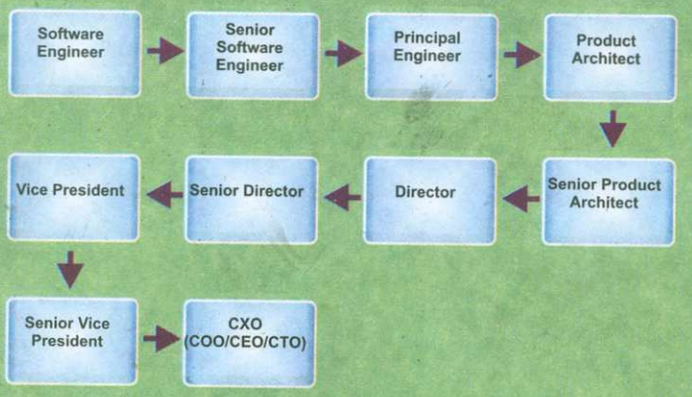
ignou
THE PEOPLE'S
UNIVERSITY

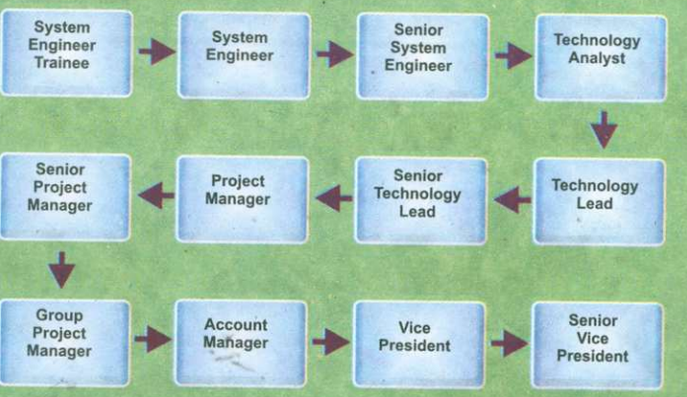
Indira Gandhi National Open University
School of Computer and Information Sciences

BCS-051
Introduction to Software Engineering

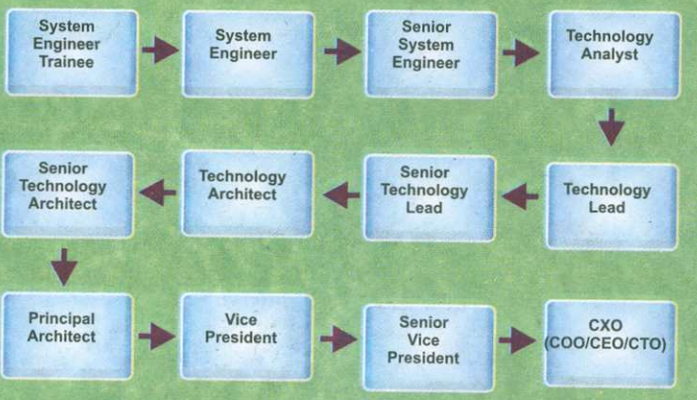
Roles for Technology Stream (Development) in IT Product Development Industry



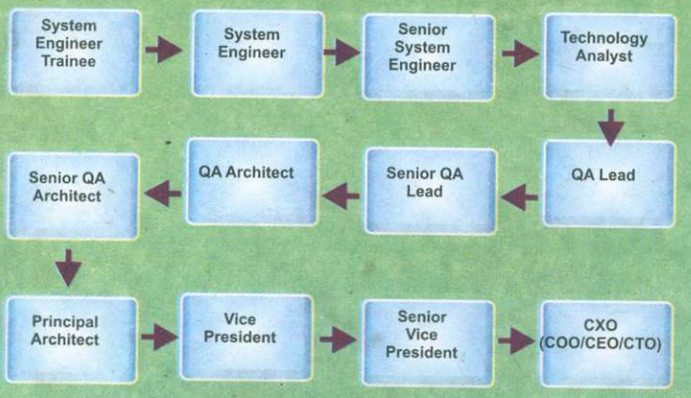
Roles for Project Management Stream (Development) in IT Services Industry



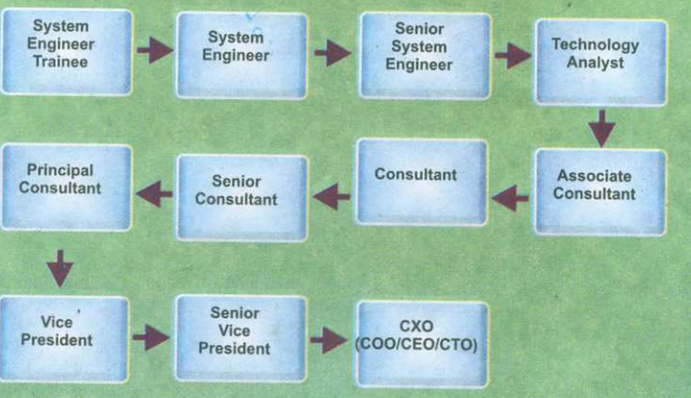
Roles for Technology Stream (Development) in IT Services Industry



Roles for QA Stream (Development) in IT Services Industry



Roles for Consulting Stream in IT Services Industry



Development of SRS

"शिक्षा मानव को बन्धनों से मुक्त करती है और आज के युग में तो यह लोकतंत्र की भावना का आधार भी है। जन्म तथा अन्य कारणों से उत्पन्न जाति एंव वर्तमान विषमताओं को दूर करते हुए मनुष्य को इन सबसे उपर उठाती है।"

– इन्दिरा गांधी

"Education is a liberating force, and in our age it is also a democratising force, cutting across the barriers of caste and class, smoothing out inequalities imposed by birth and other circumstances."

- Indira Gandhi

Indira Gandhi National Open University
School of Computer and Information Sciences

BCS-051
INTRODUCTION TO SOFTWARE ENGINEERING

Block

1

DEVELOPMENT OF SRS

UNIT 1
Characteristics of SRS 5

UNIT 2
Function Oriented Modeling 19

UNIT 3
Object Oriented Modeling 29

PROGRAMME DESIGN COMMITTEE

Prof. Manohar Lal
SOCIS, IGNOU, New Delhi

Prof. H. M. Gupta
Dept. of Elect. Engg.
IIT, Delhi

Prof. M. N. Doja, Dept. of CE
Jamia Millia, New Delhi

Prof. C. Pandurangan
Dept. of CSE, IIT, Chennai

Prof. I. Ramesh Babu
Dept. of CSE
Acharya Nagarjuna University
Nagarjuna Nagar (AP)

Prof. N. S. Gill
Dept. of CS, MDU, Rohtak

Prof. Arvind Kalia, Dept. of CS
HP University, Shimla

Prof. Anju Sahgal Gupta
SOH, IGNOU, New Delhi

Prof. Sujatha Verma
SOS, IGNOU, New Delhi

Prof. V. Sundarapandian
IITMK, Trivandrum

Prof. Dharamendra Kumar
Dept. of CS, GJU, Hissar

Prof. Vikram Singh
Dept. of CS, CDLU, Sirsa

Sh. Shashi Bhushan
Associate. Prof.
SOCIS, IGNOU, New Delhi

Sh. Akshay Kumar
Associate Prof.
SOCIS, IGNOU, New Delhi

Dr. P. K. Mishra, Associate Prof.
Dept. of CS, BHU, Varanasi

Dr. P.V . Suresh, Associate Prof.
SOCIS, IGNOU, New Delhi

Sh. V. V. Subrahmanyam
Associate Prof.
SOCIS, IGNOU, New Delhi

Sh. M. P. Mishra, Asst. Prof.
SOCIS, IGNOU, New Delhi

Dr. Naveen Kumar, Reader
SOCIS, IGNOU, New Delhi

Dr. Subodh Kesharwani, Asst. Prof.
SOMS, IGNOU, New Delhi

COURSE CURRICULUM DESIGN COMMITTEE

Sh. Shashi Bhushan
SOCIS, IGNOU, New Delhi

Dr. P. V. Suresh
SOCIS, IGNOU, New Delhi

Sh. V. V. Subrahmanyam
SOCIS, IGNOU, New Delhi

Dr. Naveen Kumar
SOCIS, IGNOU, New Delhi

Prof. A. K.Tripathi

Dept. of CSE, IIT(BHU),
Varanasi

Dr. S. R. N. Reddy

Dept. of CSE,

IGDTUW,

New Delhi

Sh. Akshay Kumar
SOCIS, IGNOU, New Delhi

Prof.Manu Sood,

Dept. of CS

HP University
Shimla

Sh. M. P. Mishra
SOCIS, IGNOU, New Delhi

SOCIS FACULTY

Sh. Shashi Bhushan, Director

Sh. Akshay Kumar
Associate Professor

Sh. M. P. Mishra
Asst. Professor

Prof. Manohar Lal

Sh. V. V. Subrahmanyam
Associate Professor

Dr. Sudhansh Sharma
Asst. Professor

Dr. P. V. Suresh
Associate Professor

Dr. Naveen Kumar
Reader

PREPARATION TEAM

Content Editor:

Dr. S. R. N. Reddy

Dept. of CSE

IGDTUW

New Delhi

Language Editor:

Sh. P. Lakshmi Kantham
(Retd.), Tenali

Block Writers:

Units 1 and 2:

Sh. Shailesh Kumar Shiva Kumar, Research Scholar (SOCIS, IGNOU) and Senior Technology Architect (Infosys Technologies)

Unit 3

Sh. Suyash Kumar

Research Scholar (SOCIS, IGNOU) and Faculty (Krishna Engineering College, Ghaziabad)

Course Coordinator: Dr. P. V. Suresh, SOCIS, IGNOU, New Delhi

Print Production

Mr. S. Burman	Mr. Tilak Raj	Mr. Yash Pal
Dy. Registrar (Publication)	Asstt. Registrar (Publication)	Section Officer (Publication)
MPDD, IGNOU, New Delhi	MPPD, IGNOU, New Delhi	MPDD, IGNOU New Delhi

January 2018 (Reprint)

ISBN-978-81-266-6610-2

Further information about the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110 068.

