UNIT 2 PRINCIPLES OF SOFTWARE REQUIREMENTS ANALYSIS

Structure **Page Nos.**

2.0 INTRODUCTION

In the design of software, the first step is to decide about the objectives of software. This is the most difficult aspect of software design. These objectives, which the software is supposed to fulfill are called *requirements*.

The IEEE [IEEE Std 610.12] definition of requirement is:

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system component, to satisfy a contract formally imposed document.
3. A documented representation of a condition or capability as in (1) or (2).

Thus, requirements specify "what the system is supposed to do?" These requirements are taken from the user. Defining the requirements is most elementary & most difficult part of system design, because, at this level, sometimes, the user himself is not clear about it. Many software projects have failed due to certain requirements specification issues. Thus, overall quality of software product is dependent on this aspect. Identifying, defining, and analysing the requirements is known as requirements analysis. Requirements analysis includes the following activities:

1. Identification of end user's need.
2. Preparation of a corresponding document called SRS (Software Requirements Specification).

3. Analysis and validation of the requirements document to ensure consistency, completeness and feasibility.
4. Identification of further requirements during the analysis of mentioned requirements.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- understand the significance of requirements analysis;
- develop SRS, and
- know various tools and techniques used for requirements analysis.

2.2 ENGINEERING THE PRODUCT

Due to the complexity involved in software development, the engineering approach is being used in software design. Use of engineering approach in the area of requirements analysis evolved the field of Requirements Engineering.

2.2.1 Requirements Engineering

Requirements Engineering is the systematic use of proven principles, techniques and language tools for the cost effective analysis, documentation, on-going evaluation of user's needs and the specification of external behaviour of a system to satisfy those user needs. It can be defined as a discipline, which addresses requirements of objects all along a system development process.

The output of requirements of engineering process is Requirements Definition Description (RDD). Requirements Engineering may be defined in the context of Software Engineering. It divides the Requirements Engineering into two categories. First is the requirements definition and the second is requirements management.

Requirements definition consists of the following processes:

1. Requirements gathering.
2. Requirements analysis and modeling.
3. Creation of RDD and SRS.
4. Review and validation of SRS as well as obtaining confirmation from user.

Requirements management consists of the following processes:

1. Identifying controls and tracking requirements.
2. Checking complete implementation of RDD.
3. Manage changes in requirements which are identified later.

2.2.2 Types of Requirements

There are various categories of the requirements.

On the basis of their priority, the requirements are classified into the following three types:

1. Those that should be absolutely met.
2. Those that are highly desirable but not necessary.
3. Those that are possible but could be eliminated.

On the basis of their functionality, the requirements are classified into the following two types:

i) **Functional requirements**: They define the factors like, I/O formats, storage structure, computational capabilities, timing and synchronization.

ii) **Non-functional requirements**: They define the properties or qualities of a product including usability, efficiency, performance, space, reliability, portability etc.

2.2.3 Software Requirements Specification (SRS)

This document is generated as output of requirement analysis. The requirement analysis involves obtaining a clear and thorough understanding of the product to be developed. Thus, SRS should be consistent, correct, unambiguous & complete, document. The developer of the system can prepare SRS after detailed communication with the customer. An SRS clearly defines the following:

- External Interfaces of the system: They identify the information which is to flow '*from and to'* to the system.
- Functional and non-functional requirements of the system. They stand for the finding of run time requirements.
- Design constraints:

The SRS outline is given below:

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Overall description
   - 2.1 Product perspective
   - 2.2 Product functions
   - 2.3 User characteristics
   - 2.4 Constraints
   - 2.5 Assumptions and dependencies
3. Specific requirements
   - 3.1 External Interfaces
   - 3.2 Functional requirements
   - 3.3 Performance requirements
   - 3.4 Logical Database requirements
   - 3.5 Design Constraints
   - 3.6 Software system attributes
   - 3.7 Organising the specific requirements
   - 3.8 Additional Comments
4. Supporting information
   - 4.1 Table of contents and index
   - 4.2 Appendixes

2.2.4 Problems in SRS

There are various features that make requirements analysis difficult. These are discussed below:

1. Complete requirements are difficult to uncover. In recent trends in engineering, the processes are automated and it is practically impossible to understand the complete set of requirements during the commencement of the project itself.
2. Requirements are continuously generated. Defining the complete set of requirements in the starting is difficult. When the system is put under run, the new requirements are obtained and need to be added to the system. But, the project schedules are seldom adjusted to reflect these modifications. Otherwise, the development of software will never commence.
3. The general trends among software developer shows that they have over dependence on CASE tools. Though these tools are good helping agents, over reliance on these Requirements Engineering Tools may create false requirements. Thus, the requirements corresponding to real system should be understood and only a realistic dependence on tools should be made.
4. The software projects are generally given tight project schedules. Pressure is created from customer side to hurriedly complete the project. This normally cuts down the time of requirements analysis phase, which frequently lead to disaster(s).
5. Requirements Engineering is communication intensive. Users and developers have different vocabularies, professional backgrounds and psychology. User writes specifications in natural language and developer usually demands precise and well-specified requirement.
6. In present time, the software development is market driven having high commercial aspect. The software developed should be a general purpose one to satisfy anonymous customer, and then, it is customised to suit a particular application.
7. The resources may not be enough to build software that fulfils all the customer's requirements. It is left to the customer to prioritise the requirements and develop software fulfilling important requirements.

