarily to management, that describes the benefits of the new design and that generally helps in selling the system. It should be kept in mind that unless the documentation is approved, the new system will never be implemented and the analyst's work will be of no use.

Table of Contents : The inclusion of a table of contents is an absolute necessity. Pages in the documentation package must be numbered and cross-referenced in this table of contents.

Narrative: With the narrative, we begin the detailed formulation of the new system.

Flowcharts : Each subsystem within the analyst's formal design should be explained with the help of flowchart.

File Specification : Each of files within the formal design must be described with regardto :

- Purpose
- Programs that will use the file
- Volume
- Frequency of use
- Source from which the file is obtained
- Description of fields
- Layout and samples

Program Specification : At this point, the analyst must segment the new design so that each unit will have separate program, assuming that the design is itself approved by the management.

Cost of the proposed system and of its alternatives: The details of the cost must be shown as part of documentation.

Test Brochures : The analyst should describe the operations and procedures that will be employed to test he new system, once it is approved.

- **SELF CHECK EXERCISE**

  X. What element in a documentation package serves as a focal point for assessing the analyst's design?

  a. Test brochures

  b. File specification

c. Narrative
d. Table of contents

XI. Which factor influences the level of formalization in technical documentation?
a. Complexity of the system
b. Avoiding documentation
c. Lack of equipment restraints
d. Disregard for management's attitude

10.8 Summary

The chapter emphasizes the significance of documenting a system throughout its development stages. It discusses the characteristics of good design documentation, such as availability, objectivity, cross-referability, ease of maintenance, and completeness. Various types of documentation, including program, operations, user, management, and systems documentation, are explored, each serving a distinct purpose. The chapter introduces software design and documentation tools, including structured flowcharts, HIPO diagrams, and Warnier/Orr diagrams. It concludes with the need for documentation and guidelines for preparing a documentation package, detailing the elements that comprise it.

10.9 Keywords

1.Documentation: Recording and detailing the various aspects of a system throughout its development.

2.Structured Flowcharts: Graphic tools enforcing modular and top-down software design.

3. HIPO Diagrams: Hierarchical Input-Process-Output diagrams used for system design.

4. Warnier/Orr Diagrams: Logical construction tools for designing program structures.

5. System Development Life Cycle (SDLC): The process of planning, creating, testing, and deploying an information system.

10.10 Practice Questions

10.10.1 Short Answer Type Questions

Q1- Why is documentation considered an essential aspect of system development?

Q2- Explain the purpose of program documentation and its components.

Q3- Why is operations documentation crucial for the smooth running of a system?

Q4- What information should be included in user documentation to facilitate system usage?

10.10.2 Long Answer Type Questions

Q1-Explain the role of documentation in the system development life cycle (SDLC). How does it contribute to the effectiveness of a developed system?

Q2-Compare and contrast structured flowcharts, HIPO diagrams, and Warnier/Orr diagrams as tools for software design. What are the advantages and limitations of each approach?

Q3- Discuss the significance of user documentation in system development. How does well-prepared user documentation contribute to the successful implementation and operation of a new system?

10.11 References

Kim, H., & Lee, H. (2014). Database design methodology for NoSQL systems. Journal of Software Engineering and Applications, 7(04), 258.

Coronel, C., Morris, S., & Rob, P. (2016). Database Systems: Design, Implementation, and Management. Cengage Learning.

Whitten, J. L., & Bentley, L. D. (2007). Systems Analysis and Design Methods. Irwin/McGraw-Hill.

Pressman, R. S. (2014). Software Engineering: A Practitioner's Approach. McGraw-Hill Education.

Hernandez, M. J. (2003). Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design. Addison-Wesley.

Shelly, G. B., Rosenblatt, H. J., & Cashman, T. J. (2004). Systems Analysis and Design: An Applied Approach. Course Technology.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). Database System Concepts. McGraw-Hill.

Dennis, A., Wixom, B. H., & Tegarden, D. (2015). Systems Analysis and Design: An Object-Oriented Approach with UML. Wiley.

10.12 Answer-key (Self-Check Exercise)

I.D, ii.C, III.A, IV. Accurate, V.Objective , VI. B, VII. B, VIII.C, IX. Numbers, X.C, XI.A.

Semester-IV

Lesson No. 11

SYSTEM ANALYSIS AND DESIGN **AUTHOR: SANJEEV KUMAR MODI**

SYSTEM ADMINISTRATION & TRAINING

STRUCTURE

11.0 Objective

11.1 Introduction

11.2 Training

- Self Check Exercise

11.3 Choosing Training

11.4 Targeting

11.5 Classroom Training

11.6 Parallel Run

- Self Check Exercise

11.7 Summary

11.8 Keywords

11.9 Practice Questions

11.9.1 Short Answer type Questions

11.9.2 Long Answer type Questions

11.10 References

11.11 Answer key

11.0 OBJECTIVE

After reading this chapter, student will be able to

To emphasize the significance of well-timed and relevant training in the successful implementation of software systems.

To highlight the various types of training required for different user groups in the context of system development.

11.1 INTRODUCTION

The successful implementation of any software is dependent in good measure on the quality of training imparted, Different user groups need to be identified. A Training Need Analysis has to be done. for each of these groups to find out what kind of training is required for each. Getting trained on aspects that are not relevant may sometimes tend to confuse the users.

So the training program has to be very carefully worked out. One aspect that tends to get ignored by most people is the timing of the training. In an organization's enthusiasm to proceed with things the training programs may be scheduled very much in advance. By the time the software is ready for implementation,

most users would have already forgotten what was taught to them earlier. It is important therefore to time the training programs appropriately. Hands-on training can follow this up on the software once.

One of the biggest nuisances about installing new technology is staff training; even though IT manufacturers never stop saying their new products will give no trouble. The language they use to state this has changed over time, from "user-friendly" to "easy to use" to "usable", but the reality behind it has stayed the same: learning a new product is never straightforward. In fact, as design guru Donald Norman has pointed out, manufacturers do themselves a disservice with these claims. If they said honestly, "these products will take some learning to use effectively, but the results will be worth it," users would not lose so much confidence when they first started with a new product and found it difficult.

Often designed by the sort of people who use instead of write it, and it would also help if manuals were better written and designed. We are now starting to see the development of more intelligent online help systems, where the user can ask "How do I make this text larger?" instead of having to look up "font - increasing point size" in the index. Assuming that it's in the index at all. These systems are still far from widespread. But Lotus, WordPerfect and Microsoft, to name but three, are all working on them. The solution to a user's problem isn't always to call technical support. Aside from the fact that many manufacturers' support hotlines take a long time to answer questions, teaching users isn't what tech support is intended for. It's fair to call technical support if the documentation on a particular feature is poor, or if a user has a specific question; it's not fair to expect technical support to teach people to write macros, or to take a novice through the basics of navigating Windows.

A large training industry has grown up around computing: books, audiocassettes, videotapes, classroom courses, and computer-based training (CBT). Each has advantages and disadvantages depending on the type of user you have and the situation in which the products are going to be used. (Cynics suggest that the number of books available on a particular software package is proportional to the number of pirated copies of the software, which, by definition, lack manuals.)

11.2 TRAINING

The computerized environment is totally new. The hi-tech environment especially for management, staff and users not having earlier exposed to it needs to be trained thoroughly.

The users basically in our reference includes all personnel, executives, management, users, clients and others who so ever is getting affected with the implementation of the system development. An analyst is supposed to train the end -user who is going to work on the newly developed system at the client site. The analyst also provides training with respect to corrective actions in case of failures especially to operational staff.

As explained in the forgoing paragraphs the importance of training cannot be underestimated. It as been normally observed that the analysts being in contact with the users of the organization, he starts assuming that the users have got trained is to be drawn which will include the following types of training for different categories of the people in an organization.

11.2.1 Brainstorming Sessions.

Though the brainstorming session is not a part of training but it is a part of fact gathering techniques and unknowingly the participants can be made aware of the various features of the system and utilization of reports likely to be generated by the system.

11.2.2 Seminars

After having developed the system seminars for short durations like half day or one day but not exceeding two days can be organized for top level and middle level management in batches, if required so. In the seminar the emphasis has to laid on the various features of the system, use of various reports, query methods, and utilization of query reports and finally benefits in the decision making process through the developed system. After the seminars if required a practical workshop can be organized for middle level management in order to have proper control over the operational staff on smooth implementation of the system.

11.2.3 Computer awareness and Technical Training

This type of training can be given to the employees of the organization after a proper screening about the aptitude and technical skill of individuals. Under this training programme the proper contents of the course can be decided depending upon the type of platform software environments and required packages required in the organization.

11.2.4 Operational Training

The production of the system will be successful on proper operation of system right from the beginning that is filling of input firms, encoding the data, and recording the data, preparing updates for files, preparation of transactions, maintenance of files and backups, knowledge of recovery procedures and maintenance of security and accounting of end-users and so on.

- **Self – Check Exercise**

I. What is a crucial factor for the successful implementation of software?
a. Software cost
b. Quality of training
c. User manuals
d. System complexity

II. Why is timing important in training programs?
a. To confuse users
b. To align with the software release
c. To save costs
d. To eliminate the need for hands-on training

III. According to design guru Donald Norman, what is a common issue with product claims?
a. Overemphasis on user-friendliness
b. Lack of technological innovation
c. Misleading language about ease of use
d. Inadequate technical support

IV. What is the purpose of brainstorming sessions in the context of training?
a. Corrective actions
b. Fact gathering
c. Operational procedures
d. Technical skill assessment

V. Which training type emphasizes the use of various reports and query methods?
a. Seminars
b. Brainstorming sessions
c. Computer awareness
d. Operational training

11.3 CHOOSING TRAINING

In making a decision, the first question to consider is the computer literacy - or lack thereof - of a given system's users. People who are already uncomfortable with technology may not react well to computer-based training packages, even though CBT seems like a logical way of showing someone how a program works. People in this position are likely to feel more comfortable with familiar types of materials - books, for example, which can be laid flat on the desk in front of the PC and used as step-by-step guides, or classroom courses that feature live humans.

On the other hand, very bright, Computer-literate people tend to be bored by classroom courses, and may be happier using books, tapes, or CBT, where they can set their own pace.

Classroom courses are generally designed for "average" users, which means they're too slow for the people who tended to jump ahead in the textbooks in their school days, and possibly too fast for the least experienced technophobes. Trainers do try to accommodate various grades of user - small classes mean they spend some time helping each trainee come to grips with the exercises they generally assign - but within a two- or three-day course intended to teach all of a program like Excel they can't do remedial teaching. As classroom courses tend to be expensive, it's important to make sure you have a good match between the types of course and the skills and aptitudes of your user population. At the same time, it's important to think about what your users actually need to learn. In the case of a DTP package, for example, they may not need to learn everything the package can do as much as they need to learn how to produce specific materials such as a company newsletter, letterhead, or memos.

11.4 TARGETING

Another problem with all types of commercial training is that everything targets mainstream applications, operating systems, and programming languages. Even the cheapest form of training - books - focuses on the top sellers, with hundreds of titles on Windows, DOS, the Internet and the common Novell, Lotus, and Microsoft office applications, but almost nothing on well-known niche applications like contact managers or personal information managers. CBT has come a long way since scientists first started talking about the idea of teaching machines back in the 1950s. For a long time, CBT seemed to be limited to putting up pages of text on screen where it was less convenient and less comfortable to read than in a book. These days, CBT comes in a variety of interactive forms that can be very effective in teaching people the ropes in a new system. The one thing to remember is that CBT can be very intimidating to completely inexperienced users. If you have never seen a mouse, for example, a mouse-driven tutorial will seem extraordinarily difficult and cumbersome - it's easy to forget that you have to practice to acquire the muscular control you need.

11.5 CLASSROOM TRAINING

Classroom courses are probably the easiest to find of all types of training except books, although they are the most expensive. You can pay anything up to around $3000 for a place on a 3 or 4-day course. All sorts of organizations offer courses. In ascending order of cost these include local colleges, user groups, and corporate training companies.