Printed and published on behalf of the Indira Gandhi National Open University, by Registrar MPDD, IGNOU, New Delhi

Printed at: Ankur Offset Pvt. Ltd., A-54, Sector-63, Noida-201307 (U.P.)

COURSE INTRODUCTION

This course introduces the learner to the subject of Software Engineering.

Software Requirements Specification (SRS) is a very important document which is developed in the phase of analysis of Software Development Life Cycle (SDLC). Though there are standard SRS templates recommended by IEEE and other bodies, every company can have its own SRS template depending on various factors such as domain of the project at hand and other issues. The success of a Software project depends on almost all the activities that are part of its development. The flow of data is depicted by Data Flow Diagrams (DFD). Entity Relationship Diagrams (ERD) will contribute to development of database structure. Data Dictionaries will provide the data about data. Unified Modeling Language (UML) enables the creation of visual models for software development projects that are based on the object oriented concepts.

Design is the activity after analysis. The solution to the problem using technical jargon is performed in this phase. Topics such as Cohesion, Coupling are very much important for successful function oriented design (FOD). In Object Oriented Design (OOD), identification of objects is a very important activity. As it was mentioned earlier, almost every activity is critical and will contribute to the ultimate success of the Software project. After the software is developed as per the prescribed process, testing of Software is an important activity in which the developer as well as customer will also get involved depending on the type of testing to be performed. Different testing techniques such as White Box Testing, Black Box Testing, etc. are present. Testing with right data is very important as the Software need not behave in the same way for any data. So, creation of test cases is critical for proper testing of software.

There are different paradigms for Software development such as Waterfall Model, Spiral Model, etc. Based on the software project at hand, appropriate SDLC model is chosen. Management of large scale software project is complex and principles of Software Project Management (SPM) are to be followed. As part of SPM, planning, execution, etc. as well as scheduling issues are dealt with. A request for change after the commencement of a Software project is unwelcome though it needs to be accepted. When accepted, managing the change properly is important as it should not disturb the work already done and will also lead to software that performs as per the set objectives.

This course consists of three blocks:

Block 1 deals with SRS development. It covers topics related to function oriented modeling, object oriented modeling as well as characteristics of SRS.

Block 2 deals with Design and Testing. It covers topics related to function oriented design, object oriented design, testing techniques as well as development and execution of test cases.

Block 3 deals with some more concepts associated with Software Engineering. It includes SDLC models, SPM concepts and other topics such as change management, etc.

BLOCK INTRODUCTION

This block introduces learner to Software Requirements Specification (SRS) development, Function Oriented Modeling and Object Oriented Modeling.

SRS is a very important document in the process of software development. Its a document developed as part of first phase of Software Development Life Cycle (SDLC). It is the document which includes all requirements that need to be fulfilled by the software that is under development. Any lack of seriousness while preparing SRS will reflect in the end product. To be straight, if SRS does not cover all requirements or if there is any error in the information in SRS, the software that is developed will not fulfil all requirements or the software that is developed will not behave the way it is expected. So, developing correct SRS is very important for the success of the software project. In this block, the characteristics of SRS are explained. SRS templates are introduced. The most important SRS template is IEEE SRS. Whenever you need to prepare SRS, you are advised to use IEEE Standard SRS template unless specified.

Function Oriented Modeling includes Data Flow Diagrams (DFDs), Entity Relationship Diagrams (ERDs), Structure Chart and Data Dictionaries. Function oriented modeling is the visual representation of functions and processes within a system. It provides a structured way to describe the system behaviour and functionality. Another key feature of functional model is that they describe the behaviour of activity/action/system without specifying how they are implemented. Computation intensive modules such as compilers and database functions are ideal candidates for being modeled in function oriented modeling. Function oriented modeling helps us to define the functions, processes, activities in a more structured way. It helps to model the function hierarchy in intuitive visual format.

Object Oriented Modeling includes introduction to Unified Modeling Language (UML), Use Case Diagrams and Class Diagrams. To build a model in software engineering, we may use several approaches. Two commonly used approaches are an algorithmic prospective and an object oriented perspective. In algorithmic perspective, the main building block of all software is the procedure or function.0020This approach leads developers to focus on the issues of control and the decomposition of larger algorithms into smaller algorithms. In object oriented approach, the main building block of all software systems is the object or class. Every object has identity, state and behaviour.

This block consists of three units and is organized as follows:

Unit 1 deals with Characteristics of SRS. It includes explanation of various components of SRS as well as IEEE SRS template outline.

Unit 2 deals with Function Oriented Modeling. It covers concepts as well as other issues associated with DFDs, ERDs, Structure Chart and Data Dictionaries.

Unit 3 deals with Object Oriented Modeling. It covers concepts as well as other issues associated with UML, Use Case Diagrams and Class Diagrams.

UNIT 1 CHARACTERISTICS OF SRS

Structure

1.0 Introduction

1.1 Objectives

1.2 Core Concepts of SRS

- 1.2.1 Benefits of SRS
- 1.2.2 Key Concerns of SRS
- 1.2.3 SRS Characteristics
- 1.2.4 Impact of SRS
- 1.2.5 Best Practices of Writing a Good Quality SRS
- 1.2.6 Deep Dive into SRS Structure

1.3 SRS Standards

- 1.3.1 IEEE Standards 830 for SRS
- 1.3.2 Department of Defense DI-MCCR-80025 A Standard for SRS

1.4 Case Study of SRS Development

- 1.4.1 Business Problem Statement
- 1.4.2 SRS for ZYZ Online

1.5 Summary

1.6 Answers to Check Your Progress

1.7 Further Readings

1.0 INTRODUCTION

This unit explains the overall concepts of Software Requirements Specification (SRS) including the standard structure of SRS document and a small case study.

Definition : *A Software Requirements Specification (SRS) is a specification of the software system which provides complete and structured description of the system's requirements, behavior, interfaces including all functional use cases and non-functional requirements.* Functional use cases provide detailed description of various functionalities that needs to be supported by the software system from user's point-of-view and non-functional requirements specify the requirements for performance, scalability, availability, accessibility, portability, etc.

At a high level, SRS is essentially a contract document which translates business and user processes and models into an exact format understood by technical stakeholders of the project. It encompasses various sections and sub-sections to provide sufficient coverage to touch base all important aspects of project implementation. Once the SRS document is signed-off and frozen during requirements elaboration phase of the project, subsequent activities such as detail design and implementation would start.

Organizations create SRS documents to provide the complete requirements for vendors to implement the software system or in case of in-house development, SRS captures user requirements in a structured fashion to aid the software implementation.

SRS being a formal document communicates all detailed requirements from end-user/customer to the software development team. It covers various kinds of requirements including functional, operational, resources, interface, documentation, safety, etc.

The fundamental difference between a system specification and a software specification is that a system specification provides details about the underlying hardware of the system. This includes things like system functions, system drawings/instructions, system interface details, hardware safety requirements etc. whereas the software specification is focused mainly on the functionality of software like functional use cases, performance requirements of the software, etc.

1.1 OBJECTIVES

After going through this unit, you should be able to

- understand core concepts of SRS,
- understand the benefits, applications and importance of SRS,
- understand IEEE standard SRS template, and
- learn to convert a problem statement into SRS with an example of a simple case study.

1.2 CORE CONCEPTS OF SRS

In this section we will examine the key concepts related to SRS including its benefits, the concerns it addresses in a project, its main characteristics and the impact of SRS on a software project.

1.2.1 Benefits of SRS

IEEE 830 standard for SRS mentions following benefits of SRS:

- **Forms the basis of agreement between customers and suppliers about the software functionality:** SRS serves as a structured contract between these parties specifying all functionalities along with constraints and mentions the behavior of the intended software. End user/customer can verify if the intended software meets all the needs and requirements stated in user requirements document.
- **Optimizes development effort:** As the requirements are fully specified beforehand, the implementation team can design the system accurately thereby reducing the effort in re-design, re-work, re-testing and defect fixing.
- **Forms basis for cost and schedule estimation:** Using the functional and non-functional requirements specified in SRS, the project management team can estimate the overall project cost and schedule in more accurate fashion and make informed decisions about risk identification and mitigation.
- **Forms basis for verification and validation:** Quality team can design the validation and testing strategy including various kinds of test cases based on the requirements specified in SRS.
- **Helps software portability and installation:** The software usability information contained in SRS helps to transfer the software across various locations including multiple inter-company departments and other external customers.
- **Helps in enhancement:** As SRS specifies each requirement in fullest details, it would be easier to assess the impact of any enhancement planned providing the cost and schedule estimate of the enhancement.