2.2.5 Requirements Gathering Tools

The requirements gathering is an art. The person who gathers requirements should have knowledge of what and when to gather information and by what resources. The requirements are gathered regarding organisation, which include information regarding its policies, objectives, and organisation structure, regarding user staff. It includes the information regarding job function and their personal details, regarding the functions of the organisation including information about work flow, work schedules and working procedure.

The following four tools are primarily used for information gathering:

1. **Record review:** A review of recorded documents of the organisation is performed. Procedures, manuals, forms and books are reviewed to see format and functions of present system. The search time in this technique is more.
2. **On site observation:** In case of real life systems, the actual site visit is performed to get a close look of system. It helps the analyst to detect the problems of existing system.

3. **Interview:** A personal interaction with staff is performed to identify their requirements. It requires experience of arranging the interview, setting the stage, avoiding arguments and evaluating the outcome.

4. **Questionnaire:** It is an effective tool which requires less effort and produces a written document about requirements. It examines a large number of respondents simultaneously and gets customized answers. It gives person sufficient time to answer the queries and give correct answers.

☞ Check Your Progress 1

1) Why is it justified to use engineering approach in requirements analysis?

……

……

……

2.3 MODELING THE SYSTEM ARCHITECTURE

Before the actual system design commences, the system architecture is modeled. In this section, we discuss various modeling techniques.

2.3.1 Elementary Modeling Techniques

A model showing bare minimum requirements is called **Essential Model**. It has two components.

1. **Environmental model**: It indicates environment in which system exists. Any big or small system is a sub-system of a larger system. For example, if software is developed for a college, then college will be part of University. If it is developed for University, the University will be part of national educational system. Thus, when the model of the system is made these external interfaces are defined. These interfaces reflect system's relationship with external universe (called environment). The environment of a college system is shown in *Figure 2.1*.

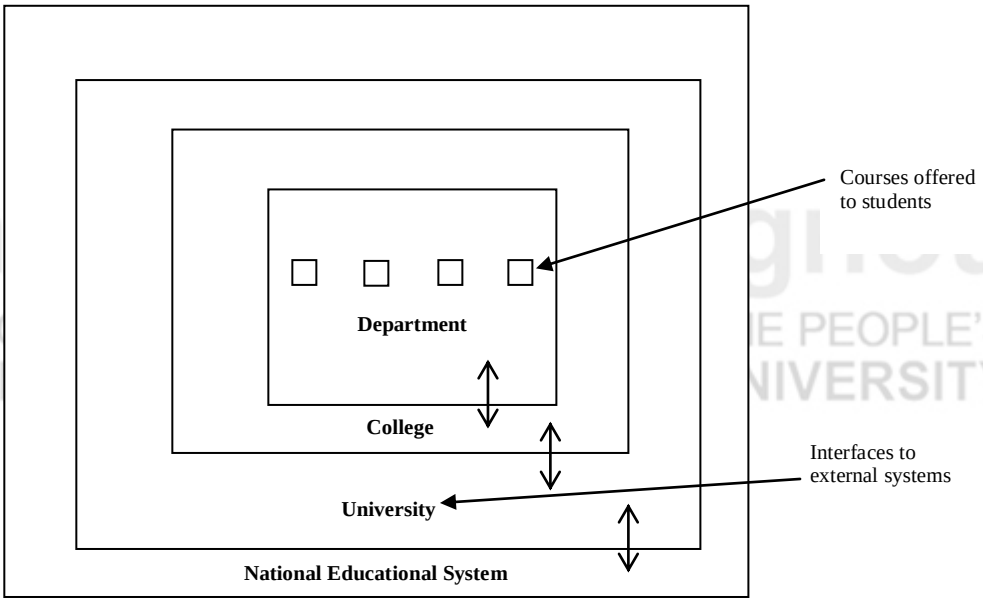


Figure 2.1 : Environmental model of educational system

In environmental model, the interfaces should clearly indicate the inflow and outflow of information from the system.

The tools of environment model are:

(i) **Statement of purpose**: It indicates the basic objectives of system.

(ii) **Event list**: It describes the different events of system and indicates functionality of the system.

(iii) **Context diagram**: It indicates the environment of various sub-systems.