Many software companies maintain lists of authorized training centers. This is in addition to programme run by companies like Novell and Microsoft that offer special

certification for certain types of support engineers and applications developers. Local colleges and corporate training companies usually list themselves in the telephone book. User groups are a little more obscure. Try asking your software publisher or supplier if there are relevant groups with nearby offices. Groups tend to vary widely in what services they offer and at what cost. However, most user groups offer some form of technical support. Training from these groups isn't always cheaper, but sometimes it fills in gaps left by more mainstream organizations.

Corporate training outfits also vary a great deal, and many offer customized services, either tailoring the material the course covers to the exact software add-ons and templates a company has, or adapting their delivery methods to suit staff schedules. You might, for example, find it more efficient in staff time to hire a trainer to spend days on the company premises answering user questions by appointment in private, one-on-one sessions rather than conducting all-day standard classroom sessions. This technique is especially useful for senior staff, some of whom may not learn well in a classroom-based course for other reasons than scheduling: some are embarrassed to admit they don't know or don't understand something, especially in front of junior staff.

11.6 PARALLEL RUN

Once the software is installed, the users are trained and the master data is created, it is moment the software is commissioned and goes live, the organization relies solely on it, discarding all previously used processes and procedures.

What generally happens is that usage of the software is commenced? In addition, the old system is also continued till such time that stability is reached and all the users are confident that the new system is foolproof and that the old one can be dispensed with

Therefore the basic idea underlying this activity is that even after the software is installed, the old system of doing things be it automated or manual is continued for some period of time. Since both the old and the new systems continue in parallel, this is called the Parallel Run.

The period for the parallel run varies from organization to organization and from application to application. Although there are no hard and fast rules for this phase, it is attempted to ensure that the following factors are looked into :

- There are no major faults in the package.
- Important figures tally.
- All modules are used and found satisfactory.
- There is a fair amount of confidence amongst users is being able to use the software.

If there are any gaps in any of the above factors, the problems are plugged. If there are no problems encountered then a decision is taken to discontinue the old system and rely exclusively on the new software. This connotes the end of the parallel run.

❖ **SELF-CHECK EXERCISE**

VI. In classroom courses, small classes allow trainers to spend time helping each trainee come to grips with the exercises they assign, but they can't do _____ teaching.

VII. The period for the parallel run varies, but it is essential to ensure there are no major faults in the _____, and all modules are used and found satisfactory.

VIII. What is a consideration when choosing training methods?
a. Software cost
b. User computer literacy
c. System complexity
d. Availability of technical support

IX. Why might computer-literate individuals prefer books, tapes, or CBT for training?
a. Slow pace of classroom courses
b. Availability of technical support
c. Cost-effectiveness
d. Customization of materials

11.7 SUMMARY

The proper implementation of software systems hinges on the quality of training provided to diverse user groups. A Training Need Analysis must precede training programs to identify specific requirements for each group, ensuring relevance and avoiding confusion. The timing of training is crucial, as scheduling programs too far in advance may lead to forgetfulness by the time the software is ready for implementation. The training process involves different types such as brainstorming sessions, seminars, computer awareness, technical training, and operational training. Additionally, the parallel run phase is introduced post-installation, allowing users to operate both old and new systems simultaneously until confidence in the new system is established.

11.8 Keywords

1. Parallel Run: Simultaneous operation of old and new systems to ensure stability before full transition.

2. Brainstorming Sessions: Collaborative discussions to generate ideas and increase awareness of system features.

3. Computer-Based Training (CBT): Training delivered through interactive computer programs.

4. System Development Life Cycle (SDLC): Phases involved in the development and implementation of a software system.

5. Technical Support: Assistance provided to users for software-related questions or issues.

6. User-Friendly: Describes software or manuals designed for easy and intuitive use.

7. Online Help System: Interactive assistance accessible within software applications.

11.9 PRACTICE QUESTIONS

11.9.1SHORT ANSWER QUESTIONS

Q1. Explain the concept of classroom training?

Q2. What are the different types of training to be done in an organization?

Q3. Why is the timing of training considered crucial in software implementation?

Q4. What is the significance of a Training Need Analysis?

Q5. Explain the concept of the parallel run in system implementation.

11.9.2 LONG ANSWER QUESTIONS

Q1. Discuss the importance of training in the successful implementation of software systems.

Q2. How does the Training Need Analysis contribute to the effectiveness of training programs?

Q3. Elaborate on the types of training recommended for different categories of individuals in an organization.

Q4. Critically analyze the challenges associated with staff training in the context of new technology adoption.

Q5. How can manufacturers improve the design and effectiveness of user manuals and documentation?

11.10 REFERENCES

Gagne, R. M., Wager, W. W., Golas, K. C., & Keller, J. M. (2005). Principles of instructional design. Cengage Learning.

Allen, M. W. (2006). Michael Allen's guide to e-learning: Building interactive, fun, and effective learning programs for any company. John Wiley & Sons.

Piskurich, G. M., Beckschi, P. K., & Hall, B. (2003). Rapid instructional design: Learning ID fast and right. John Wiley & Sons.

Clark, R. C., & Mayer, R. E. (2016). E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning. John Wiley & Sons.

Horton, W. (2011). E-learning by design. John Wiley & Sons.

Ishikawa, K., Lu, L., Teasley, S. D., & Whittaker, S. (2018). Learning from Data Streams in Collaborative Physical Tasks. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.

Siemens, G., & Baker, R. S. J. d. (2012). Learning analytics and educational data mining: towards communication and collaboration. In Proceedings of the 2nd International Conference on Learning Analytics and Knowledge.

Answer Key (Self-Check Exercise)

I.B, II.B, III.C, IV.B, V.A , VI Remedial , VII Package , VIII. B, IX.A

Semester-IV

Lesson No. 12

SYSTEM ANALYSIS AND DESIGN **AUTHOR: SANJEEV KUMAR MODI**

HARDWARE AND SOFTWARE SELECTIONS

STRUCTURE

12.0 Objectives

12.1 Introduction

12.2 Types of Software

12.3 How to select Hardware and Software

12.4 Major Phases Hardware in Selection

- Self Check Exercise

12.5 How to Select Software

12.6 Method of Software Selection

12.7 The Evaluation Process

- Self Check Exercise

12.8 Summary

12.9 Keywords

12.10 Practice Questions

12.10.1 Short Answer type Questions

12.10.2 Long Answer type Questions

12.11 References

12.12 Answer key

12.0 OBJECTIVE

After reading this chapter, students may be able to understand the crucial aspects of hardware and software selection in system development, exploring types of software, major phases in hardware selection, and methods of software selection.

12.1 INTRODUCTION

A major element in building systems is selecting compatible hardware and software.The systems analyst has to determine what software package is best for the candidate system and, where software is not an issue, the kind of hardware and peripherals needed for the final conversion. Hardware and software selection begins with requirements analysis, followed by a request for proposal and vendor evaluation. The final system selection initiates contract negotiations. It includes purchase price, maintenance agreements, and the amount of updating or enhancements to be available by the vendor over the life of the system. Contract negotiations, seemingly too legal for an analyst, require finesse and strategies designed to get the best deal for the user and protect the user's interests in the acquired system.

12.2TYPES OF SOFTWARE

Software is classified according to whether it performs internal computer functions or allows use of the computer for problem solving. The toner is called systems software-programs designed to control system operations (e.g., operating systems and data base management systems) and system implementation (e.g.; assemblers and compilers). The latter classifications called applications programs) which perform user-oriented functions.

Within this classification we have two groups:

1. Cross-industry applications software, such as accounts receivable and payroll calculations.

2. Industry specific software, such as a safe deposit tracking system, hospital-billing system, or airline reservation system.

In general, software can have one or more of the following attributes:

1. Concurrence of operation. Software allows simultaneous activities to take place; in a computed run. For example, keying in data, on the terminal takes place while the system is reading in data from disk and the central processor is operating on a separate-activity.

2. Resource and information sharing. Different programs share the same hardware resource. Different users may use various programs, or different programs use a centralized database. This implies multiple interfaces of a system with the outside world.

3. Modularity. Software consists of segments connected in various ways, partly due to the separate functions they perform.

4. Multiplexed operation. Some software systems, rotate the use of a resource, such as an input/output device, among different users, while each user have the impression of being the sole user of the system.

12.3 HOW TO SELECT HARDWARE AND SOFTWARE

Today, selecting a system is a serious and time-consuming business. The time spent on the selection process is a function of the applications and whether the system is a basic microcomputer or a mainframe. In either case, planning system selection and acquiring experienced help where necessary payoff in the long run. There are several factors to consider prior to system selection :

Define system capabilities that make sense for business. Computers have proven valuable to business in the following areas:

• Cost reduction includes reduction of inventory, savings on space, and improved ability to predict business trends.

• Cost avoidance includes early detection of problems and ability to expand operations without adding clerical help.

• Improved' service emphasizes quick availability of information to customers, improved accuracy, and fast turnaround.

• Improved profit reflects the 'bottom line" of the business and its ability to keep receivables within reason.

• Specify the magnitude of the problem; that is, clarify whether selection entails a few peripherals or a major decision concerning the mainframe.

• Assess the competence of the in-house staff. This involves determining the expertise needed in areas such as telecommunications and data base design. Acquiring a computer often results in securing temporary help for conversion. Planning for this step is extremely important.

- Consider hardware and software as a package. This approach ensures compatibility. In fact, software should be considered first, because often the user secures the hardware and then wonders what

software is available for it. Remember that software solves problems and hardware drives the software to facilitate solutions.

- Develop a schedule (a time frame) for the selection process. Maintaining a schedule helps keep the project under control.
- Provide user indoctrination. This is crucial, especially for first-time users. Selling the system to the user staff, providing adequate training, and preparing an environment conducive to implementation are pre requisites for system acquisition.

12.4 MAJOR PHASES HARDWARE IN SELECTION

The hardware selection process should be viewed, as a project, and a project, team should be organized with management support. In larger projects the team includes one or more user representatives, an analyst, and ED auditor, and a consultant. Several steps make up the selection process which are as under :

- Requirements analysis.
- System specifications.
- Request for proposal (RFP).
- Evaluation and validation.
- Vendor selection.
- Post -installation review.

12.4.1 Requirements Analysis

The first step in hardware selection is to understands the user's requirements within the framework of the organization's objectives and the environment in which the system is being installed. Consideration is given to the users resources as well as to finances.

In selecting software, the user must decide whether to develop it in-house, hire a service company or a contract programmer to create it, or simply acquire it from al software house. The choice is logically, made after the user has clearly define the requirements expected of the software. Therefore, requirements analysis sets the tone for software selection.

12.4.2 System Specifications

Failure to specify system requirements before the final selection almost always results in a faulty acquisition. The specifications should delineate the user's requirements and allow room for bids from various vendors. They must reflect the actual applications to be handled by the system and include system objectives, flowcharts, and input-output requirements, file structure, and cost. The specifications must also describe each aspect of the system clearly, consistently, and completely.

12.4.3 Request for Proposal (RFP)

After the requirements analysis and system specifications have been determined, a request for proposal (RFP) is drafted and sent to selected vendors for bidding. Bids submitted are based on discussions with vendors. At a minimum, the RFP should include the following:

1. Complete statement of the system specifications, programming language, price range, terms, and time frame.

2. Request for vendor's responsibilities for conversion, training, and maintenance,

3. Warranties and terms of license or contractual limitations.

4. Request for financial statement of vendor

5. Size of staff available for system support.

12.4.4 Evaluation and Validation

The evaluation phase ranks vendor proposals and determines the one best suited to the user's needs. It looks into items such as price availability, and technical support. System validation ensures that the vendor can, in fact, match his/her claims, especially system performance.

Role of the Consultant. For a small firm, an analysis of competitive bids can be confusing. For this, reason, the user may wish to contract an outside consultant to do the job. Consultants provide expertise and an objective opinion. The past decade has seen the growth of internal management consultant teams in large organizations, as opposed to external consulting teams.