1.2.2 Key Concerns of SRS

As per IEEE 830 standard, SRS should address the following key concerns:

- **Functionality:** Complete details of the software.

- **External Interfaces:** Details of how the software interacts with external systems, and end users.
- **Performance:** Provides details of transaction speed, software availability, response time, failover conditions, disaster recovery scenarios, etc.
- **Attributes:** Provides details about portability, correctness, maintainability, security, extensibility, flexibility, etc.
- **Constraints:** All applicable architecture and design constraints including the maximum load supported, supported browsers, JavaScript dependency and others should be detailed.

1.2.3 SRS Characteristics

IEEE specifies that a good SRS should possess following characteristics :

- **Correct:** SRS should specify the functionality correctly from all aspects. It also be continually updated to reflect all the software updates and enhancements.
- **Unambiguous:** As SRS is written in natural language, it is possible for it to be interpreted in multiple ways based on the context, cultural background etc. So, SRS should consider these things and define and refine in most unambiguous fashion as possible. This would include providing references, elaborating any abstract requirement with example scenarios etc. It is a good practice to get a proof read of SRS by another person to weed out any ambiguous descriptions.
- **Precise:** The description should not contain fizzy words so as to make it precise.
- **Complete:** SRS should provide all the details required by software designers for design and implementation of the intended software.
- **Consistent:** The terminologies, definitions and others used throughout the SRS should be consistent. It is a good practice to pre-define all definitions, abbreviations and refer them consistently throughout SRS.
- **Verifiable:** This supplements the unambiguous characteristic. All requirements should be quantified with exact and verifiable numbers. For instance "The home page should load quickly" is non-verifiable as "quickly" is subjective; it is also not mentioned if the page should load quickly across all geographies. Instead of these subjective terms the requirement should quantify it with the exact response time: "The home page should load within 2 seconds in North America region".
- **Modifiable:** The requirements should be detailed only once throughout the document so that it is easy to modify and maintain the document in long run. To ensure that SRS is modifiable it should:
  - Be coherent, well-organized and contain cross-referencing
  - Avoid redundancy
  - State each requirement separately
- **Traceable:** SRS should map the requirements to other business/user requirement documents so that it is possible to trace the requirements. It should also support backward-traceability and forward traceability.
- **Ranked for importance/stability:** The requirements should be ranked based on its deemed business/user importance. Ranking is done based on:
  - **Degree of stability:** Stability is related to number of changes required for implementing functionality.
  - **Degree of importance:** In this case, the requirements are classified into categories such as essential, conditional and optional.

Few other characteristics of a good SRS are that it should be **understandable** by people of varied backgrounds and it should be **design independent.** That is, without favoring any particular design.

1.2.4 Impact of SRS

We have seen the positive benefits of SRS in previous section. Let us look at a scenario wherein the SRS is not properly defined and its impact on the project. This will enable us to understand the importance of SRS on the project:

- **Impact on cost and schedule:** Without a complete and accurate SRS, it would be difficult to properly estimate and plan the overall cost of the project. This would have ripple effect on resource staffing, milestone planning and overall project budget. As a result the entire project schedule will be in jeopardy.
- **Quality Impact:** Incomplete requirements specification would manifest itself into incomplete test plan and impacts the quality of all project deliverables. This negatively impacts the project by re-testing, re-coding and re-design efforts leading to cost and effort overruns.
- **Impact on overall customer/user satisfaction:** An improperly translated user requirements would damage the customer confidence on the software product and reduces the usability and overall satisfaction index.
- **Impact on maintenance:** Without proper traceability, it would be difficult to extend the software, enhance it and to fix the issues.

1.2.5 Best Practices of Writing a Good Quality SRS

The following factors should be comprehended while authoring a high quality SRS:

- **Nature of SRS**: The document should follow the standard structure to ensure that it is understood and interpreted by all stakeholders.
- **Environment of SRS**: Background of SRS authors and the intended audience should be comprehended to ensure that language, terms are interpreted consistently.
- **Characteristics of SRS**: It should adhere to all the aforementioned characteristics.
- **Joint preparation of SRS**: Sometimes multiple teams would contribute to SRS. In such scenarios there should be properly defined procedures for updates and consistency checks. There should be an independent review to ensure that SRS is coherent and consistent.
- **SRS Evolution**: The document should be continuously updated along with system evolution.

☞ **Check Your Progress 1**

(1) The IEEE standard related to SRS is .....

(2) The characteristic of SRS which requires it to be specified with only one possible interpretation is .....

(3) The key impact of SRS on overall project is related to .....

1.2.6 Deep Dive into SRS Structure

Let us have a closer look at each of the section and subsection in the IEEE 830 standard for SRS and later develop a sample SRS for a problem statement.

Section 1: Introduction

The first section of IEEE standard-based SRS document broadly provides the background and lays the ground work for the software. It contains the following sub-sections:

- **Purpose**: This explains the main purpose and the intended audience of the SRS.
- **Scope**: This sub-section provides pointed brief description of high-level functionality of the software and explains the broad goals of the intended software.
- **Definition, Acronym/Abbreviation**: It defines all the main terms and provides the acronyms used in throughout the document.
- **References**: All supporting documents referred such as business process document or user requirement document will be mentioned here.
- **Overview**: It provides the organization of the entire SRS document and helps in readability.

Section 2: General Description

This section elaborates the functionality of the intended software in finer details containing following sub-sections:

- **Product Perspective**: This sub-section describes the product interfaces and relationships with other products. It also provides the operational details and business ownership for the products.
- **Product Functions**: In this sub-section all the major functionality of the software product is described in consistent fashion. This includes the functional use cases and the functionality is normally listed with "shall". It defines various actions performed by the software while accepting the input for producing the expected output. The functionality/action would include input validity checks, operations sequence, expected response and error handling, processing formula for input to output conversion.
- **User Characteristics**: Here, we describe the characteristics of intended users of the software including their background, training requirements, demographics details, technical expertise, educational background, language requirements, etc.
- **General Constraints**: This provides the list of constraints like performance constraints, memory constraints, load constraints, etc.
- **Assumptions and dependencies**: This provides the list of all assumptions including the assumptions related to OS, browser, hardware, implied features, security, performance etc. and all the dependencies required for software including the dependency on upstream services, hardware and other resources.

Section 3: Specific Requirements

This section provides in-depth details about the functional and non-functional requirements. Project teams can customize the section to include any project/domain specific requirements.

- **External interface requirements** describe about the contracts of integration systems like web services, database, ERP, legacy systems etc. This interface specification provides the contract for intended software and interfaces in structured way.
- **Functional requirements** provide detailed functional use cases, business functions and the behavior of the intended software. Requirements are prioritized based on their business importance.

- **Performance requirements** detail all the expected performance requirements of the system. This include response time, process completion time, transaction time, etc.
- **Design constraints** provide all the constraints for designing the software.
- **Standards compliance** states all the regulations and standards that the software needs to comply. This also depends on the domain area of software as well as regulations stipulated by law. For instance this would involve privacy policy, accessibility standards, data archival policy, auditing policy, export policy, Intellectual property policies, etc.
- **Logical database requirement** provides high-level database related functionalities.
- **Software system attributes** provide details about critical non-functional requirements:
    - Reliability elaborates fault tolerance and error handling scenarios and the expected reliability requirements from the system in those scenarios.
    - Availability elaborates the system availability percentage (normally 99.999% or five nines), system recovery, disaster recovery and restart.
    - Security provides all the security requirements related to software functionality such as data encryption/decryption, password policy, transport level security, data integrity constraints, authentication and authorization, etc.
    - Maintainability explains the expectation of the system during upgrades, patches, etc.

Appendixes Section

This section provides details of references used within main sections. Normally this includes glossary, mock up screens, reports, data dictionary, business model etc.

1.3 SRS STANDARDS

Following are the two main standards for SRS. First one is from Institute of Electrical and Electronic Engineer (IEEE) which is most widely used and adopted in industries. It gives the outline and sections of SRS and it also provides provision for specifying the requirements specific for a particular project in section 3. Second one is a military standard which is adopted by US department of defense (DoD).

1.3.1 IEEE Standard 830 for SRS

Various sections and sub-sections of SRS as per IEEE standard 830-1998 are outlined below (Courtesty: IEEE):

1. Introduction
    - 1.1. Purpose of the Software Requirement Specification
    - 1.2. Scope of the Product
    - 1.3. Definitions, Acronyms and Abbreviations
    - 1.4. References
    - 1.5. Overview of the Rest of the Software Requirement Specification
2. General Description
    - 2.1. Product Perspective
        - 2.1.1. System Interface

2.1.2. User Interface

2.1.3. Hardware Interface

2.1.4. Software Interface

2.1.5. Communication Interface

2.1.6. Operations

2.2. Product Functions

2.3. User Characteristics

2.4. General Constraints

2.5. Assumptions and Dependencies

3. Specific Requirements

3.1. External interface requirements

3.2. Functional requirements

3.3. Performance requirements

3.4. Design constraints

3.4.1. Standards compliance

3.5. Logical database requirement

3.6. Software system attributes

3.6.1. Reliability

3.6.2. Availability

3.6.3. Security

3.6.4. Maintainability

Appendices

This standard allows industries to customize the SRS structure to fit their requirements.