2. **Behavioural Model:** It describes operational behaviour of the system. In this model, various operations of the system are represented in pictorial form. The tools used to make this model are: Data Flow Diagrams (DFD), E-R diagrams, Data Dictionary & Process Specification. These are discussed in later sections.

   Hence, behavioural model defines:

   Data of proposed system.

   (i) The internal functioning of proposed system,
   (ii) Inter-relationship between various data.

In traditional approach of modeling, the analyst collects great deal of relatively unstructured data through data gathering tools and organize the data through system flow charts which support future development of system and simplify communication with the user. But, flow chart technique develops physical rather than logical system.

In structured approach of modeling the standard techniques of DFD, E-R diagram etc. are used to develop system specification in a formal format. It develops a system logical model.

2.3.2 Data Flow Diagrams (DFD)

It is a graphical representation of flow of data through a system. It pictures a system as a network of functional processes. The basis of DFD is a data flow graph, which pictorially represents transformation on data as shown in *Figure 2.2*.

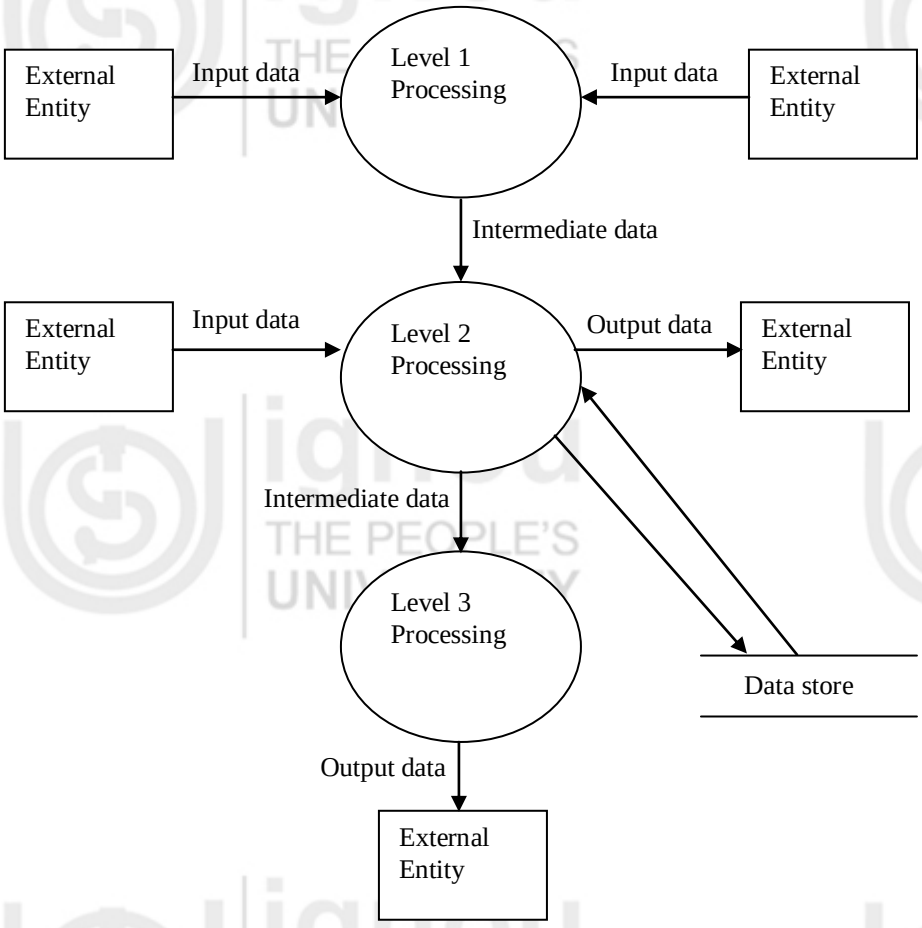


Figure 2.2: Data flow diagram

In this diagram, the external entities provide input data for the processing. During the processing, some intermediate data is generated. After final processing, the final output data is generated. The data store is the repository of data.

The structured approach of system design requires extensive modeling of the system. Thus, instead of making a complete model exhibiting the functionality of system, the DFD's are created in a layered manner. At the first layer, the DFD is made at block level and in lower layers, the details are shown. Thus, level "0" DFD makes a fundamental system (*Figure 2.3*).