12.4.5 Vendor Selection

This step determines, the vendor with the best combination of reputation, reliability, service record, training, delivery time, lease/ finance terms, and conversion schedule.

Initially, a decision is made on which vendor to contact. The sources available to check on vendors include the following

- Users
- Software houses
- Trade associations
- Universities
- Publications/ Journals.
- Vendor software lists.
- Vendor referral directories.
- Published directories.
- Consultants.
- Industry contacts.

12.4.6 Post-Installation Review

Sometime after the package is installed, a system evaluation is made to determine how closely the new system conforms to plan. System specifications and user requirements are audited to pinpoint and correct any differences.

- **SELF-CHECK EXERCISE**

I. What is the purpose of the request for proposal (RFP) in the hardware selection process?
a. Vendor selection
b. System validation
c. Requirements analysis
d. Bid solicitation

II. What aspect does system validation primarily address?
a. Vendor reputation
b. System performance
c. Financial statements
d. Programming languages

III. What is a characteristic of cross-industry applications software?
a. Industry-specific functions
b. User-oriented functions
c. Concurrence of operation
d. Modularity

IV. In hardware selection, what is the first step in understanding the user's requirements?
a. System specifications
b. Request for proposal (RFP)
c. Evaluation and validation

d. Requirements analysis

12.5 HOW TO SELECT SOFTWARE

Software selection is a critical aspect of system development. There are two ways of acquiring software: custom-made or "off-the-shelf packages.

1. A good package can get the system running in a matter of days rather than the weeks or months required for "home-grown" packages.

2. MIS personnel are released for other projects.

3. Packages are generally reliable and perform according to stated documentation.

4. Minimum risks are usually associated with large-scale systems and programming efforts.

5. Delays in completing software projects in house often occur because programmers quit in midstream.

6. It is difficult to predict the cost of "home-grown" software.

7. The user has a chance of seeing how well the package performs before purchasing it.

There are drawbacks, however, to software packages :

1. The package may not meet user requirements adequately.

2. Extensive modification of a package usually results in loss of the vendor's support.

3. The methodology for package evaluation and selection is often poorly defined.

4. The result is a haphazard review based on a faulty process or questionable selection criteria.

5. For first-time software package users, the overall expectation from a package is often unclear and ill defined.

It can be seen, then, that the quality of a software package cannot be determined by price alone. A systematic review is crucial.

12.6 METHOD OF SOFTWARE SELECTION

Prior to selecting the software the project team must set up criteria for selection.

Selection criteria fall into the categories described here.

- **Reliability:** Hardware may become inoperative because of design errors,manufacturing errors ordeterioration caused by heat, humidity, friction, and the like. In contrast, software does not fail or wear out. Any reliability problems are attributable to error s introduced during the production process. Furthermore, whereas hardware failure is basedlargely on random failure, software reliability is based on predestined errors. Although reliable software is a desirable goal, limited progress has been made toward improving it in the last decade. The fact of unreliable software had led to the practice of securing maintenance agreements after the package is in operation. In a sense, unreliability is rewarded. Software reliability brings up the concept of modularity, or the ease with which a package can be modified. This depends on whether the package was originally designed as a package or was retrofitted after its original development for single installation use. A package with a, high degree of modularity has the capacity to operate in many machine configurations and perhaps across manufacturers' product lines The following questions. should be considered :

1. Is there room for expanding the master file?

2. How easily can additional fields, records, and files be added?

3. How much of the system becomes unusable when a part of it fails?

4. Are there errors a user can make that will bring down the system?

5. What are the recovery capabilities?

- **Functionality.** It is a definition of the facilities, performance, and other factors that the user requires in the finished product. All such information comes from the user.
- **Capacity.** Capacity refers to the capability of the software package to handle the user's requirements for size of files, number of data elements, volume of transactions and reports, and number of occurrences of data elements.
- **Flexibility.** It is a measure of the effort required to modify an operational program.One feature of flexibility is adaptability, which is a measure of the ease of extending the product.
- **Usability.** This criterion refers to the effort required to operate, prepare the input, and interpret the output of a program.
- **Security.** It is a measure of the likelihood that a system's user can accidentally or intentionally access or destroy unauthorized data.

- **Performance.** It is a measure of the capacity of the software package to do what it is expected to do. This criterion focuses on throughput, or how effectively a package performsunder peak loads. Each package should be evaluated for acceptance on the user's system.The language in which a package is written and the operating system are additional performance considerations.

- **Serviceability**. This criterion focuses on documentation and vendor support.Complete documentation is critical for software enhancement. It includes a narrative description of the system, system logic and logic flowcharts, input-output and file descriptions and layouts and operator instructions. Vendor support assures the user adequate technical support for software installation, enhancements, and maintenance.
- **Minimal Costs**. Cost is a major consideration in deciding between in-house and vendor software. Cost -conscious users consider the following points:

1. Development and conversion costs.

2. Delivery schedule.

3. Cost and frequency of software modifications.

4. Usable life span of the package.

12.7 THE EVALUATION PROCESS

Sources for Evaluation

Three sources of information are used in evaluating hardware and software :

(1) benchmark programs

(2) experience of other users and

(3) product reference manuals.

Benchmark Programs: A benchmark is a sample program used for evaluating different computers and their software. This is necessary because computers often will not use the same instructions, words of memory, or machine cycle to solve a problem. In the context of software selection, benchmarking may include the following:

1. A determination of the minimum hardware configuration needed to operate a package. A package may be extremely sensitive to its hardware or have considerable flexibility.

2An acceptance test as specified in the contract.

3. Testing in an ideal environment to determine comparative timings and in a normal environment to determine its influence on other programs.

The more elaborate the benchmarking, the more costly is the evaluation. The user's goals must be kept in perspective. Time constraints also limit how thorough the testing process can be. There must be a compromise on how much to test while still ensuring that the software (or hardware) meets its functional criteria. Since benchmarks only validate the vendor's claims, other sources of information are necessary. The experience of other users with the same system, software, or service is important. An important step in the evaluation process is to read product reference manuals that evaluate system capabilities. Since such a search is often laborious, one alternative is to contact organizations that publish reports based on ongoing research and system testing in various sites. For example, Auerbach, Inc.(Philadelphia), publishes loose-leaf referencesoninformation processing,telecommunications, and computer graphics. The reports elaborate on computer products, service and prices. Considering the benefits offered, the cost is relatively low.

- **SELF CHECK EXERCISE**

V. Software reliability is based on predestined errors, and _____ often involves securing maintenance agreements after the package is in operation.

VI. _____ refers to the capability of the software package to handle the user's requirements for various aspects such as file size and transaction volume.

VII. The evaluation process involves considering information from _____, user experience, and product reference manuals.

12.8 SUMMARY

This section emphasizes the significance of selecting compatible hardware and software in system development. It delves into types of software, such as systems software and applications programs, and provides insights into the major phases of hardware selection. The process involves requirements analysis, system specifications, request for proposal (RFP), evaluation, vendor selection, and post-installation review. The method of software selection, including custom-made and off-the-shelf packages, is discussed with considerations for reliability, functionality, capacity, flexibility, usability, security, performance, and serviceability.

12.9 KEYWORDS

1.Request for Proposal (RFP): A document used in procurement processes to solicit bids from potential vendors for a project, outlining requirements and evaluation criteria.

2. Modularity: The degree to which a system or software is divided into components or modules, promoting ease of maintenance, flexibility, and understanding.

3. Reliability: The ability of a system or software to consistently perform its intended functions without failures or errors.

4. Security: The protection of computer systems, networks, and data from unauthorized access, attacks, and damage.

5. Serviceability: The documentation and vendor support provided for a software package, ensuring adequate technical support for installation, enhancements, and maintenance.

6. Custom-made Software: Software that is specifically developed for a particular user or organization, tailored to their unique requirements.

7. Off-the-shelf Software: Pre-packaged software applications available for purchase and use by a wide range of users without significant customization.

12.10 PRACTICE QUESTIONS

12.10.1 SHORT ANSWER QUESTIONS

Q1. Define systems software and applications programs.

Q2. What are the key factors to consider prior to system selection?

Q3. Explain the importance of requirements analysis in hardware selection.

Q4. What are the sources of information used in evaluating hardware and software?

12.10.2 LONG ANSWER QUESTIONS

Q1. Discuss the types of software and their classification.

Q2. Explain the major phases involved in hardware selection.

Q3. What are the key considerations in defining system capabilities for business?

Q4. Elaborate on the role of post-installation review in system development.

Q5. How does benchmarking contribute to the evaluation process in system development?

12.11 REFERENCES

Laudon, K. C., & Laudon, J. P. (2018). "Management Information Systems: Managing the Digital Firm" (15th ed.). Pearson.

Dennis, A., Wixom, B. H., & Tegarden, D. (2015). "Systems Analysis and Design: An Object-Oriented Approach with UML" (5th ed.). Wiley.

Tanenbaum, A. S. (2018). "Structured Computer Organization" (7th ed.). Pearson.

Ramakrishnan, R., & Gehrke, J. (2002). "Database Management Systems" (3rd ed.). McGraw-Hill.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). "Introduction to Algorithms" (3rd ed.). MIT Press.

Schwalbe, K. (2018). "Information Technology Project Management" (8th ed.). Cengage Learning.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). "Design Patterns: Elements of Reusable Object-Oriented Software." Addison-Wesley.

Patterson, D. A., & Hennessy, J. L. (2017). "Computer Organization and Design: The Hardware/Software Interface" (5th ed.). Morgan Kaufmann.

12.12 ANSWER KEY (SELF CHECK EXERCISE)

I.D, II.B, III.A, IV.D, V.Unreliability, VI. Capacity , VII.Benchmarkprograms

PROTOTYPING

STRUCTURE

13.0 Objectives

13.1 Introduction to Prototype

13.2 Definition of Prototype Model

13.3 Stages of Prototyping Model

- Self-check Exercise

13.4 Different kinds of Prototyping models

13.5 Applications best suited to rapid prototyping

13.6 Benefits of rapid prototyping

13.7 Limitations of Prototyping

- Self-check Exercise

13.8 Summary

13.9 Keywords

13.10 Practice Questions

13.10.1 Short Answer type Questions

13.10.2 Long Answer type Questions

13.11 References

13.12 Answer key

13.0 OBJECTIVES

After reading this chapter, students will be able to

Understand the concept of the Prototyping Model in system development life cycle (SDLC).

Recognize the key stages and principles involved in Prototyping.

13.1 INTRODUCTION

So far we have learnt various models of system development life cycle (SDLC), their role, advantages, and disadvantages. Now in this lecture we will be going to study the last and one of the important models, which helps in analyzing and developing the system. The name of this model is -Prototyping Model. This model involves the user more directly in the analysis and design experience that does the system development life cycle (SDLC) or the structured analysis method. Prototyping is very effective under the correct circumstances, as we point out. However, like the other methods we discuss in previous lectures, it is useful only if it is employed at the right time and in the appropriate manner. Prototyping has a reputation for being difficult to manage. The keys to managing a prototyping effort are becoming clear, however; they include knowing what you want to learn from the prototype, access to rapid prototyping techniques, and end-user involvement in development of the prototype.

13.2 WHAT DO YOU UNDERSTAND BY PROTOTYPE?

A Prototype is any working system-not just an idea on paper-that is developed to test ideas and assumptions about the new system. Like any computer-based system, it consists of working software that accepts input, performs calculations, produces printed or displayed information, or performs other meaningful activities. It is the first version or iteration of an information system-an original model.

Why we should use systems prototyping?

Information requirements are not always well defined. Users may know only those certain business areas need improvement or that existing procedures must be changed or they may know that they need better information for managing certain activities but are not sure what that information is. The user's requirements might be too vague to even begin formulating a design. In other cases, a well-managed systems investigation may produce a comprehensive set of systems requirements, but building a system that will meet those requirements may require development of new technology.