Complete SRS template can be downloaded from
http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf

1.3.2 Department of Defense DI-MCCR-80025A Standard for SRS

This standard is mainly used by software projects executed for US department of defense. The standard is very rigid and focuses on safety, quality, interfaces and handling manual errors which is relevant for military projects. The SRS template for this standard is outlined below (courtesy:

http://segoldmine.ppi-int.com/documents/di-mccr-80025a-software-requirements-specification-srs)

1. Scope

1.1. Identification

1.2. CSCI overview

1.3. Document overview

2. Applicable documents.

2.1. Government documents

2.2. Non-Government documents

3. Engineering requirements
   - 3.1. CSCI external interface requirements
   - 3.2. CSCI capability requirements
   - 3.3. X Capability X
   - 3.4. CSCI internal interfaces
   - 3.5. CSCI data element requirements
   - 3.6. Adaption requirements
     - 3.6.1. Installation dependent data
     - 3.6.2. Operational parameters
   - 3.7. Sizing and timing requirements
   - 3.8. Safety requirements
   - 3.9. Security requirements
   - 3.10. Design constraints
   - 3.11. Software quality factors
   - 3.12. Human performance/human engineering requirements
     - 3.12.1. Human information processing
     - 3.12.2. Foreseeable human errors
     - 3.12.3. Total system implications (e.g. training, support, operational environment)
   - 3.13. Requirements traceability
4. Qualification requirements
   - 4.1. Methods
   - 4.2. Special
5. Preparation for delivery
6. Notes

Appendices

1.4 CASE STUDY OF SRS DEVELOPMENT

Let us take a sample problem statement and develop an IEEE standard-based SRS by applying the concepts explained in previous section.

1.4.1 Business Problem Statement

XYZO (XYZ Online) rental has done a recent survey of their market competition and found that its competitors are heavily using the online channel to supplement their in-store sales. The survey also indicated that online platform better suits XYZ rental's long-term strategy of expanding the user base across various locations and improves customer loyalty through targeted campaigns and promotions.

Hence XYZ Corporation decided to develop an online commerce platform in iterative fashion. The first iteration allows customers to search for a product to view the product details and pricing, allows the product to add to shopping cart and to perform a check out. In future iterations, there are plans to enhance this by adding mobile specific applications, personalized recommendations and other commerce features.

1.4.2 SRS for XYZ Online

Following SRS is drafted based on above problem statement. IEEE 830 standard-based SRS is used.

1. Introduction

1.1 Purpose

The purpose of the SRS document is to define the system under development, namely the XYZ online (XYZO). The intended audience of this document include the site administrator of the XYZ and the end users of the XYZO. Other intended

audience include development team business analysts, design team, enterprise architect, testing team, infrastructure team and system team

1.2 Scope

XYZO was envisioned by executive management as part of their overall sales strategy after discovering that there is a huge scope for increasing sales by selling the products online. Key goals of XYZO are:

a) Increase the sales-per-visit from regular customer. Currently this is at $20 at brick-and-mortar stores. XYZO should increase it to $25 within 1 year

b) Increase customer satisfaction from 40% to 60% within 1 year

c) Increase customer loyalty from 10% to 15% within 1 year

Following high-level functionality is in-scope of this project:

a) Develop XYZO online platform for end customers

b) XYZO should support product search, shopping cart and product check out

1.3 Definitions and Acronyms

Definitions

Term	Definition
User	Online users or customers of XYZO
Site admin	Administrator who maintains/enhances XYZO
Stakeholder	People who has business and other interests in XYZO including sales team, IT team, System team

Acronyms :

- ROI: Return on Investment
- SDLC: Software Development Life Cycle
- MSIE: Microsoft Internet Explorer

1.4 Overview

The document is organized as follows: Second section mainly provides the interfaces and third section provides details about important functional and non-functional requirements/use-cases.

2 General Description

2.1 Product Perspective: The system mainly consists of XYZ Online system containing application server and web server. XYZ Online interacts with Pricing system to get the product pricing information and Inventory ERP system to get the product availability information. Product catalog and hierarchy information is stored in product database.

High level block diagram of the system is shown in Figure 1.1.

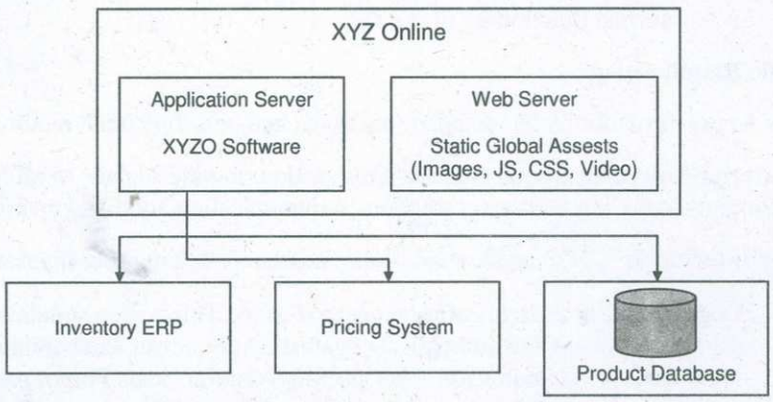


Figure 1.1: Block Diagram of XYZ Online

2.2 Product Functions: XYZO mainly supports the following high level functions:

- **Search:** Users will be able to search using various product attributes. System would display the products matching their criteria.
- **Shopping Cart:** System allows users to add the products from search results page into shopping cart.
- **Checkout:** Users can check out their products from shopping cart after providing shipping information and completing a successful payment.

2.3 User characteristics: There are mainly two kinds of users of XYZO:

- **Online users:** These users are XYZO customers who will access the system online. The users are from US region for first iteration and the preferred language is English. Users have minimal technical knowledge and hence need intuitive navigation aids and simple page layouts.
- **Site administrator:** Administrator is responsible for maintaining the XYZO system and will be involved in software fixes, deployment and regular maintenance.

2.4 Constraints

Following are the main constraints

- Initial release only allows maximum 10,000 products in search results due to restrictions imposed by pricing and inventory system.
- Initial release is only supported on MSIE, Google Chrome and Firefox browsers.
- XYZO web application does not work without JavaScript support.

2.5 Assumptions and Dependencies

The following are main assumptions:

- The IT department of XYZ has all the available hardware required to support the intended user load.
- All the upstream systems including the product database, pricing and inventory system provides the response within agreed SLAs to ensure the optimal performance.

XYZO has following key dependencies:

- XYZO depends on Pricing system, Inventory ERP and product database and hence these systems should be up and available for normal functioning of XZYO.

3 Specific Requirements

This section provides all the detailed functional and non-functional requirements.

3.1 External Interface Requirements: This section provides details of all inputs and outputs including hardware, software, communication and mockup prototype.

3.1.1 User Interfaces: XYZO application should contain following user interfaces:

- **Login page** for authenticating registered users. This screen should accept user id, password and authenticate against corporate authentication system. It also provides features for "New user registration" and "Forgot password" and "Forgot user id".

- **Product search page** where registered user can search product based on product attributes. User can search by product name, brief description, and product category and product id. Search should support intuitive features such as type-ahead, synonym support, categorized results grouping and spell correction.
- **Search results page** displays the results of search operation. The results should be paginated with configurable page size. It should allow the user to add the product to shopping cart.
- **Shopping cart page** displays the existing items in shopping cart along with total amount and allows the user to check out.
- **Checkout page** allows the user to purchase the products using credit/debit card or using PayPal account. It should be integrated with payment gateway and display the order details after successful payment. A confirmation email should be sent to registered email after successful completion.

3.1.2 Hardware Interface: There is no direct hardware interface specifically for XYZO application. The web application runs on an application server hosted in-house on enterprise hardware.

3.1.3 Software Interface: XYZO application should integrate with following interfaces:

- **Product database** to get product details. JDBC APIs are the most preferred way of integration.
- **Pricing System** to get the product pricing in real-time for the selected products. Integration should be done using web services.
- **Inventory ERP** to get the product availability information. Integration should be done using web services.

3.1.4 Communication Interface: There are no XYZO specific communication interface requirements. Existing OS and network infrastructure will be leveraged for communication.

3.2 Functional Requirements: This section provides details of all major functionalities supported by XYZO system.

3.2.1 Functional Requirement 1.1: Login Operation

Functional requirement Id	FR1
Requirement Title	Login Operation
Requirement Description	• Registered user enter user id and password. • User clicks on "Login" button. • System should authenticate and create a valid user session upon successful authentication and redirect user to product search page.
Business Rationale	Allow registered users to buy products and provide personalization features later.
Exception Scenarios	• If authentication fails, user should be redirected to error page showing "User id or password incorrect" message. Maximum password retries allowed are 3 after which the account should be temporarily locked and request user to contact customer support. • The operation should also support first time user registration and forgot user id/password.
Dependencies	• Authentication service of corporate LDAP.