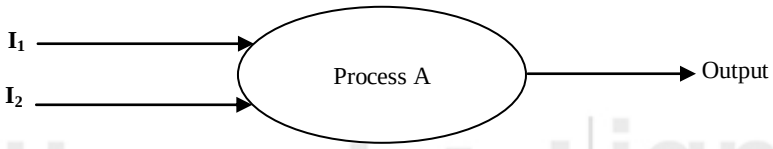


Figure 2.3: Layer 1 depiction of process A

DFD's can represent the system at any level of abstraction. DFD of "0" level views entire software element as a single bubble with indication of only input and output data. Thus, "0" level DFD is also called as Context diagram. Its symbols are shown in *Figure 2.4.*

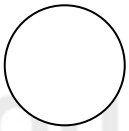
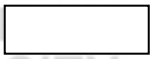
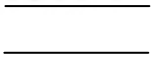
Symbol	Name	Description
	Data Flow	Represents the connectivity between various processes
	Process	Performs some processing of input data
	External Entity	Defines source or destination of system data. The entity which receives or supplies information.
	Data Store	Repository of data

Figure 2.4: Symbols of a data flow diagram

2.3.3 Rules for making DFD

The following factors should be considered while making DFDs:

1. Keep a note of all the processes and external entities. Give unique names to them. Identify the manner in which they interact with each other.

2. Do numbering of processes.

3. Avoid complex DFDs (if possible).

4. The DFD should be internally consistent.

5. Every process should have minimum of one input and one output.

   The data store should contain all the data elements that flow as input and output.

To understand the system functionality, a system model is developed. The developed model is used to analyze the system. The following four factors are of prime concern for system modeling:

1. The system modeling is undertaken with some simplifying assumptions about the system. Though these assumptions limit the quality of system model, it reduces the system complexity and makes understanding easier. Still, the model considers all important, critical and material factors. These assumptions are made regarding all aspects like processes, behaviors, values of inputs etc.

2. The minute details of various components are ignored and a simplified model is developed. For example, there may be various types of data present in the system. The type of data having minute differences are clubbed into single category, thus reducing overall number of data types.

3. The constraints of the system are identified. Some of them may be critical. They are considered in modeling whereas others may be ignored. The constraints may be because of external factors, like processing speed, storage capacity, network features or operating system used.

4. The customers mentioned preferences about technology, tools, interfaces, design, architecture etc. are taken care of.

Example 1: The 0^{th} and 1^{st} levels of DFD of Production Management System are shown in *Figure 2.5 (a) and (b)*

Let us discuss the data flow diagram of Production Management System.

// Copy Figures 2.5 (a), (b)