The prototype is actually a pilot or test model; the design evolves through use. If use of the sales prototype system reveals that entering customers' names and addresses through a portable terminal creates too many errors, designers may modify the system so that only customers' names are necessary; the addresses can be automatically retrieved from files stored in the system.

Although the prototype is a working system, it is designed to be easily changed. Information gained through its use is applied to a modified design that may again be used as a prototype to reveal still more valuable design information. The process is repeated as many times as necessary to reveal essential design requirements.

In general, systems analysts find prototypes to be most useful under the following conditions :

- No by The system with the characteristics of the one proposed has yet been constructed the developers
- The essential features of the system are only partially known; others are not identifiable even through careful analysis of requirements.
- Experience in using the system will significantly add to the list of requirements system should meet.
- Alternate versions of the system will evolve through experience and additional development and refinement of its features.

The system user(s) will participate in the development process.

Prototyping Model was developed on the assumption that it is often difficult to know all of data, nor your requirements at the beginning of a project. Typically, users know many of the objectives that they wish to address with a system, but they do not know all the nuances of the do they know the details of the system features and capabilities. The Prototyping Model allows for these conditions, and offers a development approach that yields results without first requiring all information up-front.

Underlying principle of prototyping is :

Users can point to features they like or dislike and so indicate shortcomings in an existing and working system more easily than they can describe them in a theoretical or proposed system. Experience and use produce more meaningful comment than analysis of charts and narrative proposals.

The Prototyping process includes:

1. Requirement Gathering
   Customer and developer meet to define the overall objective of software.
2. Quick Design
   Quick Implementation and representation of these aspects agreed between the customer and developer of this software will be made available to the user.
3. Build prototype
   Leads to the design of a prototype of the project to serve as a mechanism for identifying software requirements.
4. Evaluate and Refine Requirements

Analysis of the prototype and further refinement.

5. Engineer Product.
   A requirement of the project is meet.

13.3 STAGES OF PROTOTYPING MODEL

13.3.1 Analyze Requirements

This stage involves the developer understanding the content and nature of the customer's initial requirements. During this analysis, the developer should determine thefunctionality to be represented in the prototype. This functionality should focus upon the requirements that are unclear or fuzzy. It is undesirable and costly to prototype requirements that are fully understood. Also, the choice of process model and prototyping approach should be determined by the fuzzy nature of the requirements, and hence an approach should be taken which permits the prototyping of such requirements. Therefore, the prototyping approach chosen for this particular task was Throwaway Prototyping.

This approach was selected due to the nature of the requirements - some were known, some were unclear, and there might have been a possibility that other requirements may come to light during the prototyping process. Also, throwaway prototyping supports the rapid development of a trial version of the functional behaviour for demonstration purposes. This form of prototyping can be contrasted against both the other forms of prototyping (i.e. evolutionary or operational) and the development of the production quality system. The objective of the prototype is to secure accurate requirements specification through customer " validation, and that the prototype will be discarded after use. Therefore, the implementation need not be maintainable or be resource efficient.

It the case is important to inform the customer about the prototype and its characteristics. In of "Spell Check" these were that the prototype would only reflect the intended behaviour of the system and not possess other features or performance characteristics of the final the version. Also, the user interface would be limited and would probably not resemble interface of the production quality system.

In this particular case it has been decided to prototype all the functionality of the spell. checking sub-system, and in particular concentrate upon the skipping of words. Therefore the developer analyzed the system's requirements, resulting in the following list of requirements to be prototyped:

1. The word processing system should possess a means of checking the spelling of inputted text and reporting incorrectly spelled words.

2. The system should provide a means of changing the incorrectly spelled words, against which inputted text will be compared.

3. The dictionary should be expandable, i.e. there should be a facility for users to add new words.

4. The system should posses a dictionary of correctly spelled words

5. During the spell checking process, a user should be able to ignore the spelling of any word, irrespective of the word being spelled correctly or not.

These requirements would form the Preliminary Requirements. Also at this stage, any assumptionsabout the requirements should be recorded. These assumptions can also be presented to the customer during prototype evaluation.

PROTOTYPE DESIGN

The Implementation Approach

To construct the prototype, a suitable implementation approach must be used. This approach must offer features, which satisfy the general requirements of prototyping, such as rapidity and ease of modification. Obviously, an approach, which relies upon a complex and lengthy development cycle, is unsuitable. It can be argued that Microsoft's Visual Basic possesses many features, which a developer can use to facilitate an effective prototyping process. The features are :

1. An interactive/visual user interface design tool.

2. Easy connection of user interface components to underlying functional behaviour.

3. Easy to learn and use implementation language (i.e. Basic).

4. Modifications to the resulting software are easy to perform.

As well this, Visual Basic is appropriately suited to this form of rapid development because the software is executed on an interpretative basis, and does not rely upon lengthy compilation processes. Although this maybe considered a limiting factor when developing production quality systems, whose speed and response times are crucial, prototyping does not require this. For these reasons, Microsoft's Visual Basic has been chosen as the implementation approach.

The Design of "Spell Check"

The "Spell Check" software was designed in accordance with general prototyping principles, and in particular, rapid construction. Its overall structure was kept simple, while accommodating the required behaviour as described by the initial requirements. Usually, within the development of production quality systems, a simple and un-optimized design may lead to low performance. However, during the use of a throwaway prototype, performance is of not not a critical factor, and therefore the efficiency of the design is not a priority. Also, since the prototype will be discarded after use, the maintainability or elegance the design is not an important consideration. Large amounts of documentation are also required. By removing these particular aspects, the processes of design and implementation can be performed rapidly.

In order to produce a running system as quickly as possible, a number of decisions were taken regarding the amount of functionality provided. For instance, it was decided not to incorporate any text editing facilities, since this functionality is provided by many other existing tools. Instead, the prototype was designed to simply 'read' an existing text file created by an external editor. Indeed, much development effort was saved by this decision.

13.3.2 Prototype Construction

This section describes the construction of the "Spell Check" prototype. This stage involves the actual coding of the prototype. The construction of a prototype using Visual Basic can be described in terms of three tasks. These tasks were performed during the construction of the "Spell Check" software. The following sections demonstrate these tasks, giving specific examples where appropriate. The tasks are:

User Interface Construction: It is desirable that any prototype construction activities be performed as quickly as possible. User interface construction using Visual Basic is indeed rapid, and has been simplified by the use of the ‘drag-and-drop’ style of interactive editing. User interface components, such as buttons and text-boxes, are placed upon a Visual Basic form and are then connected to appropriate functionality.

Implementing Functional Behaviour: A prototype represents a model of the behaviour of a proposed system. This behaviour is described by a corresponding implementation. This section describes the issues surrounding implementing functional behaviour using Visual Basic.

Above all, the most important principle is that this section of the prototype should be developed as quickly as possible

Remember, that the software is being used only for validation purposes, i.e. experimentation, learning, and evaluation, and will not be used after the prototyping process has completed. Therefore, the elegance of the code, its efficiency, and error handling, as well as other aspects such as reusability and maintainability, are not vitally important.

To this end, Visual Basic provides an appropriate language and development tools to support the development of software of this nature. This is due to the underlying language, Basic, being easy to learn and understand, and reasonably easy to read and use.

As with any development project, the implementation must be driven by the requirements of the system. The requirements for the "Spell Check" prototype are given in the Initial Requirements section. It is essential that the implementation reflect these requirements as closely as possible, so effective validation can take place. The implementation must also take into account the limited user interface characteristics of the prototype, and any constraints, which it imposes.

Developing the Data Model: In terms of data, a prototype is no different from any other piece of software, in that it requires information in some form to process. This information is usually stored in memory, or on disc-based files, as data structures. This section describes the main data structures used by the "Spell Check" prototype.

When developing any data model, the types of data structures available in the development environment have to be taken into account, along with the needs of the application itself, and what kind of data needs to be stored, and the volatility/persistence of the has data in question (i.e. whether the data should remain in existence when the program ended).

Microsoft's Visual Basic provides an adequate set of data types, namely integer, floating point, string, boolean, and array. These are the building blocks from which more complex data types are constructed.

So, what kind of data does the "Spell Check" prototype have to represent? This question can be answered by examining the requirements of the system.

1. The need to store words which have to be checked - i.e. the input file. Since the decision was taken to store the text to spell check in separate files (which can be created by any text editor), then this file must be read into memory, and its contents stored ready for processing. To achieve this, another question must be answered - what is the nature of these text files? Simply, the answer is a sequence of characters, one after another. Also, what type of access mechanism is required so these characters can be used effectively?
   Due to these considerations, the decision to represent the text in memory as an array. This structure is sequential in nature, but provides easy access to individual characters at any point within the sequence. the implementation, this array is defined in the 'declarations' section of the form's 'general' section. The variable name "File$()"

2.place to store words which have been checked. When words have been checked, they have to be stored either in memory, or in a file. The decision was taken to store the checked words in a disc file. The reason for this was that once checked, the words were no longer needed, and that a file can easily be 'built-up' (i.e. words added to the end of it) until all the words have been checked.The file is called "done.txt", and is stored in the same directory, which houses the "Spell Check" prototype. The file can be examined using any text editor.

3. The dictionary. The dictionary represents all the correctly spelled words known to the system, and it is desirable for such data to remain when the prototype has terminated. Therefore, it was decided to store the dictionary in a disc-based file. This structure also allows new words to be added easily. The dictionary file is called "dict.dic", and is stored in the same directory, which houses the "Spell Check" prototype. The file can be examined using any text editor.

Further to this, the dictionary will have to be read into memory, and to this end, an array structure is used. Within the implementation, the array is defined in the 'declarations' section of the form's 'general' section. The variable name is "Dict$()" Again, it must be remembered that efficiency of computation and storage are not of paramount importance to prototype use. Therefore, more optimal data storage techniques need not be considered until the final design for the production quality system is being developed.

- **SELF CHECK EXERCISE**

  I. Which stage of the prototyping process involves refining the requirements based on prototype analysis?
  a. Quick Design
  b. Build Prototype

c. Evaluate and Refine Requirements
d. Engineer Product

II. Which implementation approach was chosen for the "Spell Check" prototype?
a. Throwaway Prototyping
b. Evolutionary Prototyping
c. Operational Prototyping
d. Spiral Prototyping

III. Where are the checked words stored in the "Spell Check" prototype?
a. In memory
b. In a disc file called "done.txt"
c. In the dictionary file
d. In the same directory as the prototype

13.4 DIFFERENT KINDS OF PROTOTYPING MODELS

13.4.1 Rapid Throwaway Prototyping .

It is used to analyze the most risk items that the developer has not much experience about that problem. In this model, quick and dirty prototypes are built, verified with customers, and thrown away until a satisfactory prototype is reached, at which time, full- scale development begins. The following stages works like the traditional the corresponding part of waterfall model.

13.4.2 Quick and Dirty Prototyping

Quick and dirty is term describing the approach of quickly bring up a version of a system, then modifying it until the customer can grant minimal approval. The programs are modified based on the feedback from the user as they see the output. Unlike rapid throwaway prototyping, quick and dirty prototyping might be kept to modify. The software is almost sure to be expensive. This approach is rarely accompanied with documents.

13.4.3 Detail Design-Driven Prototyping

The model is complete and as perfect as it can be made to be. It is then test-driven to uncover any defects, which can be corrected before delivery of the final product. The sort of prototyping is problematic because it is so detailed - analysis and design documents being allegedly complete-that it will be expensive to change all the documents and the system when prototype demonstrates uncover unexpected or undesirable results, having the similar shortcomings of waterfall models. Actually, this kind of prototyping is much like a traditional waterfall process model instead of a prototyping model.

13.4.4 Evolutionary Prototyping

In the evolutionary prototyping approach, a prototype is built based on some known requirements and understanding. Contrast to rapid throwaway prototyping, this model is then refined and evolved. Whereas throwaway prototypes are usually used because of poorly understanding of the new system, evolutionary prototypes are more likely to be used toevolve based on the best understanding of the system and thus build on the development team's strength.