3.2.2. Functional Requirement 1.2: Search Operation

Functional requirement Id	FR2
Requirement Title	Search Operation
Requirement Description	• User enters any of these product attributes: Product name, product description, product keyword and clicks on "search". • Search function should retrieve the matching records from product database. • Search results should be displayed in pagination format with configurable page size. Default page size should be 10. • It should allow user to add the product(s) to shopping cart.
Business Rationale	Help users to discover the product. This will be supplemented by product recommendation to boost sales value in future.
Exception Scenarios	• Display "No products found" if no products match the criteria.
Dependencies	• FR1: Users should be logged in.

3.2.3. Functional Requirement 1.3: Shopping Cart Operation

Functional requirement Id	FR3
Requirement Title	Shopping Cart Operation
Requirement Description	• The shopping cart function should display all the products added by user to the shopping cart. • While displaying the product, it should get the product availability from inventory system and product price from pricing system to display the availability and pricing information • It should allow the user to check out the product(s) in shopping cart.
Business Rationale	Provide the online users with a seamless shopping experience.
Exception Scenarios	• If the selected quantity of products is not available, system should not allow the user to check out that product. It should display appropriate error message
Dependencies	• FR1: Users should be logged in • FR2: Users should have selected non-zero products from search operation

3.2.4. Functional Requirement 1.4: Checkout Operation

Functional requirement Id	FR4
Requirement Title	Checkout Operation
Requirement Description	• The checkout function should display the final amount of products in shopping cart and the shipping information. • It should support the user to enter credit/debit card information to complete the transaction. Additionally, any other online payment schemes should also be supported. • Post successful payment, it should display the order information and send out a detailed mail to registered email account.
Business Rationale	Provide flexible options for users to purchase the product. More purchase options like "cash on delivery" will be added in future

Exception Scenarios	• If the payment gateway is down, system should display appropriate error message • If the credit/debit card account declines the transaction, appropriate error message should be displayed and allow the user to retry the transaction. Maximum retries allowed are three.
Dependencies	• FR1: Users should be logged in. • FR2: Users should have selected non-zero products from search operation. • FR3: Users should have used checkout from shopping cart.

3.3 Performance Requirements: The following are the key performance requirements:

- All pages should load within 2 seconds throughout the US region
- Search results should be displayed within 1 second
- Checkout should happen within 5 seconds after providing payment information

3.4 Design Constraints: XYZO should adhere to following standards:

- Web pages should be designed using HTML 4.0 transitional standards
- W3C Web Accessibility standards (Web Content Accessibility Guidelines (WCAG)) should be followed including keyboard navigation, alternate titles for images, etc.

3.5 Logical Database requirements: Existing product database will be leveraged and no changes to database is planned for XYZO application

3.6 Software System Attributes Requirements:

3.6.1 Reliability: XYZO should provide reliable and relevant search results 100% of times. The checkout operation should end reliably within 5 seconds.

3.6.2 Availability: XYZO should be available 99.999% of times throughout US region. All software upgrades, patches and fixes should be done without shutting down the application. There should be disaster recovery environment to handle natural disasters.

3.6.3 Security: Following security standards should be followed:

- Login operation should be performed using transport layer security (HTTPS)
- All user id and password information should be encrypted using one-way hash algorithms in the database
- Registration process should use CAPTCHA to prevent machine/robot brute force attacks.
- All input fields should be validated for SQL injection scenarios and HTML reserved words scenarios. Input should be sanitized before sending them to the upstream systems.
- There should be well-defined password policy covering password change frequency, invalid attempts allowed, etc.

3.6.4 Maintainability: The following maintainability features should be supported:

- XYZO should adopt standards based integration for extensibility and scalability.

- All code artifacts should have proper documentation.
- All code components should be thoroughly tested and the test coverage should be more than 80%.

☞ **Check Your Progress 2**

(1) The sub-section within Introduction section which elaborates of all terms used .....

(2) Integration with enterprise legacy systems is specified in .....

(3) Non-functional requirements like scalability, availability is specified in .....

1.5 SUMMARY

In this unit, we started looking at the definition of SRS and then moved on to examine the key concepts of SRS including its benefits, the main issues it addresses in a project, its characteristics, impact on project, and best practices. Then we looked at two standards of SRS and each section, sub-section of IEEE 830 standard are explained. We finally saw a case study to develop SRS from a problem statement.

1.6 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

(1) IEEE 830

(2) Unambiguous

(3) Cost, quality and schedule

Check Your Progress 2

(1) Definition, Acronym section

(2) External interface requirements section

(3) Specific requirements section

1.7 FURTHER READINGS

Reference Books

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

Dr. David Tuffley (2010), " Software Requirements Specifications: A How to Guide for Project Staff".

Reference Websites

http://en.wikipedia.org/wiki/Software_requirements_specification

http://segoldmine.ppi-int.com/documents/di-mccr-80025a-software-requirements-specification-srs

http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf

UNIT 2 FUNCTION ORIENTED MODELING

Structure

2.0 Introduction

2.1 Objectives

2.2 Data Flow Diagram (DFD)

- 2.2.1 Elements of DFD
- 2.2.2 Process Steps for Developing DFD
- 2.2.3 Example of Developing a DFD

2.3 Entity Relationship Diagram (ERD)

- 2.3.1 Elements of ERD
- 2.3.2 Process Steps for Developing ERD
- 2.3.3 Example of Development of a ERD

2.4 Structure Chart

- 2.4.1 Elements of Structure Chart
- 2.4.2 Process for Construction of a Structure Chart

2.5 Software Requirements Specification (SRS)

- 2.5.1 Needs of SRS

2.6 Data Dictionaries

- 2.6.1 Contents of Data Dictionary
- 2.6.2 Data Element/Item Details
- 2.6.3 Data Store Details
- 2.6.4 Data Process Details
- 2.6.5 Data Structure and Definition
- 2.6.6 Developing Data Dictionary

2.7 Summary

2.8 Answers to Check Your Progress

2.9 Further Readings

2.0 INTRODUCTION

This unit elaborates core concepts and various elements of function oriented modeling such as Data flow diagram (DFD), Entity-Relationship Diagram, Structure chart, Software requirement specification (SRS) and Data Dictionary.

Function oriented modeling is the visual representation of functions and processes within a system. It provides a structured way to describe the system behavior and functionality. Another key feature of functional model is that they describe the behavior of activity/action/system without specifying how they are implemented.

Computation intensive modules such as compilers and database functions are ideal candidates for being modeled in function oriented modeling.

Following are the key benefits of function oriented modeling:

- Function oriented modeling helps us to define the functions, processes, activities in a more structured way.
- It helps to model the function hierarchy in intuitive visual format.
- It helps in discovery of information and depict flow of data/information.

- It depicts the system from function and behavior perspective.
- It helps in comparing functionality and modules based on its behavior.
- It helps in identifying information thereby leading to new solution.

Elements such as system functions, activities, actions, data transformation are ideal candidates for Function Oriented Modeling.

The process of creating function oriented modeling for ground-up design involves following steps:

1. The first step of function oriented modeling is to come up with functional requirements which provide detailed description of the system.
2. From the requirements, we identify the major, high level functionality. We identify nouns and verbs related to the system's behavior.
3. In the final step, the identified functions are modeled depicting their hierarchy and relationship. At the root of the hierarchy, there will be a most abstract function and when we traverse down the hierarchy, the functions are more simpler and less abstract.

The process of creating function oriented modeling for reverse-engineering process involves following steps:

1. Start with the overall highest level module which describes the system's functionality.
2. Iteratively identify the sub-functions/sub-component and its functionality.
3. Model this hierarchy in functional modeling.

Conceptually function oriented modeling of process, activities, actions and function is aimed at depicting the computational behavior of the system. It breaks the system into its constituent functionality and depicts the data flow and relationship between those functions. The high level behavior is depicted at top-level functions and it is incrementally broken down at low-level sub-functions.

On the other hand, Object oriented modeling aims at modeling the system as real-world objects. The objects encompass both behavior and data.

2.1 OBJECTIVES

After going through this unit, you should be able to

- define key concepts of function oriented modeling,
- understand elements of Data Flow Diagram (DFD) and the process of its construction,
- understand elements of Entity Relation Diagram (ERD) and the process of its construction,
- understand elements of Structure chart,
- know details related to Software Requirements Specification (SRS), and
- know concept of Data Dictionary.

2.2 DATA FLOW DIAGRAM (DFD)

Data Flow Diagrams depict the flow of the data, data transformation along with source and destination of data. It visualizes the process of data transformation, relations and data processing among various inputs and outputs and internal/external entities of the system.

In the simple example shown below (Figure 2.1), the function "compute_square_area" takes "width" and "height" as input parameters and provides the "Area of the square" as the output.

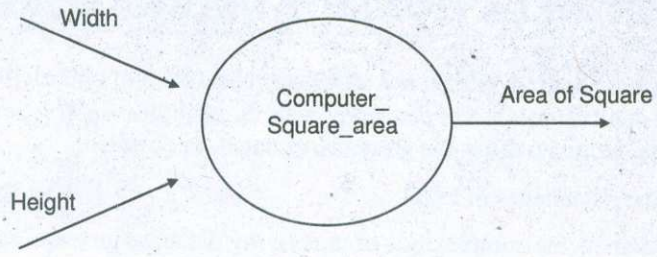


Figure 2.1 : Example of DFD

2.2.1 Elements of DFD

Following are the key visual elements of DFD

- **Rectangle** to represent physical entities such as Computer, Building, etc.
- **Circle** to depict the function like *compute_area_of_square*
- **Directed line** to show the data flow
- **Horizontal parallel lines** to depict the data structure or data store like File System or Database or ERP system

2.2.2 Process Steps for Developing DFD

To create level 0 DFD, normally, following steps are followed:

1. Identify the main data objects and associate operations.
2. Identify external entities which produce and consume the data.