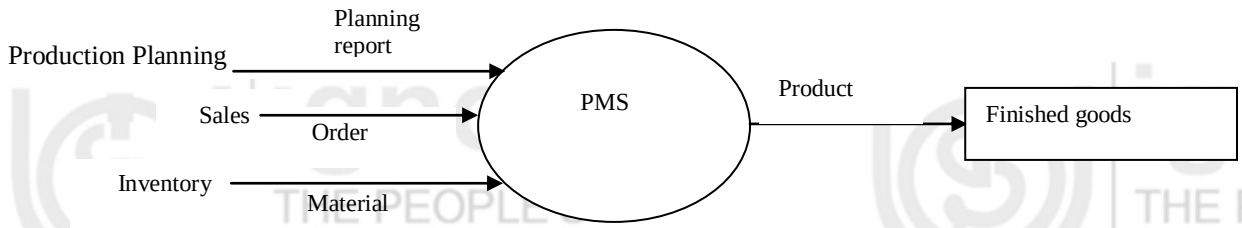


Figure 2.5 (a) : Level 0 DFD of PMS

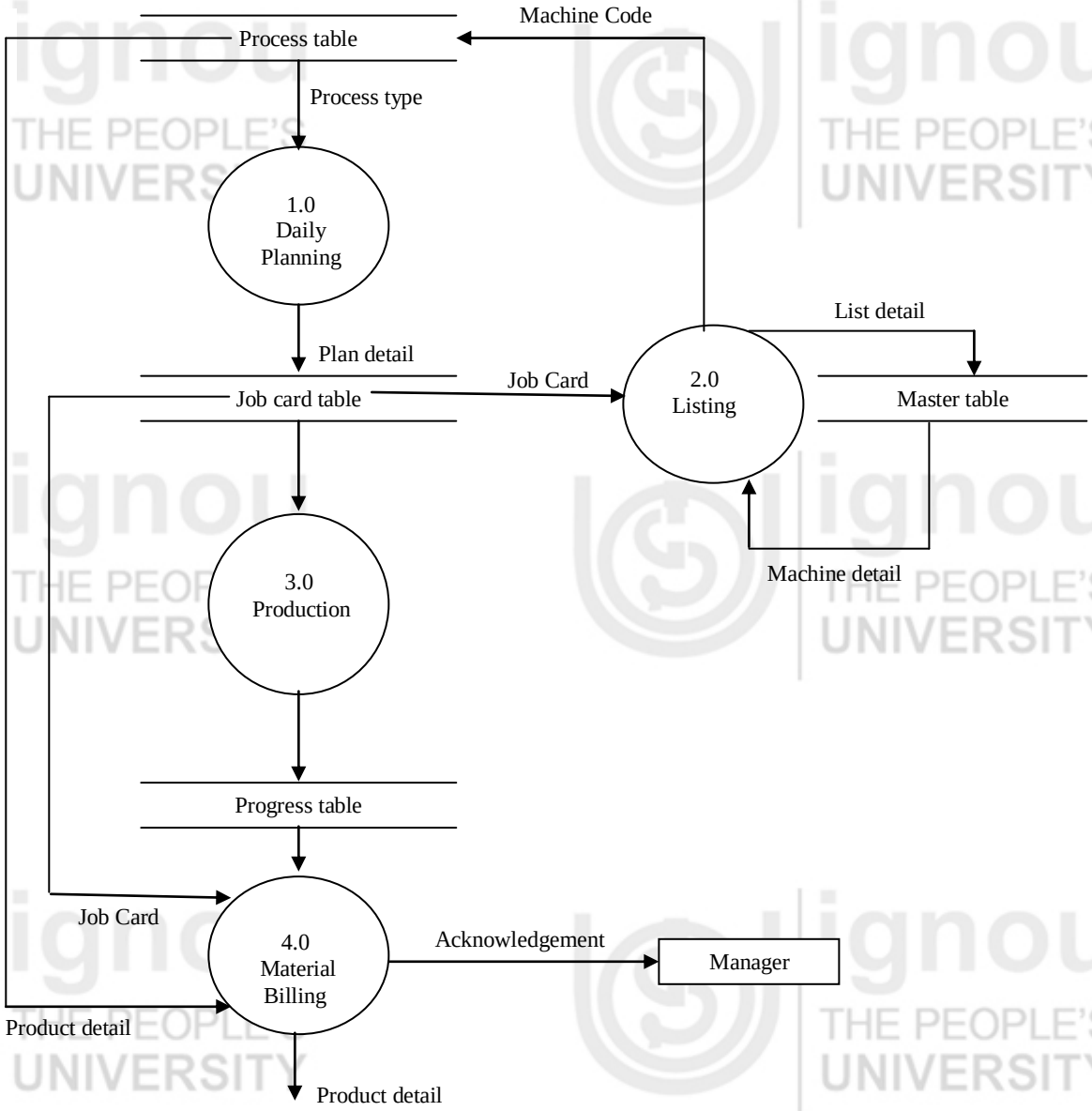


Figure 2.5 (b): Level 1 DFD of PMS

2.3.4 Data Dictionary

This is another tool of requirement analysis which reduces complexity of DFD. A data dictionary is a catalog of all elements of a system. DFD depicts flow of data whereas data dictionary gives details of that information like attribute, type of attribute, size, names of related data items, range of values, data structure definitions etc. The name specifies the name of attribute whose value is collected. For example, fee deposit may be named as FD and course opted may be named as CO.

Related data items captures details of related attributes. Range of values store total possible values of that data. Data structure definition captures the physical structure of data items.

Some of the symbols used in data dictionary are given below:

X= [a/b]	x consists of either data element a or b
X=a	x consists of an optional data element a
X=a+b	x consists of data element a & b.

X=y{a}z	x consists of some occurrences of data elements a which are between y and z.
|	Separator
**	Comments
@	Identifier
( )	Options

Example 2: The data dictionary of payroll may include the following fields:

PAYROLL	=	[Payroll Details]
	=	@ employee name + employee + id number + employee address + Basic Salary + additional allowances
Employee name	=	* name of employee *
Employee id number	=	* Unique identification no. given to every employee*
Basic Salary	=	* Basic pay of employee *
Additional allowances	=	* The other perks on Basic pay *
Employee name	=	First name + Last name
Employee address	=	Address 1 + Address 2 + Address 3

2.3.5 E-R Diagram

Entity-relationship (E-R) diagram is detailed logical representation of data for an organisation. It is data oriented model of a system whereas DFD is a process oriented model. The ER diagram represent data at rest while DFD tracks the motion of data. ERD does not provide any information regarding functionality of data. It has three main components – data entities, their relationships and their associated attributes.

Entity: It is most elementary thing of an organisation about which data is to be maintained. Every entity has unique identity. It is represented by rectangular box with the name of entity written inside (generally in capital letters).

Relationship: Entities are connected to each other by relationships. It indicates how two entities are associated. A diamond notation with name of relationship represents as written inside. Entity types that participate in relationship is called degree of relationship. It can be one to one (or unary), one to many or many to many.

Cardinality & Optionally: The cardinality represents the relationship between two entities. Consider the one to many relationship between two entities – class and student. Here, cardinality of a relationship is the number of instances of entity student that can be associated with each instance of entity class. This is shown in *Figure 2.6.*

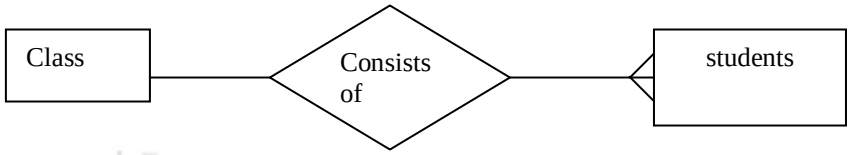


Figure 2.6: One to Many cardinality relationship

The minimum cardinality of a relationship is the minimum number of instances of second entity (student, in this case) with each instance of first entity (class, in this case).