13.5 APPLICATIONS BEST SUITED TO RAPID PROTOTYPING

13.5.1 Small versus large application system

In the early days of rapid prototyping, the tools for prototyping had very poor performance characteristics, making them unsuitable as implementation vehicles for system that featured large numbers of concurrent users or large amounts of stored data. Thus evolutionary prototyping was only applied to small application systems. With the introduction of prototyping tools with much better performance characteristics, and the additional features which make possible combination of prototype modules and modules

produced using standard software development techniques within the same system, a prototype of a large system may be evolved into the final deliverable system. One of the problems with discussing small versus large application systems concerns how one measures system size. As a general rule, systems that have evolved from prototypes will have fewer lines of code and small amounts of stored data than traditionally developed systems. One-line of 4+GL may replace 40 or more statements in a third generation procedure language. A visually programmed module may contain no visible lines of code at all. Evolved prototypes will also be easier for maintenance staff to understand and modify throughout the useful life of the system. Prototyped systems tend to be smaller and simpler, even though they often solve large problems.

13.5.2 Real-time versus Data processing systems

The critical issue for real-time system is often response time. The tools available today for prototyping offer a means for improving the response time of prototype modules to a level equal to that of traditional developed software. The modularity of an evolutionary prototype- combined with an integrated environment with the interfaces to other types of development environments-is what provides an evolutionary prototype with the ultimate in flexibility. Example of prototyping User Interface Systems versus Embedded systems

13.6 BENEFITS OF RAPID PROTOTYPING

13.6.1 Quality Assurance and control

Rapid prototyping could be considered a quality assurance technique, since its primary benefit and ultimate objective is to greatly meet the requirements of the user by the functionality of delivered software. Quality control is built into a rapid prototyping project since users are involved in early prototype iterations during the analysis phase and some prototyping models can even response user changing requirements at other development stages. Therefore, the final product will ultimately and greatly assurance that it can meet user requirements.

13.6.2 Accommodating new or unexpected user requirements

Users are not always certain what they want requested software systems to do. Users are often accused of unfairly changing their minds about the desired functionality of a system under development during the test phase of a project. Sometimes reality is far different from conceptualizations based on paper specifications

Rapid prototyping provides an opportunity to accommodate these changes, which basically represent creative thinking on the part of the user, early in the life cycle and at a very low cost. The result is a dramatic decrease in disruptive and costly change activity during testing and a significant reduction of costly post implementation perfect maintenance. When users are encouraged to take an active role in the development of a system, that system stands a much better change to become a source of pride for them.

Needless to say, such a psychological phenomenon is positive for system quality and prototyping is valuable in

13.6.3 Cost saving in development and maintenance

Evolutionary rapid prototyping provides two powerful new ways to help beat down the high costs of software development and software maintenance. The first one is that systems will be easier to modify in place, since the product will be developed based on previous stages which has been proven to meet user's previous requirements. The second one in which prototyping reduces maintenance costs is that need fewer modification after delivery.

Since user's requirements can be reflected in the product during development, the maintenance cost after delivery will be greatly reduced.

13.6.4 User satisfaction with software systems

With prototyping, a "what you see is what you get" approach eliminates the bulk of communication problems, allow delivery of systems as least as quickly as ever, and permits the correction of errors due to bad specs in the early, inexpensive phases. When a wrong specification is designed and implemented that

does not meet user requirements, even the most modern tools and technology cannot salvage the product. The more functions the system has, the more it will cost to correct the systems structure after delivery, therefore, the worse the user satisfaction is.

When users are allowed into the analysis or even into the development, they can "see" first-hand problems, errors or any misunderstanding at the early stages of product lifetime.

Such is reassuring to a user who can observe the action. This is why "rapid" is such an important part of rapid prototyping.

13.7 LIMITATION OF AND OBJECTIONS TO RAPID PROTOTYPING

First, prototyping requires the user to participate into the project as long as possible.

That is the key point of prototyping. Ideally, user can follow all the development stages.

However, for most projects, users are not always available. The power of the prototyping is to communicate with the customer. It is imperative that a project plan is started as the first phase of activity, and then the plan is continually refined to meet the customer changing requirements. When the number of deliverables or functions of the proposed system increases or becomes more complicated, so does the schedule lengthen, raising the cost?

Second, the iterative process of prototype might continue forever if it is not managed and monitored properly, since management is often not aware of the status of prototyping.

Therefore, a successful prototyping needs efficient and effective management.

Third, there are very few prototypes and even fewer trained maintainers of prototyped systems.

In addition, there is little training available to help program get the main idea of this new way of thinking about system development.

A successful example of prototyping models

(Cusumano 1997) shows a successful example of prototyping model. It tells us the process Microsoft uses to build software. Although the company does not use a pure prototyping process model, the main features of its process are the characteristics of rapid prototyping.

At the requirement stage, Microsoft Company does not use waterfall model, which would produce a large amount of document. Instead, it begins by creating a "vision statement" defining the goals for a new product and orders the user activities that need to be supported by the product features. The key point of this model is to grasp user requirements as much as possible and as soon as possible.

It is easy and cost saving to modify the "vision statement" instead of a large amount of document. After figuring out the "vision statement", product manager consult the program manager and the program manager consults the programmers to outline the main features of the product. Unlike Waterfall models which try to cover all the details of the product, the specifications in Microsoft leaves much flexibility and freedom to the programmer to accomplish. This is the main characteristic of prototyping model.

In implementation, the project manager then divides the project into parts and divides the project Schedules into several milestone junctures (sequential subprojects), representing for major portions of product, according to the priorities of the features. With regard to each of the subprojects, Microsoft uses a combination of multiple process models to achieve the greatest benefits.

It uses Waterfall models to develop the sub product,

integrate the product, test the product and fix errors. In addition, it uses a special technique called "daily synchronization" to synchronize all changes. Actually the idea comes directly from software configuration management (SCM), which is what SEl advocates.

Another aspect of the Microsoft model is that it uses an iteration process to evolve from one milestone to another. Each milestone is based on the previous one. The more the product evolves, the more closely the

product meets user requirements. Once user changes requirements in a iteration cycle, the next iteration cycle will reflect those changes.

In general, the essentials of Microsoft process model are that it uses rapid prototyping process to outline the main features of the new product, prioritize the activities and divide the project into several parts. Each iteration is built and stabilized and evolve to the next iteration (next milestone) like a spiral model. In addition, Microsoft uses user feedback as an important mechanism to improve development.

- **SELF CHECK EXERCISE**

IV**.** What is a benefit of Rapid Prototyping in terms of user satisfaction?
a. Reduced development speed
b. Early correction of errors
c. Lack of communication with users
d. Limited user involvement

V. What application systems are more likely to benefit from Rapid Prototyping?
a. Large application systems
b. Systems with a high number of concurrent users
c. Small application systems
d. Systems with detailed documentation

VI.Rapid prototyping could be considered a _____ technique, aiming to meet the requirements of the user by the functionality of delivered software.

VII. In the evolutionary prototyping approach, a prototype is built based on some known requirements and understanding, and then it is refined and _____.

13.8 SUMMARY

The Prototyping Model is a crucial system development approach that directly involves users in the analysis and design process. Unlike traditional SDLC methods, prototyping allows for the creation of a working system, not just a conceptual idea. Prototypes are essential for testing ideas and assumptions about the new system, and the model evolves through user feedback and rapid modifications. The stages include requirement gathering, quick design, prototype construction, evaluation, refinement, and engineering the final product.

13.9 KEYWORDS

1. Iterative Process: A process that involves repeating and refining a sequence of steps until a desired result is achieved.

2. Throwaway Prototyping: A prototyping approach where the prototype is discarded after use, focusing on validation and learning.

3. User Interface: The point of interaction between the user and the software.

4. Quality Assurance: Processes to ensure that the software meets specified requirements and standards.

5. Quick and Dirty Prototyping: An approach to rapidly create a system and modify it until customer approval is obtained.

13.10 PRACTICE QUESTIONS

13.10.1 Short Answer type Questions

Q1. Under what conditions do systems analysts find prototypes most useful?

Q2. What is the underlying principle of prototyping?

Q3. How does Throwaway Prototyping differ from Evolutionary Prototyping?

Q4. What are the benefits of Rapid Prototyping?

13.10.2 Long Answer type Questions

Q1. Explain the role of prototypes in analyzing and developing a system.

Q2. Discuss the stages of the Prototyping Model and their significance.

Q3. Compare and contrast Rapid Throwaway Prototyping and Quick and Dirty Prototyping.

Q4. How does Prototyping contribute to quality assurance and control in software development?

Q5. Provide an example of a successful implementation of the Prototyping Model.

13.11 REFERENCES

Sommerville, I. (2011). Software Engineering (9th ed.). Addison-Wesley.

Pressman, R. S. (2014). Software Engineering: A Practitioner's Approach (8th ed.). McGraw-Hill.

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. ACM SIGSOFT Software Engineering Notes, 11(4), 14-24.

13.12 ANSWER KEY (SELF-CHECK EXERCISE)

I.C ,II.A, III.B,IV. B, V.C,VI. Quality Assurance, VII. Evolved

Semester-IV

Lesson No. 14

SYSTEM ANALYSIS AND DESIGNAUTHOR: ANKIT JINDAL

TESTING AND QUALITY ASSURANCE

STRUCTURE

14.0 Objective

14.1Introduction

14.2Definition

14.3Types of Tests

- Self-Check Exercise

14.4Levels of Testing

- Self-Check Exercise

14.5Introduction to Quality Assurance

14.6Quality Factors Specifications

14.7Levels of Quality Assurance

- Self-Check Exercise

14.8Summary

14.9 Keywords

14.10 Practice Questions

14.10.1 Short Answer Type Questions

14.10.2 Long Answer Type Questions

14.11 References

14.12 Answer key

14.0 OBJECTIVES

After reading this chapter , students will be able to :

Explore the philosophy behind software testing, focusing on finding errors through test cases and the two general testing strategies: code testing and specifications testing and gain insights into the various levels of testing, including unit testing and system testing, and the importance of quality factors specifications in the software development life cycle.

14.1 INTRODUCTION

Dear students it should be clear in mind that the philosophy behind testing is to find errors. Test cases are devised with this purpose in mind. A test case is a set of data that the system will process as normal input. However, the data with the express intent of determining whether the system will process then correctly. For example, test cases for inventory handling should include situation in which the quantities to be withdrawn from inventory exceed, equal, and are less than the actual quantities on hand. Each test case is designed with the intent of finding errors in the way the system will process it. There are two

general strategies for testing software: code testing and specifications testing. In code testing,the analyst develops that case to execute every instructions and path in a program.

Under specification testing, the analyst examines the program specifications and then writes test data to determine how the program operates under specific conditions. Regardless of which strategy the analyst follows, there are preferred practices to ensure that the testing is useful. The levels of tests and types of test data, combined with testing libraries are important aspects of the actual test process.

14.2 DEFINITION

Testing is not a technique for building quality systems; rather, it is a technique that is used because we recognize that we have failed to build a fault free system. Testing assists in its repair by identifying where the program fails to meet its specification. I will broaden the usual definition of testing that typically refers only to the testing of code. The expanded definition includes the testing of the many types of models, requirements, analysis, architectural and detailed design, that are constructed in the early development phases.

These models may be represented in an executable format in a CASE tool such as BetterStateA or they may require a form of symbolic execution or inspection. My justification for classifying these activities as testing is that the inputs to one of these executions will be much more specific than the inputs to a typical review process (more about this below).

Using this expanded definition, there are a number of products of the software development process that should be tested including:

- Requirements models
- Analysis and design models
- Architectural models
- Individual components
- Integrated system code.

In fact, there should be a testing activity associated with each step in the development process. Adding the testing of models will find some faults earlier and process resulting in cheaper repair costs.

Although these products most often will be tested to determine their correctness, testing may be used to investigate a variety of attributes including:

- Performance
- Usability
- Robustness
- Reusability
- Extensibility
- Flexibility.