After this, level 1 DFD can be created by following these steps:

1. Specify the data transformation logic.
2. Depict the data transformation for input and output and all the data flows.

2.2.3 Example of Developing a DFD

Following is the DFD for "Searching a number" function. High level function "searching a number" using binary search contains following sub-tasks implemented as sub-routines:

1. Collect the input into a numerical list.
2. Sort the list in ascending order.
3. Perform binary search and return the result.

Applying the steps given for constructing DFD mentioned above, the DFD for this is given in Figure 2.2:

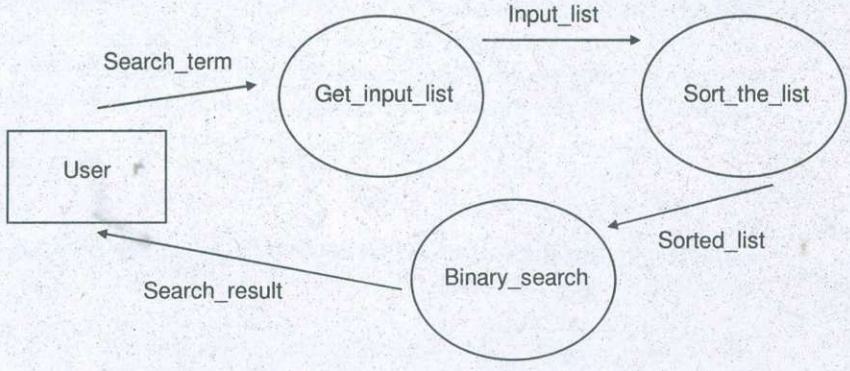


Figure 2.2 : DFD for Searching a Number

2.3 ENTITY RELATIONSHIP DIAGRAM (ERD)

Entity Relationship Diagram (ERD) is a visual representation of underlying data model in the database. It depicts various entities along with its attributes and relationships with other entities and helps visualize the structure of database objects.

Following are the advantages of ERD:

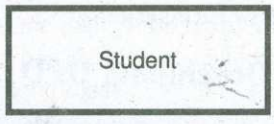
(1) It depicts the complexities of underlying database in visual structure. This helps in structured depiction and communication.

(2) ERD helps in design of data model by identifying different elements and relationships among the elements.

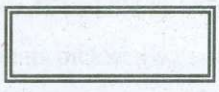
2.3.1 Elements of ERD

The main building blocks of ERD are given below along with their symbols:

- **Entity:** All nouns such as person, place, object, event qualifies as an entity in the ERD. An entity usually has attributes which define its properties. Examples of entities include Address Details, Person Details, Vehicle details, Student details etc. An entity is depicted by following symbol:

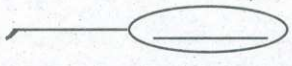


Entities are termed as "weak entities" if they depend on other entity for its existence. For instance, an entity such as "Item" which depends on another entity "ItemDetail" is a weak entity. They are denoted by following symbol:



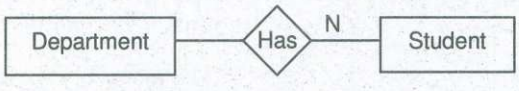
- **Attributes** describe the properties of an entity. For instance, a Student Entity has attributes such as StudentId, StudentName, StudentAddress and StudentPhone. It is denoted by the following symbol:

Key attributes are the ones which uniquely identify the entity. In the above example, StudentId is the key attribute as it can uniquely identify a student. It is denoted by the following symbol:

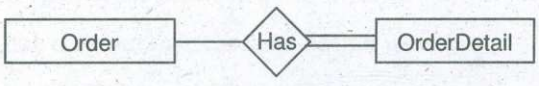


Attributes can also be multi-valued if they contain multiple values such as names of countries. This is depicted by

- **Relationship** depicts the relationship between two entities and how the data is shared across two entities. Cardinality depicts the number of instances of one entity related to instance of another entity. It can be one-to-one or one-to-many or many-to-many. Following diagram depicts one-to-many cardinality in the relationship as **Department** entity has many students



Complete participation of **entity OrderDetail** in the relationship is depicted as follows:



Some entities can have "self-relationship" wherein they relate to themselves. One example is "Employee" entity relate to itself through "manage" relationship as Manager is also an employee but he/she manages at least one other employee.

2.3.2 Process Steps for Developing ERD

The following are the main steps in building an ERD:

1. Identify the nouns in database domain which depict objects. These objects become entities in ERD.
2. Identify all the properties which are required and used in the current domain. The properties become the attributes for an entity.
3. Identify the relationship between entities.
4. Refine entities and remove any redundant entities or relationships.

2.3.3 Example of Development of a ERD

The following is the ERD for modeling employee and his/her relationship with a department in an organization. After applying the above steps, we identify two main entities: Employee and Department which are related by "Assigned To" relationship.

All main attributes of Employee are identified and "EmployeeId" is the primary key attribute which uniquely identifies employee. Similarly, all main attributes of Department entity are identified with "DeptId" as the primary key attribute.

The ERD is shown in Figure 2.3.

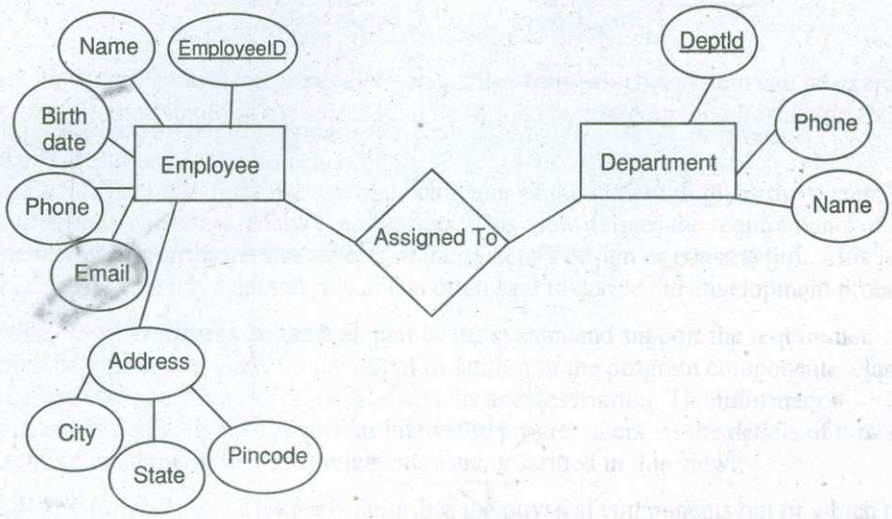


Figure 2.3: Employee-Department ERD

2.4 STRUCTURE CHART

A structure chart (SC) decomposes the high level system into multiple, executable tasks. It follows the top-down design approach and represents module hierarchy in tree structure. Structure chart essentially describes the list of functions, sub-functions along with their relationship that constitute a system along with data and control flow. Structure chart is the next step after DFD during design and implementation as SC provides more details than DFD. SC uses information from Data dictionary which is detailed in subsequent sections.

A structure serves following purposes during design:

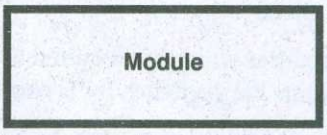
- Breaks the system into smaller and executable functional tasks
- Depicts the complexity and size of the system

2.4.1 Element of Structure Chart

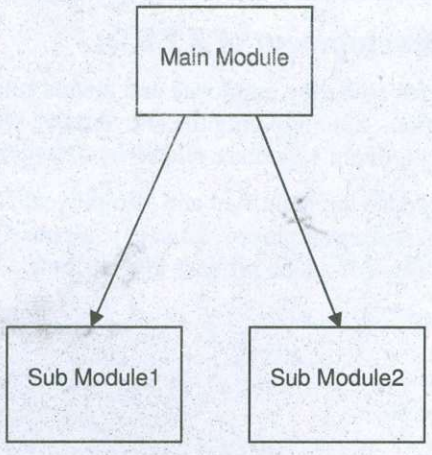
The main building blocks of Structure Chart are given below along with their symbols:

- **Module** depicts a function or a sub-function and is represented by a rectangle. If a function invokes multiple sub-functions, then the main module branches to sub-modules. It is basically a unit of execution which accepts input parameters and produces output parameters.