In a situation where there can be no instance of second entity, then, it is called as optional relationship. For example' if a college does not offer a particular course then it will be termed as optional with respect to relationship 'offers'. This relationship is shown in *Figure 2.7*.

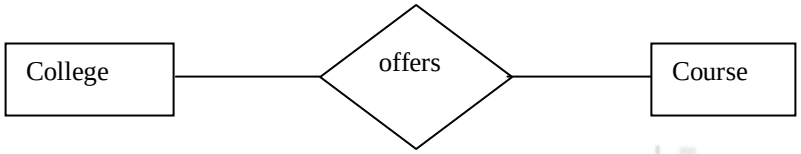


Figure 2.7: Minimum cardinality relationship

When minimum cardinality of a relationship is one, then second entity is called as mandatory participant in the relationship. The maximum cardinality is the maximum number of instances of second entity. Thus, the modified ER diagram is shown in *Figure 2.8.*

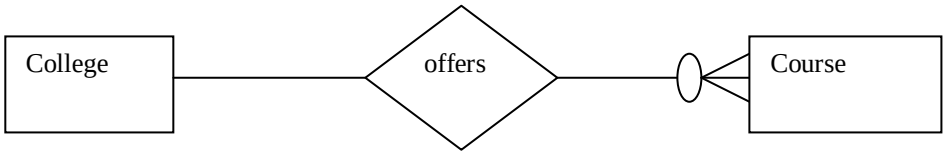


Figure 2.8: Modified ER diagram representing cardinalities

The relationship cardinalities are shown in *Figure 2.9.*

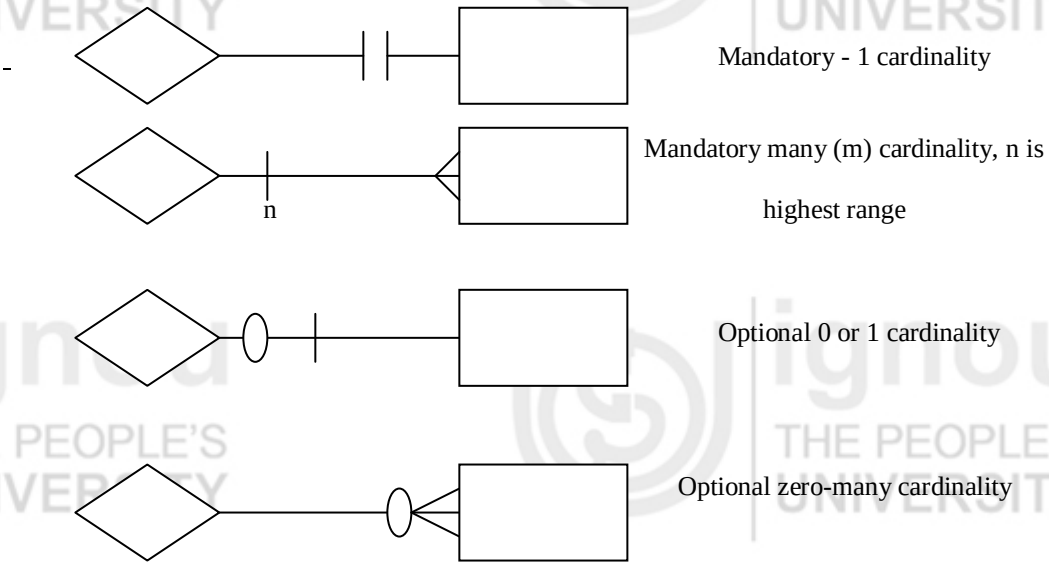


Figure 2.9: Relationship cardinalities

Attributes: Attribute is a property or characteristic of an entity that is of interest to the organisation. It is represented by oval shaped box with name of attribute written inside it. For example, the student entity has attributes as shown in *Figure 2.10.*

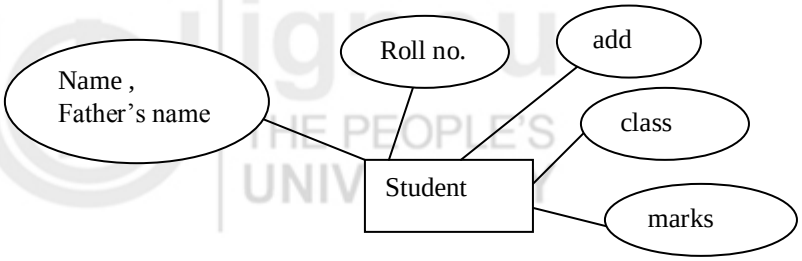


Figure 2.10: Attributes of entity *student*

2.3.6 Structured Requirements Definition

Structured Requirements definition is an approach to perform the study about the complete system and its various sub-systems, the external inputs and outputs, functionality of complete system and its various sub-systems. The following are various steps of it:

Step 1: Make a user level data flow diagram.