The testing perspective is an attitude that questions the validity of every product and utilizes a thorough search of the product to identify faults. Systematic algorithms and intuitive insights guide this search. It is this systematic search that makes testing a more powerful tool than reviews and inspections. A review will almost never find something that isn't there. That is, a review typically only seeks to validate what is in the model and does not systematically search to determine if all the entities and relationships that should be there are. The testing perspective requires that a piece of software demonstrate that it is performing as its complete specification indicates that it should (and that there is no extra behavior). A product is tested to determine that it will do what it is supposed to do (a positive test). It should also be tested to ensure that it does not do what it is not supposed to do a negative test). This leads to the need to define the parameters of the search.

Test cases are created as part of the testing process to direct the search. A test case presents a context in which the software is to operate and a description of the behavior expected from the software within that context. In most traditional testing situations the context is an operational one in which the software is

executed to determine its actual behavior. If the product under test is a model rather than actual code, the test case may contain a textual description of the context as well as some specific input values.

The testing perspective may be adopted by the same person who developed the product. under test or by another person who brings anindependent view of the specification andthe product. Developer-led testing is efficient since there is no learning curve on the product under test, but it can be less effective because the developer may not search as exhaustively as the independent tester. The completeness and clarity of the specification becomes very important when a second person becomes involved in the testing. If a tester is not given a specification, then any test result is correct!

WHAT IS A COMPONENT?

What do I mean by component when I say component testing? The short answer is, it is anything you want it to be! The techniques that I will illustrate can be used for pieces of any size or complexity. It may be a class or a cluster of classes that are tightly coupled, but it is always a conceptually atomic unit. For example, it might be the set of classes that provide the functionality of binary tree. This would include an iterator class and a link class in addition to the tree class. The intent is that the coupling among internals of this component is more intense than the coupling of the component with any external objects.

It is the component testing perspective that is important and not the size of the pieces being tested. That perspective views the software being tested as intended for integration with other pieces rather than as a complete system in itself. This both helps to determine [3:25 pm, 18/12/2023] Sheenu: what features of the software are tested and how they are tested.

WHICH ONES SHOULD WE TEST?

One of the most intense arguments in testing object-oriented systems is whether detailed component testing is worth the effort. That leads me to state an obvious (at least in my mind) axiom: Select a component for testing when the penalty for the component not working is greater than the effort required to test it. Not every class will be sufficiently large, important or complex to meet this test so not every class will be tested independently. There are several situations in which the individual classes should be tested regardless of their size or complexity :

Reusable components - Components intended for reuse should be tested over a wider range of values than a component intended for a single focused use.

Domain components - Components that represent significant domain concepts should be tested both for correctness and for the faithfulness of the representation.

Commercial components - Components that will be sold, as individual products should be tested not only as reusable components but also as potential sources of liability.

14.3 TYPES OF TESTS

• **COMPATIBILITY TESTING**. Testing to ensure compatibility of an application or Web site with different browsers, OSs, and hardware platforms. Compatibility testing can be performed manually or can be driven by an automated functional or regression test suite.

- **CONFORMANCE TESTING.** Verifying implementation conformance to industry

standards. Producing tests for the behavior of an implementation to be sure it provides the portability, interoperability, and/or compatibility a standard defines.

• **FUNCTIONAL TESTING**. Validating an application or Web site conforms to its specifications and correctly performs all its required functions. This entails a series of tests, which perform a feature-by-feature validation of behavior, using a wide range of normal and erroneous input data. This can involve testing of the product's user interface, APIs, database management, Security installation, networking etc. testing can be performed on an automated or manual basis using black box or white box methodologies.

• **LOAD TESTING.** Load testing is a generic term covering Performance Testing and Stress Testing.

- **PERFORMANCE TESTING.** Performance testing can be applied to understandyour application or WWW site's scalability, or to benchmark the performance in an environment of third party products such as servers and middleware for potential purchase.This sort of testing is particularly useful to identify performance bottlenecks in high use applications. Performance testing generally involves an automated test suite as this allows easy simulation of a variety of normal, peak, and exceptional load conditions.

• **REGRESSION TESTING**. Similar in scope to a functional test, a regression test allows a consistent, repeatable validation of each new release of a product or Web site. Such testing ensures reported product defects have been corrected for each new release and thatno new quality problems were introduced in the maintenance process. Though regression testing can be performed manually an automated test suite is often used to reduce the time and resources needed to perform the required testing.

• **SMOKE TESTING.** A quick-and-dirty test that the major functions of a piece of software work without bothering with finer details. Originated in the hardware testing practice of turning on a new piece of hardware for the first time and considering it a success if it does not catch on fire.

• **STRESS TESTING.** Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements to determine the load under which it fails and how. A graceful degradation under load leading to non-catastrophic failure is the desired result. Often Stress Testing is performed using the same process as Performance Testing but employing a very high level of simulated load.

❖ **SELF CHECK EXERCISE**

I. In code testing, what does the analyst focus on executing?
   A. User interfaces
   B. Test cases
   C. Program instructions and paths
   D. System specifications

II. What is the expanded definition of testing that goes beyond code testing?
   A. Testing of user interfaces
   B. Testing of models, requirements, and design
   C. Manual testing processes
   D. Code optimization techniques

III. Why are test cases created as part of the testing process?
   A. To confuse developers
   B. To increase project costs
   C. To guide the search for errors
   D. To replace the review process

14.4 LEVELS OF TESTING

14.4.1 Unit Testing

In unit testing, the analyst tests the program making up a system. For this reason, unit testing is sometimes called program testing. Unit testing gives stress on the modules independently of one another, to find errors. This helps the tester in detecting errors in coding and logic that are contained within that module alone. The errors resulting from the interaction between modules are initially avoided. For example, a hotel information system consists of modules to handle reservations; guest check in and checkout; restaurant, room service and miscellaneous charges; convention activities; and accounts receivable billing.

For each, it provides the ability to enter, modify or retrieve data and respond to different types of enquiries or print reports. The test cases needed for unit testing should exercise each condition and option.

Unit testing can be performed from the bottom up, starting with smallest and lowest level modules and proceedings one at a time. For each module in bottom-up testing a short program is used to execute the

modules and provided the needed data, so that the modules is asked to perform the way it will when embedded within the larger system.

14.4.2 System Testing

The important and essential part of the system development phase, after designing and developing the software is system testing. We cannot say that every program or system design is perfect and because of lack of communication between the user and designer, some error is there in the software development. The number and nature of errors in a newlydesigned system depend on some usual factors like communication between the user and the designer; the programmer's ability to generate a code that reflects exactly the systems specifications and the time frame for the design.

Theoretically, a newly designed system should have all the parts or sub-systems are in working order, but in reality, each sub-system works independently. This is the time to gather all the sub-system into one pool and test the whole system to determine whether it meets the user requirements. This is the last change to detect and correct errors before the system is installed for user acceptance testing. The purpose of system testing is to consider all likely variations to which it will be subjected and then push the system to its limits.

Testing is an important function of the success of the system. System testing makes a logical

assumption that if all the parts of the system are correct, the goal will be successfully activated. Another reason for system testing is its utility as a user-oriented vehicle before implementation. System Testing consists of the following five steps:

- Program Testing
- String Testing
- System Testing
- System Documentation
- User acceptance testing

14.4.3 Program Testing

A program represents the logical elements of a system. For a program to run satisfactorily, it must compile and test data correctly and tie in properly with other programs. It is the responsibility of a programmer to have an error free program. At the time of testing the system, there exist two types of errors that should be checked. These errors are syntax and logical. A syntax error is a program statement that violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted key words are common syntactical errors. A logical error, on the other hand, deals with incorrect data fields out of range items, and invalid combinations. Compiler like syntax errors does not detect them; the programmer must examine the output carefully to detect them.

14.4.4 String Testing

Programs are invariably related to one another and interact in a total system. Each program is tested to see whether it conforms to related programs in the system. Each part of the system is tested against the entire module with both test and live data before the whole system is ready to be tested.

14.4.5 System Testing

It is deigned to uncover weaknesses that were not found in earlier tests. This includes forced system failure and validation of total system, as its user in the operational: environment will implement it. Under this testing, we take low volumes of transactions based on live data. This volume is increased until the maximum level for each transaction type is reached.

14.4.6 System Documentation

All design and test documentation should be well prepared and kept in the library for future reference. The library is the central location for maintenance of the new system.

14.4.7 User Acceptance Testing

A acceptance test has the objective of selling the user on the validity and reliability of the system. It verifies that the system's procedures operate to system specifications and that the integrity of important data is maintained. Performance of an acceptance test-is. actually the user's show. User motivation is very important for the successful performance of the system.

❖ **SELF-CHECK EXERCISE**

IV. Which testing level is designed to uncover weaknesses not found in earlier tests?
A. Program testing
B. String testing
C. System testing
D. User acceptance testing

V. In program testing, what type of errors does a programmer examine the output for?
A. Interaction errors
B. Syntax errors
C. Logical errors
D. System errors

VI.What does user acceptance testing aim to verify?
A. Interaction between modules
B. System documentation
C. Integrity of important data
D. Syntax errors

14.5 INTRODUCTION TO SOFTWARE QUALITY ASSURANCE

As discussed in the previous lectures, the analysts' objective is always to develop a computer based system, which will fulfill the requirements as per the study and design specifications with no failure rates and higher level of customer satisfaction. In order to develop a computer based system with lesser probability of failures and higher levels of customer satisfaction, an analyst has to ensure that the computer based system developed is correct, reliable, accurate, efficient and consistent etc. Hence the analysts must include checks and reviews in the development process of the computer based system to ensure that the implemented system meets the requirements of the uses and achieves higher level of customer satisfaction. To ensure that the computer-based system/software is of high quality a system analyst should check and review each and every process development life cycle from requirement analysis to implementation within the system

The "quality" can also be described as a combination of few questions, which are to be answered by the system analyst while developing a computer, based system / software and the questions are given below.

(i) What is to be done?

(ii) How it is to be done?

(iii) What is being done?

(iv) How can it be done more efficiently?

The word "quality" can also he described as developing- error free computer Based systems/software with minimum usage of resources.

The software life cycle includes various stages of development and each has the goal of quality assurance. Steps taken in this regard are summarized below:

14.6 QUALITY FACTORS SPECIFICATIONS

The goal of this stage is to describe various factors mainly responsible for quality of the proposed system. They are as follows:

(i) Correctness: The extent to which a program meets system specifications and user.

(ii) Reliability: The degree to which the system performs its intended functions overa time.

(iii) Efficiency: Computer resources required by a program to perform a particularfunction.

(iv) Usability: The efforts required to understand and operate a system.

(v) Maintainability: The ease with which the program errors are detected and removed.

(vi) Testability: The effort required to test the program to ensure its correctperformance.

(vii) Portability: The ease of transporting a program from one hardware configurationto another.

(viii) Accuracy: The required precision in input, editing, computations and output.

(ix) Error Tolerance: Error detection and correction versus error avoidance.

(x) Expandability : Ease of expanding the existing database.

(xi) Access control and audit: Control of access to the system and the extent to which that access can be audited.

(xii) Communicativeness: Usefulness and effectiveness of the inputs and outputs of the system.

Software Requirements Specifications

The quality assurance goal of this stage is to generate the requirements document that helps in providing technical specifications for developing the software.

Software Design Specifications

In this stage, the software design document defines the overall architecture of the software that provides the functions and features given in the software requirements document.

Software Testing and Implementation

The quality assurance goal of the testing phase is to ensure that completeness and accuracy of the system and minimize the retesting process. In the implementation phase, the goal is to provide logical order for the creation of the modules, and in turn, the creation of the system.

Maintenance and Support

This phase provides the necessary software development for the system to continue to comply with the original specifications. The quality assurance goal is to develop a procedure for correcting errors and enhancing software.

14.7 LEVELS OF QUALITY ASSURANCE

Analysts use three levels of quality assurance: testing, verification with validation and certification.