  It is denoted as follows:

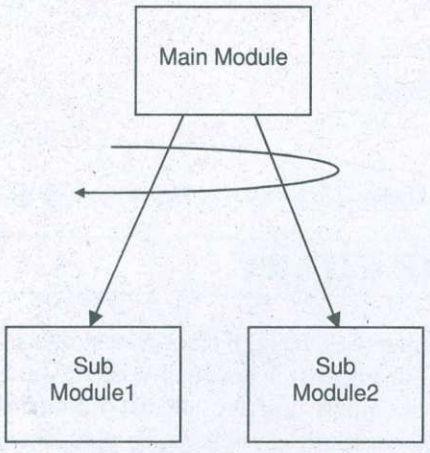


Main module invoking sub modules is depicted below:



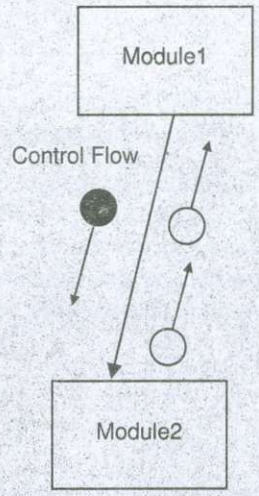
- **Condition** decides which module is to be invoked based on the condition. It is denoted by a diamond. A conditional invocation of sub module is denoted by  in the main module.
- **Loop** indicates the repetition of one or more modules and is depicted by a curved arrow

  Execution of sub modules within a loop is denoted as follows:

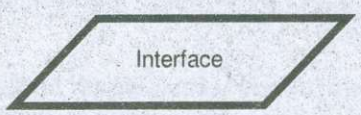


- **Data couple** is shown by an arrow with empty circle and it denotes the data that is flown from one module to another. The flow of information has a direction.
- **Control Flow** is shown by an arrow with filled circle and denotes the function call from one module to another.

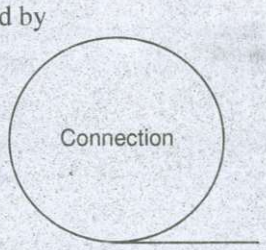
Data and control flow are depicted below:



- Devices such as peripheral devices and external interfaces are denoted by



- Software infrastructure and connections to external systems, databases, ERP systems are denoted by



2.4.2 Process for Construction of a Structure Chart

Let us construct a structure chart for calculating a hotel bill for a customer. This high level function is decomposed into four sub functions:

- Calculate total order amount
- Calculate value added tax (VAT)
- Calculate service charge
- Calculate any discounts applicable for the customer

Each of these sub-functions take input and output parameters. The structure chart along with data flow is shown in Figure 2.4.

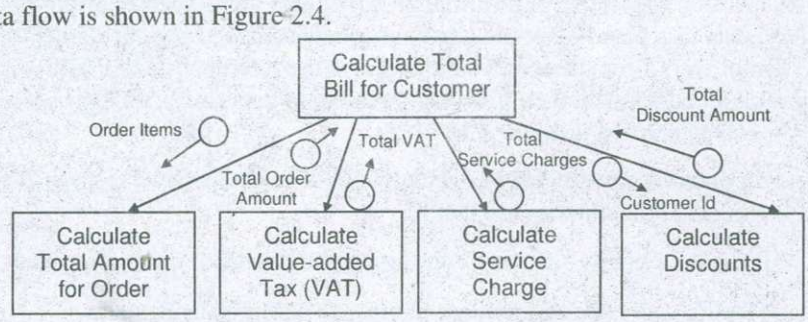


Figure 2.4 : Structure Chart

☞ **Check Your Progress 1**

(1) A data flow in DFD is depicted by .....

(2) In ERD, entities which depend on other entities for existence are known as .....

2.5 SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

A Software Requirements Specification (SRS) is a specification which provides structured description of the system's requirements, behavior and interfaces.

SRS is essentially a contract document which encompasses various sections and sub-sections to provide sufficient coverage to touch base all important aspects of project implementation. It elaborate detailed and specific functional requirements such as product functionality, interface integration and specifies non-functional requirements such as performance, security, reliability, maintainability, scalability, etc. Refer to the structure of the SRS in earlier unit.

2.5.1 Need for SRS

Some of the key benefits of SRS are as follows:

- **Serves as a solid contract specification about the software functionality:** SRS serves as a structured contract between these parties specifying all functionalities along with constraints and mentions the behavior of the intended software.
- **Reduces development and testing effort:** Project management team and technical team get clear set of requirements which reduces duplicate/unnecessary effort.
- **Forms basis for cost and schedule estimation:** It is possible to meaningfully estimate the overall project cost and schedule in using details specified in SRS.
- **Forms basis for verification and validation:** Quality team can design the validation and testing strategy including various kinds of test cases based on the requirements specified in SRS.
- **Helps software portability and installation:** The software usability information contained in SRS helps to transfer the software across various locations including multiple inter-company departments and other external customers.
- **Helps in enhancement:** As SRS specifies each requirement in fullest details, it would be easier to assess the impact of any enhancement planned providing the cost and schedule estimate of the enhancement.

2.6 DATA DICTIONARIES

Data dictionary is the repository of information about data items such as origin of data, data structure, data usage and other metadata information related to data elements. It is used as system of record for structure chart and for other references. Data dictionaries help to organize and document the information related to data flows, data processes and data elements in a structured fashion.

Usually data dictionaries are created in parallel with DFD in top-down fashion. The main benefits of having data dictionaries are that it:

- Provides a highly structured definition and details of data elements
- Identifies all alias and reduces duplicates within data elements
- Helps in developing logic for processes
- Helps in construction and validation of DFD and structure chart
- Helps in development of reports.

2.6.1 Contents of Data Dictionary

Data dictionaries contain information about following high level elements:

- Data element/item details
- Data structure and definition
- Data process
- Data stores

2.6.2 Data Element/Item Details

Data element or data item is the most primitive and the smallest information about data. For example StudentFirstName, BookId etc. Multiple data elements combine to form a data record. For example all data elements related to student will form student record.

The following information about data element is stored in data dictionary:

- Unique Id: This uniquely identifies the data element. Example StudentId
- Name: Specifies the name of the data element. Example: StudentFirstName
- Alias: All known aliases by which the data element is referenced or identified. A Student data element may be referenced by these aliases: Pupil, ProgramMember, CollegeStudent
- Default value: The value stored if no value is specified. Example default value of "Indian" for "Citizenship" attribute
- Length: Includes minimum length and maximum length
- Description: About the data element
- Data Type: Specifies the data type of the element such as integer, real number, String etc.
- Input format
- Output format
- Comments
- Validation criteria: This includes if the element requires any data type validation or number format validation or range validation.

2.6.3 Data Store Details

It specifies the storage format of data element such as flat file or data base record. It also provides details including the volume of data stored and frequency of data storage.

If the data storage is file system, then details such as file name, file path, file restrictions etc. will be mentioned. In the case of database storage, data dictionary provides information related to table, schema name etc.

2.6.4 Data Process Details

Data dictionary contains information related to data process such as:

- Process Name
- Process inputs
- Process output
- Type of the process
- Logic executed in process

2.6.5 Data Structure and Definition

Complex data elements are described using following symbols:

- "composed" denoted by "equals"
- "and" denoted by "+"
- "or" denoted by "/"

- "optional" denoted by "()"
- "repetition" denoted by "{ }"

For instance, a "book record" for a library can be described as follows:

Book Record = Book ID + Book Name + Author Name + Publisher Name + (Publication Date).

2.6.6 Developing Data Dictionary

For creating data dictionary we first identify the main data elements, data store, data processes, external and internal entities. Once the high level data elements are identified, more details such as data structure, element details are described. Similarly all data flows and data stores are further expanded to describe the data structure of its data elements.

☞ **Check Your Progress 2**

(1) Two examples of software system attributes in SRS are ……. and …….

(2) The characteristic of SRS which make it to have a single interpretation is called ……..

(3) ……detail of data element in data dictionary identifies all known references.

(4) Optional element within a record of data dictionary is denoted by ……

2.7 SUMMARY

In this unit, we started by understanding the building blocks of function oriented modeling and the high level process of creating it. We then examined the main differences between object oriented modeling and function oriented modeling. We then discussed the elements, process steps for developing with an example for DFD, ERD and structure chart. At the end, we looked at various concepts of data dictionaries like contents of data dictionary and how to develop a data dictionary.

2.8 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

(1) Directed arrow

(2) Weak entities

Check Your Progress 2

(1) Reliability and Availability

(2) Unambiguous

(3) alias

(4) ()

2.8 FURTHER READINGS

Reference Books

- Peretz Shoval: *Functional and Object Oriented Analysis and Design: An Integrated Methodology.*
- Booch, G. (2002): *Growing the UML.* Software and Systems Modeling.
- Bhattacharyya, S. S., Murthy, P. K., and Lee, E. A.(1996): *Software Synthesis from Dataflow Graphs.*

Reference Websites

http://www.en.wikipedia.org/wiki/Data_dictionary

http://www.en.wikipedia.org/wiki/Structure_chart

http://www.en.wikipedia.org/wiki/Entity–relationship_model

http://www.en.wikipedia.org/wiki/Data_flow_diagram

UNIT 3 OBJECT ORIENTED MODELING

Structure

3.0 Introduction

3.1 Objectives

3.2 Unified Modeling Language (UML)

- 3.2.1 Diagrams in the UML
- 3.2.2 Architecture of the System

3.3 Use Case and Use Case Diagram

- 3.3.1 Use Case Approach
- 3.3.2 Rules to Create Use Case
- 3.3.3 Template for Use Case
- 3.3.4 Use Case Diagram
- 3.3.5 Use Case Diagram for Flight Booking System

3.4 Classes and Class Diagram

3.5 Exercise

3.6 Summary

3.7 Further Readings

3.0 INTRODUCTION

An organization that can develop the quality software according to user's needs in a timely and knowable fashion, with a professional and effective use of resources, both human and material is one that has sustainable business. Modeling is the heart of all the activities that lead up to the deployment of good software. We build models to communicate the desired structure and behaviour of the system. Models are the replicas of the system for simplification and reuse. We build models to manage risk.