This step is meant for gathering requirements with the interaction of employee. In this process, the requirements engineer interviews each individual of organisation in order to learn what each individual gets as input and produces as output. With this gathered data, create separate data flow diagram for every user.

Step 2: Create combined user level data flow diagram.

Create an integrated data flow diagram by merging old data flow diagrams. Remove the inconsistencies if encountered, in this merging process. Finally, an integrated consistent data flow diagram is generated.

Step 3: Generate application level data flow diagram.

Perform data analysis at system's level to define external inputs and outputs.

Step 4: Define various functionalities.

In this step, the functionalities of various sub-systems and the complete system is defined.

☞ Check Your Progress 2

1) What is the advantage of DFD over ER diagram?

……

……

……

……

2) What is the significance specifying functional requirements in SRS document?

……

……

……

……

3) What is the purpose of using software metrics?

……

……

……

2.4 SOFTWARE PROTOTYPING AND SPECIFICATION

Prototyping is a process that enables developer to create a small model of software. The IEEE 610.12 standard defines prototype as a preliminary form or instance of a system that serves as a model for later stages for the final complete version of the system.

A prototype may be categorised as follows:

1. A paper prototype, which is a model depicting human machine interaction in a form that makes the user understand how such interaction, will occur.
2. A working prototype implementing a subset of complete features.
3. An existing program that performs all of the desired functions but additional features are added for improvement.

Prototype is developed so that customers, users and developers can learn more about the problem. Thus, prototype serves as a mechanism for identifying software requirements. It allows the user to explore or criticise the proposed system before developing a full scale system.

2.4.1 Types of Prototype

Broadly, the prototypes are developed using the following two techniques:

Throw away prototype: In this technique, the prototype is discarded once its purpose is fulfilled and the final system is built from scratch. The prototype is built quickly to enable the user to rapidly interact with a working system. As the prototype has to be ultimately discarded, the attention on its speed, implementation aspects, maintainability and fault tolerance is not paid. In requirements defining phase, a less refined set of requirements are hurriedly defined and throw away prototype is constructed to determine the feasibility of requirement, validate the utility of function, uncover missing requirements, and establish utility of user interface. The duration of prototype building should be as less as possible because its advantage exists only if results from its use are available in timely fashion.

Evolutionary Prototype: In this, the prototype is constructed to learn about the software problems and their solutions in successive steps. The prototype is initially developed to satisfy few requirements. Then, gradually, the requirements are added in the same prototype leading to the better understanding of software system. The prototype once developed is used again and again. This process is repeated till all requirements are embedded in this and the complete system is evolved.

According to SOMM [96] the benefits of developing prototype are listed below:

1. Communication gap between software developer and clients may be identified.
2. Missing user requirements may be unearthed.
3. Ambiguous user requirements may be depicted.
4. A small working system is quickly built to demonstrate the feasibility and usefulness of the application to management.

5. It serves as basis for writing the specification of the system.

The sequence of prototyping is shown in following *Figure 2.11.*

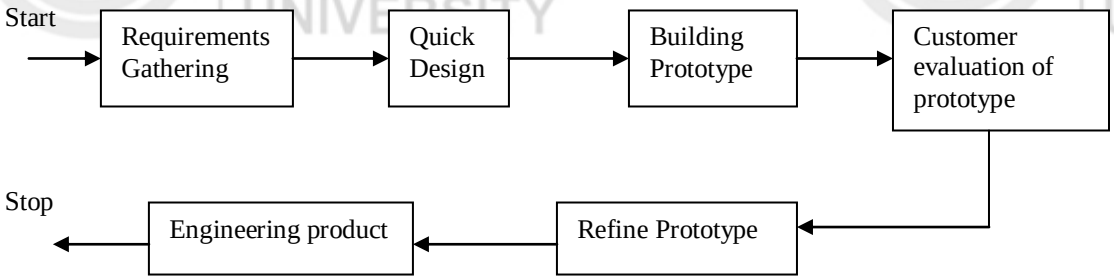


Figure 2.11: Sequence of prototyping

2.4.2 Problems of Prototyping

In some organisations, the prototyping is not as successful as anticipated. A common problem with this approach is that people expect much from insufficient effort. As the requirements are loosely defined, the prototype sometimes gives misleading results about the working of software. Prototyping can have execution inefficiencies and this may be questioned as negative aspect of prototyping. The approach of providing early feedback to user may create the impression on user and user may carry some negative biasing for the completely developed software also.