Testing

System Testing is quite expensive and time-consuming process. The common view of testing held by users is that it is performed to prove that program is error free. But this is quite difficult since the analyst cannot prove that software is free from all sorts of errors.

Verification with Validation

Like testing, verification is also intended to find errors. Executing a program in a simulated environment to find errors performs it.

When commercial systems are developed with the main aim of distributing them to dealers for sale purposes, they first go through verification, sometimes called alpha testing.

The feedback from the validation phase generally brings some changes in the software to deal with errors and failure that are uncovered. Then a set of user sites is selected for putting the system into use on a live basis.

Certification

The last level of quality assurance is to certify that the software packages developed conforms to standards. With a growing demand for purchasing ready-to-use software, importance of certification has increased. A package that is certified goes through a team of computer specialists who test, review and determine how well it meets the user's requirements and vendor's claims. Certification is issued only if the package is successful in all the tests. Certification, however, does not mean that it is best package to adopt. It only attests that it will perform what the vendor claims.

❖ **SELF-CHECK EXERCISE**

VII.What is the primary goal of quality assurance in the software development life cycle?
A. Error-free program execution
B. Minimization of resource usage
C. Customer dissatisfaction
D. Compliance with system specifications

VIII. How can "quality" in software development be described?
A. Maximum usage of resources
B. Error-free systems with minimum resource usage
C. High failure rates
D. Incomplete system specifications

IX. Which quality factor refers to the degree to which a program meets system specifications and user requirements?
A. Efficiency
B. Usability
C. Reliability
D. Testability

14.8 SUMMARY

This text delves into the philosophy of software testing, emphasizing the importance of test cases in finding errors. It discusses two testing strategies—code testing and specifications testing. The document expands the definition of testing beyond code, encompassing models, requirements, analysis, and design. It introduces the concept of component testing, discussing its importance irrespective of the size or complexity of the components. The document then outlines different types of tests, including compatibility testing, conformance testing, functional testing, load testing, regression testing, smoke testing, and stress testing. The levels of testing—unit testing, system testing, and user acceptance testing—are explored, along with the role of system documentation in the testing process. The document concludes by introducing software quality assurance, covering its levels—testing, verification with validation, and certification.

14.9 KEYWORDS

1.Quality Assurance: Systematic activities to ensure that software processes, products, and services conform to specified requirements and standards.

2. Verification: The process of evaluating a system or component during or at the end of the development process to ensure that it satisfies the specified requirements.

3.Validation: The process of evaluating a system or component during or at the end of the development process to ensure that it meets the user's needs and expectations.

4.Certification: The formal process of confirming that a software package or system meets specified standards or requirements.

5.Unit Testing: The testing of individual units or components of a software application to ensure their correctness and proper functioning.

6.Functional Testing: The type of testing that validates whether a software application performs its specified functions as intended.

7.Load Testing: The process of testing a software application's performance under normal and peak load conditions to assess its scalability and identify performance bottlenecks.

14.10 PRACTICE QUESTIONS

14.10.1 Short Answer Questions

Q1. What is the primary goal of testing in software development?

Q2. How does the expanded definition of testing include early development phases?

Q3. Explain the importance of test cases in the testing process.

Q4. What is the significance of component testing, and what perspective is crucial in this process?

Q5. Enumerate three quality factors specifications in the software development life cycle.

Q6. What are the three levels of quality assurance mentioned in the document?

14.10.2 Long Answer Questions

Q1.Discuss the philosophy behind software testing and how it contributes to the development of high-quality systems.

Q2. Explain the differences between code testing and specifications testing, highlighting their respective approaches and objectives.

Q3. Elaborate on the concept of component testing and why it is considered important, irrespective of component size.

Q4. Provide an overview of different types of tests mentioned in the document, emphasizing their specific purposes in the testing process.

Q5. Explore the levels of testing, including unit testing, system testing, and user acceptance testing, and their roles in ensuring software quality.

14.11 REFERENCES

Pfleeger, S. L., & Atlee, J. M. (2010). Software Engineering: Theory and Practice (4th ed.). Pearson.

Kaner, C., Falk, J., & Nguyen, H. Q. (1999). Testing Computer Software (2nd ed.). Wiley.

Beizer, B. (1995). Software Testing Techniques (2nd ed.). Van Nostrand Reinhold.

Myers, G. J. (1979). The Art of Software Testing. John Wiley & Sons.

Humphrey, W. S. (1989). Managing the Software Process. Addison-Wesley.

IEEE Standard Glossary of Software Engineering Terminology. (1990). IEEE.

Boehm, B. W. (1981). Software Engineering Economics. Prentice-Hall

14.12 ANSWER KEY (SELF CHECK EXERCISE)

I.C, II.B, III.C, IV.C,V.C,VI.C , VII.D, VIII.B,IX.C

Lesson No. 15 **SYSTEM ANALYSIS AND DESIGN**

AUTHOR : SANJEEV KUMAR MODI

PROCESSING OF AN EXPORT ORDER- EXPORT PROCEDURE

STRUCTURE

15.0 Objectives

15.1 Introduction

15.2 Various Activities During the Implementation Phase

- Self-Check Exercise

15.3 Implementation Strategies

15.4 Post Implementation

15.5 Maintenance Issues

- Self-Check Exercise

15.6 Summary

15.7 Keywords

15.8 Practice Questions

15.8.1 Short Answer Type Questions

15.8.2 Long Answer Type Questions

15.9 References

15.10 Answer key

15.0 OBJECTIVES

After Reading this Chapter, student will be able to

Understand the significance of the implementation phase in the software development life cycle and explore the various activities involved in the implementation phase, including installation planning and physical procedures.

Examine data preparation and conversion strategies for a successful implementation and recognize the importance of user training and parallel run in the implementation process.

Investigate the approval and contingency planning stages post-implementation.

15.1 INTRODUCTION

In the classic life cycle, a lot of emphasis is laid on all other phases' viz., analysis, design and coding. Implementation normally gets related to the background and is generally talked about in a rather cursory manner by most books that you would come across on software engineering.

You might be tempted to think therefore that implementation is a rather routine and straightforward affair- after all, what you need to do is take the software that has been thoroughly tested by the development team, install it in the appropriate places and make it run. Life is never quite as simple as that! It is estimated that more than 80% of the problems encountered in software development projects get thrown up only at the time of implementation.

How then can we afford to ignore this phase?

15.2 VARIOUS ACTIVITIES DURING THE IMPLEMENTATION PHASE

Several issues, which might very often be taken, for granted, are actually integral to the successful usage of any software. These include issues like user training, documentation, rigorous enforcement of usage discipline etc.

Once the software is coded and tested, the package is ready for implementation. As have seen the module on testing, after the various stages of internal testing are completed, the software is then put through an acceptance test. Here the package is tested out comprehensively in front of the user, so that the user is convinced that every aspect of the software works, as it was intended to. Once the software goes through the acceptance test successfully, it is ready for implementation.

Unlike the other phases, there are no standard methodologies for the Implementation phase. But Implementation Planning is an important activity of the SDLC. The specific approach adopted for implementation is largely dependent on the size of the organization, the nature of applications and the standard practices which the project team might have evolved.

However, despite such individual variations, there are some standard activities, which would have tobe performed. We shall take a look at the more important of these.

15.2.1 Installation Plan

Implementation is a phase where a number of activities that have been going onparallelly so far have to culminate. To ensure that all these activities indeed coverage at the right time and at the right place without any delays on account of unforeseen problems, to get the soft ware to be used as it should,is what managing the implementation phase is all about.

An important activity in this phase is preparation of the installation plan. Much before the software could be actually ready , a detailed plan has to be prepared in term of hardware procurement and installation.

What machines are to be installed at which location and at what point in time, provision of the necessary infrastructure for installation of the same, site preparation , arranging any machine specific or commodity software specific training for different categories of users etc. are details which need to be worked out.

Likewise a plan has also got to be made for installation of the software to be implemented. It is exceedingly important to ensure that the right version of the software is installed. By the time this phase of software development is reached, the software would already have undergone several iterative changes.

Version control has to be maintained very scrupulously. We will look at this topic in greater detail under Configuration Management. The eventual objective is to ensure that the latest update of the software (after rigorous testing) is what is installed at the client's site. Different modules of the package might be relevant in different locations. Details of which modules are to be enabled/disabled in each location have to be worked out.

Depending on the Implementation Strategy adopted, the time at which these have to be installed may also differ. Therefore a comprehensive installation plan needs to be in place, which gives location wise installation details as relevant to different user groups.

15.2.2 Implementation of Physical Procedures

The successful implementation of systems depends as much on the adaptability of the people using them, as it does on the quality of software. Often times one tends to concentrate only on the latter while taking the former for granted.

You will be amazed to know the kind of problems that have been faced on account of the mental barriers people have to any kind of change, howsoever small it may be

It always takes time to unlearn what we have been doing all along and get used to a new set of practices. Some people adapt very fast, others take time, while there may be a few others who might just give up.

The last category is the one that we have to be wary of. Successful implementation depends to a large extent on the successful adaptation of people to the new scheme of things.It is very important therefore to 'carry the people along'. Incomplete or erroneous understanding of the new systems may also result in implementation hitches.

So how does one contend with such problems?

The issues can be addressed through a combination of proper communication, training and practice. What might seem quite complicated at first becomes a matter of routine with time.

Do you find it difficult to remember the route from here to your home or the telephone numbers of your best friends, however complicated they might be? Obviously you don't.

Quite simply because these are things which you use frequently. So is the case with adapting to new procedures on account of automation.

Much before implementation actually commences, the Analysis Team would have spelt out the operational differences in the proposed system through the Implementation Model. Likewise, the Design Team would have specified further refinements and details of the new set of procedures.

This might include establishment of new coding schemes, switching over to new document formats, changes in existing chronology of operations, additional activities to be performed or existing activities to be dispensed with and so on.

15.2.3 Data Preparation And Conversion

Data preparation is normally the most time consuming and tedious task in the implementation of most systems. Assuming the system is expected to go live from a particular date, all backlog of data has to be transcribed to fit into the new formats and coding schemes, and rigorously error checked to ensure no loDepending on whether the organization is switching over from some existing computerized systems or is starting afresh, this might be achieved through a set of conversion programs and utilities or by entering all data afresh or a combination of both.

This would depend on the Conversion Strategy adopted for implementation.

This involves careful co-ordination between the development team who would provide the data transcription procedures, formats and conversion software and the users who need to carefully understand the details and go about them.

Often times this entails deployment of additional manpower to cope with the enormous transcription and data entry work that this might entail. Some organizations even choose to subcontract such jobs to external agencies on account of the data volumes involved.

Data preparation could broadly be categorized into master data and transaction data preparation. Master data preparation is an activity, which involves compiling an exhaustive list of all the pertinent data relating to information that is relatively static over a period of time but might be required at any time in future as well.

The volume of transaction data preparation would be dictated by not only the data pertaining to the current period, but also past details. The exact volumes and the associated effort would be determined by what extent of historical information the company would like to maintain. The data entered has got to be thoroughly error checked so that there are no problems in future on account of faulty data.

15.2.4 Conversion Testing

Where data is converted from existing files or databases, rigorous Conversion Testing needs to be done to ensure proper conversion. The following is generally done as part of **Conversion testing**:

- Tallying of input and output records.
- Use of custom-built verification software, which does the following:
- Checks value ranges of fields within each record.

- Checks whether the required relationships between records is maintained.
- Extraction pf sample records through a statically proven random sampling method from both the old and the new set of records. These are then directly compared with each other.

15.2.5 User Training

The successful implementation of any software is dependent in good measure on the quality of training imparted, Different user groups need to be identified.

A Training Need Analysis has to be done. for each of these groups to find out what kind of training is required for each. Getting trained on aspects that are not relevant may sometimes tend to confuse the users.

So the training program has to be very carefully worked out.