2.4.3 Advantages of Prototyping

The advantages of prototyping outperform the problems of prototyping. Thus, overall, it is a beneficial approach to develop the prototype. The end user cannot demand fulfilling of incomplete and ambiguous software needs from the developer.

One additional difficulty in adopting this approach is the large investment that exists in software system maintenance. It requires additional planning about the re-engineering of software. Because, it may be possible that by the time the prototype is build and tested, the technology of software development is changed, hence requiring a complete re-engineering of the product.

2.5 SOFTWARE METRICS

Measurement is fundamental to any engineering discipline and software engineering is no exception. Software metric is a quantitative measure derived from the attribute of software development life cycle [Fanton and Kaposi, 1987]. It behaves as software measures.

A software measure is a mapping from a set of objects in the software engineering world into a set of mathematical constructs such as numbers or vectors of numbers.

Using the software metrics, software engineer measures software processes, and the requirements for that process. The software measures are done according to the following parameters:

- The objective of software and problems associated with current activities,
- The cost of software required for relevant planning relative to future projects,
- Testability and maintainability of various processes and products,
- Quality of software attributes like reliability, portability and maintainability,

- Utility of software product,
- User friendliness of a product.

Various characteristics of software measures identified by Basili (1989) are given below:

- **Objects of measurement**: They indicate the products and processes to be measured.
- **Source of measurement**: It indicates who will measure the software. For example, software designer, software tester and software managers.
- **Property of measurement**: It indicates the attribute to be measured like cost of software, reliability, maintainability, size and portability.
- **Context of measurement**: It indicates the environments in which context the software measurements are applied.

Common software measures

There are significant numbers of software measures. The following are a few common software measures:

Size : It indicates the magnitude of software system. It is most commonly used software measure. It is indicative measure of memory requirement, maintenance effort, and development time.

LOC : It represents the number of lines of code (LOC). It is indicative measure of size oriented software measure. There is some standardisation on the methodology of counting of lines. In this, the blank lines and comments are excluded. The multiple statements present in a single line are considered as a single LOC. The lines containing program header and declarations are counted.

☞ Check Your Progress 3

1) Explain rapid prototyping technique.

……

……

……

2) List languages used for prototyping.

……

……

……

SUMMARY

This unit discusses various aspects of software requirements analysis, the significance of use of engineering approach in the software design, various tools for gathering the requirements and specifications of prototypes.

Due to the complexity involved in software development, the engineering approach is being used in software design. Use of engineering approach in the area of requirements analysis evolved the field of Requirements Engineering. Before the actual system design commences, the system architecture is modeled. Entity-relationship (E-R) diagram is detailed logical representation of data for an

organisation. It is data-oriented model of a system whereas DFD is a process oriented model. The ER diagram represent data at rest while DFD tracks the motion of data. ERD does not provide any information regarding functionality of data. It has three main components–data entities, their relationships and their associated attributes. Structured Requirements definition is an approach to perform the study about the complete system and its various subsystems, the external inputs and outputs, functionality of complete system and its various subsystems. Prototyping is a process that enables developer to create a small model of software. The IEEE 610.12 standard defines prototype as a preliminary form or instance of a system that serves as a model for later stages for the final complete version of the system. Measurement is fundamental to any engineering discipline and software engineering is no exception. Software metric is a quantitative measure derived from the attribute of software development life cycle.

2.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) In the present era, the development of software has become very systematic and the specification of requirements is very important. Accordingly, to analyse requirements completely, the engineering approach is used in requirements analysis.

Check Your Progress 2

1) The DFD presents more pictorial representation of flow of data. Moreover, by creating different levels of DFD, the accurate and in depth understanding of data flow is possible.

2) By specifying functional requirements, general understanding about overall working of system is possible.

3) The purpose of using software metrics is to achieve basic objectives of the system with low cost and high quality. Metrics provide scale for quantifying qualities and characteristics of a software. An analogy real life is that meter is the metric of length but to determine the length of cloth, one requires the measuring means like meter-tape etc. Similarly, in software measurement, method must be objective and should produce the same result independent of various measures.

Check Your Progress 3

1) Rapid prototyping techniques are used for speedy prototype development. There are three techniques used for this purpose.

   - Dynamic high level language development
   - Dynamic programming
   - Component and application assembly.

2) High level languages that are used for prototyping are as follows:

   Java, Prolog, Small talk, Lisp etc.

2.8 FURTHER READINGS

1) *Effective Requirements Practices, 2001,* Ralph R. Young; Artech House Technology Management and Professional Development Library.

2) *Software Requirements and Specifications, 1995,* M.Jackson; ACM Press.

3) *Software Architecture in practice, 2003,* Len Bass, Paul Clements, Rick Kazman; Addison-Wesley Professional.

Reference Websites:

http://standards.ieee.org
http://www.rspa.com