One aspect that tends to get ignored by most people is the timing of the training. In an organization's enthusiasm to proceed with things the training programs may be scheduled very much in advance. By the time the software is ready for implementation, most users would have already forgotten what was taught to them earlier. It is important therefore to time the training programs appropriately. Hands-on training can follow this up on the software once.

15.2.6 Parallel Run

Once the software is installed, the users are trained and the master data is created, it is time to start full-fledged use of the software. However, seldom does it happen that the moment the software is commissioned and goes live, the organization relies solely on it, discarding all previously used processes and procedures.

What generally happens is that usage of the software is commenced? In addition, the old system is also continued till such time that stability is reached and all the users are confident that the new system is foolproof and that the old one can be dispensed with

Therefore the basic idea underlying this activity is that even after the software is installed, the old system of doing things be it automated or manual is continued for some period of time. Since both the old and the new systems continue in parallel, this is called the Parallel Run.

The period for the parallel run varies from organization to organization and from application to application. Although there are no hard and fast rules for this phase, it is attempted to ensure that the following factors are looked into:

- There are no major faults in the package.
- Important figures tally.
- All modules are used and found satisfactory.
- There is a fair amount of confidence amongst users is being able to use the software.

If there are any gaps in any of the above factors, the problems are plugged. If there are no problems encountered then a decision is taken to discontinue the old system and rely exclusively on the new software. This connotes the end of the parallel run.

15.2.7 Implementation Approval

Once the software has gone live and has been working successfully for a certain period of time, it can be reasonably assumed that there are no hidden bugs in the same.

During this period the users would also need to ensure that every aspect of the software has been tried out and found to be working as expected.

Although it is difficult to expect all exception conditions to occur during this time, the users have to take care of such contingencies by simulating such conditions (to theextent that they can be foreseen) during the acceptance-testing phase.

If there are any problems detected during this phase or during the parallel run, then it has to be rectified immediately by the developers (provided it is within the scope of what was agreed upon earlier). The revised software has then got to be once again tested rigorously and installed. If the software is found to be functioning satisfactorily and the manuals are all found to be in order then the user is expected to give her formal approval of the software implemented. Managing each of the above activities is critical to the successful implementation of any software. What are the strategies that one can adopt for the above; what are the problems that one is likely to encounter in the process - these are issues that we will look at in the next session.

- **SELF-CHECK EXERCISE**

I. What phase is often overlooked in the classic software life cycle?
A. Analysis
B. Design
C. Implementation
D. Coding

II.What is the primary purpose of an installation plan in the implementation phase?
A. Designing software modules
B. Hardware procurement and installation details
C. User training schedules
D. Coding and testing procedures

III. What is crucial for successful implementation according to the passage?
A. Technical complexity
B. Adaptability of users
C. Coding efficiency
D. Software design

IV. The period where both old and new systems run simultaneously is termed _____ run.

15.3 IMPLEMENTATION STRATEGIES

In the above section, we have seen the main activities during the Implementation phase. Here, we will take a look at some of the oft-used strategies during implementation.

There is no standard strategy as far as implementation of software is concerned. The strategy to be adopted is generally developed by a senior consultant if the software has been contracted out) or by a person with sizeable experience if the package is being developed internally.

It is the experience and acumen of the person arriving at the strategy which ensures smooth implementation, for this phase involves a number of both small and large issues, which if missed out or not dealt with properly could result in a lot of delays and added costs.

The expert therefore has to rely largely on her experience and insight gleaned from other projects to arrive at the most appropriate plan of action. Implementation strategies could depend on a number of factors, which could include:

- The nature and size of the application
- Type of organization
- The organizational culture
- End user profiles
- Previous exposure of the organization to automation
- Existence or otherwise of an internal EDP / Systems department
- Number of personnel involved in usage of software
- Data volumes
- Locational spread of the application
- Type of hardware and availability of certain tools

- Manpower availability
- Critically of the application etc.

Despite the large number of imponderables involved, here we will try and generalize on some frequently used strategies, one or a combination of which might be used for implementation.

15.3.1 Data Conversion Strategies

As discussed earlier data preparation and conversion is one of the most time consuming activities during implementation. The methods adopted for the same would depend on various factors. Let us look at some of the approaches:

(a) When data is available as part of some existing applications.

In such cases data is generally not re-entered. A set of conversion programs is written to convert existing data into the new formats. This task gets more complicated if coding schemes are different. This would necessitate building up of the necessary look-up files for code conversion.

Generally a database auditor is used to verify that the conversion has taken place properly. As explained during the earlier session, it is important to ensure that Conversion Testing is carried out properly to ensure that there are no unexpected problems in future.

(b) When application is being automated for the first time.

Here the data entry programs, which are a part of the application itself, can be used to enter data. This would be a fairly straightforward and foolproof method, because the data entry programs would themselves take care of all the error checking and validations.

(c) When volumes of data are very high.

Screen oriented data entry becomes very time consuming for very large volume data.

In these cases it is common to use special purpose data entry packages to facilitate high speeds of data entry by professional data entry operators. A suit of programs is then built to validate data thus entered, if necessary.

15.3.2 Parallel Run

This is a very commonly used strategy during implementation. So much so that a number of people treat the parallel run as an integral activity during implementation.

During the parallel run, reports and figures being produced from the new system can be verified with the old one to ensure that the new system is producing results, as it should.

Although the formats and the number of outputs from the two systems are bound to be different, some of the more critical figures can always be compared and reconciled. This gives an enormous amount of confidence to the users that the new system is indeed accurate.

It also ensures that in case something is drastically wrong with the software, to the extent that it cannot be used, the activities in the organization do not come to a grinding halt, since the system is old anyway.

In reality though this is really a double-edged sword as we will discuss in detail under implementation problems.

15.3.3 Direct Implementation

This is an approach wherein the new system becomes operational directly from day one. This is usually done from the first day of the month or the year. This becomes the cutoff period for the changeover to the new system. The old systems are dispensed with right away and all personnel are re-deployed to manage and maintain the new system.

This strategy is generally adopted for small and medium sized applications that are not highly critical. As you can see, the criticality factor is an important consideration for taking such decisions.

The biggest advantage of this the approach is that it completely disables users' tendency to slip back into their old practices, which they have been adopting for years and are therefore happy and comfortable with.

15.3.4 Contingency Planning

Howsoever good the quality of software might be, however appropriate the strategy for implementation and however good the quality of training, it is never quite possible to ensure a 100% successful implementation.

In a manner of speaking software implementation is a bit of a gamble. It could result in fantastic results or it might fail for the silliest of reasons.

You will be amazed to know how one of the most professionally developed packages with the best manpower and very stringent quality control procedures, for a large public sector organization was a dismal failure when it came to implementation.

Let me recount my experience here. The project had been very carefully planned.Hardware was procured and installed. The users were very happy with the results of the acceptance tests. All relevant data had been entered and thoroughly checked. The package was an industrial first and given its scope of coverage it was expected to work wonders for the organization. User groups had been trained and were very eager to start using the package.

What do you think went wrong?

The State had a power crisis and the State government decided to impose a 4-hour power cut on all industrial establishments. The organization did have generators, but these supported only the more critical activities and departments. So all plans of the organization to have a 24-hour on-line package flew out of the window, at least temporarily.

Was Implementation a success?

Far from it, and for no apparent fault of any of the concerned parties.

It is in such cases that one realizes the importance of contingency planning. It quite simply means planning for an eventuality where the software might be rendered either partly or fully unusable or where there is a significant shortfall in performance. Obviously contingency plans are very specific to the nature of the project and organization. However what is generally done is to ensure the presence of a back up or a parallel system so that should the main system fail the back-up system (either manual or automated) can be resorted to

15.4 POST IMPLEMENTATION

The formal acceptance of the software by the user does not connote severing of the relationship between the developer and the user. The software business is never quite as simple as that!

Irrespective of the quality of the product, the user will continue to need the support of the developers on an on-going basis. This could be on account of the following:

- Additional Training Programs either for new set of users or as a refresher for existing users.
- Trouble shooting on account of problems encountered by the users.
- Bug fixes.
- Modifications to existing requirements.
- Additional Requirements.
- Porting onto a different platform...etc.

All these requirements are taken care of by the client signing up for a formal contract with the developer, which is called an Annual Maintenance Contract.

While changes within the purview of what has been done earlier, as well as support in terms of trouble shooting is generally taken up as part of the AMC, additional requirements with respect to either the software or training are costed on a case-to-case basis.

15.5 MAINTENANCE ISSUES

Many studies at the private, University and government level have been conducted to learn about maintenance requirements for information systems. These studies reveal the following facts:

(a)From 60 to 90 percent of the overall cost of software during the life of a system is spent on maintenance

(b) Often maintenance is not done very efficiently.

(c) Software demand is growing at a faster rate than supply. Many programmers are spending more time on system maintenance than on new software development.

- **SELF-CHECK EXERCISE**

V. Which implementation strategy involves the direct operation of the new system from day one?
A. Parallel Run
B. Direct Implementation
C. Contingency Planning
D. Maintenance Strategy

VI. What is the primary purpose of contingency planning in software implementation?
A. Ensuring smooth data conversion
B. Managing user training schedules
C. Planning for unforeseen events or system failures
D. Implementing parallel run strategies

VII. What is an essential ongoing requirement after the formal acceptance of software by the user?
A. Annual software development
B. Continuous user training
C. Implementation phase
D. Annual Maintenance Contract (AMC)

15.6 SUMMARY

The implementation phase in the classic software development life cycle is crucial, and issues can arise despite thorough testing. Various activities, including installation planning, physical procedures, data preparation, and user training, play vital roles. Data conversion strategies, such as parallel run and direct implementation, are explored. Contingency planning is emphasized to address unforeseen challenges. Post-implementation involves approval, ongoing support, and maintenance considerations.

15.7 KEYWORDS

1. Software Implementation: The process of deploying and executing a software application or system after development, encompassing installation, data conversion, user training, and transitioning to live operation.

2. Data Conversion: The process of transforming data from existing formats or databases to fit the requirements of a new system, often using conversion programs and utilities.

3. User Training: The systematic imparting of knowledge and skills to end-users, enabling them to effectively operate and interact with the newly implemented software.

4. Contingency Planning: The preparation for unforeseen events or failures during software implementation, involving backup systems or procedures to maintain operational continuity.

5. User Acceptance Testing (UAT): The phase in software development where end-users evaluate and validate the software to ensure it meets their requirements and expectations.

6. Configuration Management: The discipline of systematically controlling changes to software, hardware, or documentation throughout the development and implementation process.

15.8 PRACTICE QUESTIONS

15.8.1 Short Answer Questions

Q1. What is the purpose of the installation plan in the implementation phase?

Q2. Explain the concept of parallel run during software implementation.

Q3. Why is user training essential for successful software implementation?

Q4. What are the key activities involved in data preparation and conversion?

Q5. What is the significance of implementation approval in the software development life cycle?

15.8.2 Long Answer Questions

Q1. Discuss the challenges and importance of user adaptation during software implementation.

Q2. Explain the role of contingency planning in ensuring successful software implementation.

Q3. Compare and contrast different data conversion strategies in the implementation phase.

Q4. Discuss the factors influencing the choice of implementation strategies in software development.

Q5. Prepare the data description formats for all files in the Finished Goods System.

15.9 REFERENCES

Pressman, R. S. (2014). Software Engineering: A Practitioner's Approach. McGraw-Hill Education.

McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction. Microsoft Press.

Sommerville, I., & Kotonya, G. (1998). Requirements Engineering: Processes and Techniques. Wiley.

Humphrey, W. S. (1989). Managing the Software Process. Addison-Wesley.

IEEE Standard Glossary of Software Engineering Terminology. IEEE.

Yourdon, E., & Constantine, L. L. (1979). Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Prentice-Hall.

Davis, A. M. (1987). Software Requirements: Analysis and Specification. Prentice-Hall.

15.10 ANSWER KEY (SELF- CHECK EXERCISE)

I.C,II.B, III.B, IV.PARALLEL RUN,V.B, VI.C, VII.D