Importance of Modelling

If you want to build a house for people, you need some experience for that. If you want to build a multistory building, then, you need expertise for that. You should do extensive planning, because the cost of failure is high.

Higher level of expertise is needed when you want to build such software that will be used by other users. Modeling is proven and well-accepted engineering technique. We build architectural models of houses and towers to help owners to visualize the final product. We may even build mathematical models in order to analyze the effects of winds or earthquakes on the buildings.

A model is simplification of reality. We build models so that we can better understand the system we are developing. Through modelling, we can achieve the following:

- A model helps us to visualize a system as it is or as we want it to be.
- Models permit us to specify the structure or behavior of a system.
- A model gives us template that guides us in building a system.
- A model documents the decision we have made.

Principles of Modeling

The use of modeling has a wealthy history in all the engineering disciplines. Four basic principles of modeling are the following:

1. The choice of what models to create has a profound influence on how a problem is approached and how a solution is prepared.
2. Every model may be expressed at different levels of precision. The same model can be scaled up or down to different granularities.
3. The best models that are relevant to reality are made. They are simplified without hiding important information.
4. No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models

Object Oriented Modeling

To build a model in software engineering, we may use several approaches. Two commonly used approaches are an algorithmic prospective and an object oriented perspective.

In algorithmic perspective, the main building block of all software is the procedure or function. This approach leads developers to focus on the issues of control and the decomposition of larger algorithms into smaller algorithms. The disadvantage of this approach is that as the requirement change and the system grows, system built with an algorithmic focus turns out to be very hard to maintain.

In object oriented approach, the main building block of all software systems is the object or class. Every object has identity, state and behavior. A class is a description of a set of common objects.

3.1 OBJECTIVES

After going through this unit, you should be able to

- represent user requirements pictorially,
- design the document in the form of use cases,
- know various components of UML (Unified Modeling Language),
- know various components of use cases, and
- know the use of use cases.

3.2 UNIFIED MODELING LANGUAGE (UML)

The purpose of UML is to visualize, specify, construct and document object oriented system. It is very expressive language that addresses all the views needed to develop and then deploy the system. The UML is used in a process that is use case driven, architecture centric, iterative in nature and incremental. The UML can be used for domains such as Enterprise information system, Banking and financial services, Retail services, Communication industry, Military, Medical electronics, Scientific issues and Distributed web services.

3.2.1 Diagrams in the UML

UML supports specifications for following diagrams which capture and document different perspectives of problem domain starting from inception to installation and maintenance too. The diagrams are as follows:

- Class diagram (Design View)
- Use case diagram (use case view)
- Sequence diagram (use case and design view)
- Collaboration diagram (use case and design view)

- Deployment diagram (deployment view)
- Object diagram (use case and design view)
- State chart diagram (design and process view)
- Activity diagram (design and process view)
- Component diagram (implementation view).

3.2.2 Architecture of the System

Architecture is the set of significant decisions about the following: the organization of the software system, the selection of structural elements and their interfaces by which the system is composed of, their behaviours, as specified in the collaborations among those elements, and the composition of these structural and behavioural elements into progressively larger subsystems.

The architectural style that guides this organization are the static and dynamic elements and their interfaces, their collaborations and composition. Figure 3.1 depicts the architecture of the system.

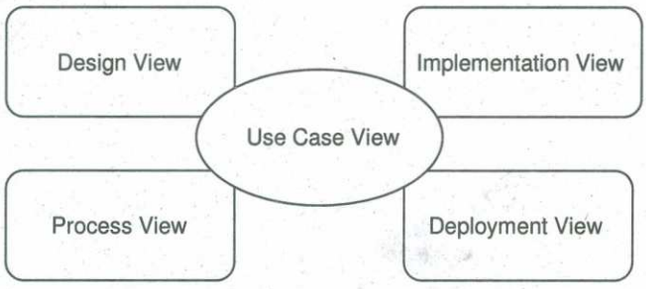


Figure 3.1: Architecture of the System

Each view corresponds to a particular perspective from which a system can be examined. A complete description of the system can be formed by merging the information found in all views.

Use Case View: It defines the external behaviour of the system. It gives the system's description to end users, analysts and testers. This view defines the requirements of the system which describe certain aspects of the system's design or construction. This is why the use case view has a central role and is often said to derive the development process.

Design View: It focuses on the how part of the system and support the requirements expressed in use case view. It consists of definition of the program components, classes along with the specification of data, behaviour and description. The information contained in this view is of particular interest to programmers, as the details of how the system's functionality will be implemented are described in this view.

Implementation View: This view describes the physical components out of which the system is to be constructed. These are distinct from the logical components described in the design view and includes things such as executable files, libraries of code and databases. The information produced by this view is used for configuration management and system integration.

Process View: It deals with the issues of concurrency within the system.

Deployment View: It describes the distribution of physical components across the environment such as network of computers.

Process and deployment views address non functional requirements of the system such as performance issues. These views are less developed in UML as compared to design view.

3.3 USE CASE AND USE CASE DIAGRAM

This section introduces Use case approach, rules to create use case, template for use case, Use case diagrams and an example of use case diagram for flight booking system.

3.3.1 Use Case Approach

This is used to represent the system pictorially. It is a combination of text and images to understand the requirements of system clearly. In this approach, we mainly use three terms, namely, use case, use case scenario and use case diagram. Use cases are templates for the description of user requirements and written in English language point wise. Use case scenario is unstructured description of user requirements. Use case diagrams are the pictorial representation of system

3.3.2 Rules to Create Use Case

The following are the rules to create a use case:

- Identify all the end users. That is, actors of the system.
- Build the profile of user according to assigned task.
- Build a use case for each requirement.
- Create the template for use case.
- Review and validate the users.

3.3.3 Template for Use Case

The Jacobson et al. proposed a template for writing use case as given in Table 3.1:

Table 3.1 : Template for Use Case

1.	Introduction: Describe the purpose of use case
2.	Actors: List the actors of the system
3.	3. Flow of Events 3.1 Basic Flow: List the primary events that will occur when this use case is executed. 3.2 Alternative Flows: Any other possible flow in this use case. Such flows should be separately listed. A use case may have many alternate flows
4.	Special Requirements: Business rule for the basic and alternative flows should be listed as special requirements in the use case narration. These business rules will also be used for writing test cases. Both success and failure scenarios should be described here
5.	Pre Conditions: Condition that needs to be satisfied for the use case to execute
6.	Post Condition: After the execution of use case, different states of the system are defined here
7.	Extension Points

3.3.4 Use Case Diagram

It shows the relationship between the user and the system and creates the boundary of the system. This diagram is used to:

- Understand the requirements clearly.
- Enable users to understand the system.
- Generate test case to validate the system.
- Build project schedule.

In Use case diagram, we use seven symbols. They are given below:

- **System:** It shows the boundary of the system and is represented by a rectangle (Figure 3.2).

Figure 3.2 : System

- **Actor:** It represents the user or people or device which interacts with the system. It gives input to the system. They are outside of the system boundary. Actor is represented as given in Figure 3.3.

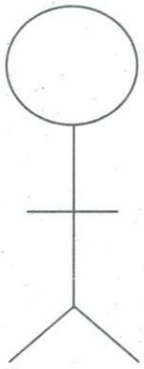


Figure 3.3 : Actor

- **Use Case:** It shows the desired function of the system. It defines the requirements of the system. Actors gives input to the use case and use case provides output for the given input. It can be represented as given in Figure 3.4.

Figure 3.4 : Use Case

- **Association:** It is a structural relationship that describes a set of links (a link being a connection among objects). Aggregation is a special kind of association, representing a structural relationship between a whole and its parts.

  It is represented by a line (may be directed), including labels such as multiplicity and role name as given in Figure 3.5.

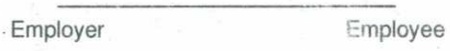


Figure 3.5 : Association

- **Dependency:** This is used to show the relationship between use cases and is represented by a directed dashed line and occasionally includes label (Figure 3.6).

Figure 3.6 : Dependency

- **Generalization:** It shows the concept of inheritance among use cases and actors. A generalization is a relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent). In this way, child shares the structure and behavior of the parent. It is represented by a line with a hollow arrowhead pointing towards the parent (Figure 3.7).

Figure 3.7 : Generalization

- **Realization:** It is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. It is represented by a dashed line with a hollow arrowhead pointing towards the parent (Figure 3.8).

Figure 3.8 : Realization

3.3.5 Use Case Diagram for Flight Booking System

Requirements for the flight reservation system are given below (Figure 3.9):

1. Use cases: Login
   Book Ticket
   View booking status
   View flight schedule
   Ticket Cancellation
   Update flight schedule
   Report generation
2. Actors:
   - Administrator for the system
   - Traveler Reservation clerk

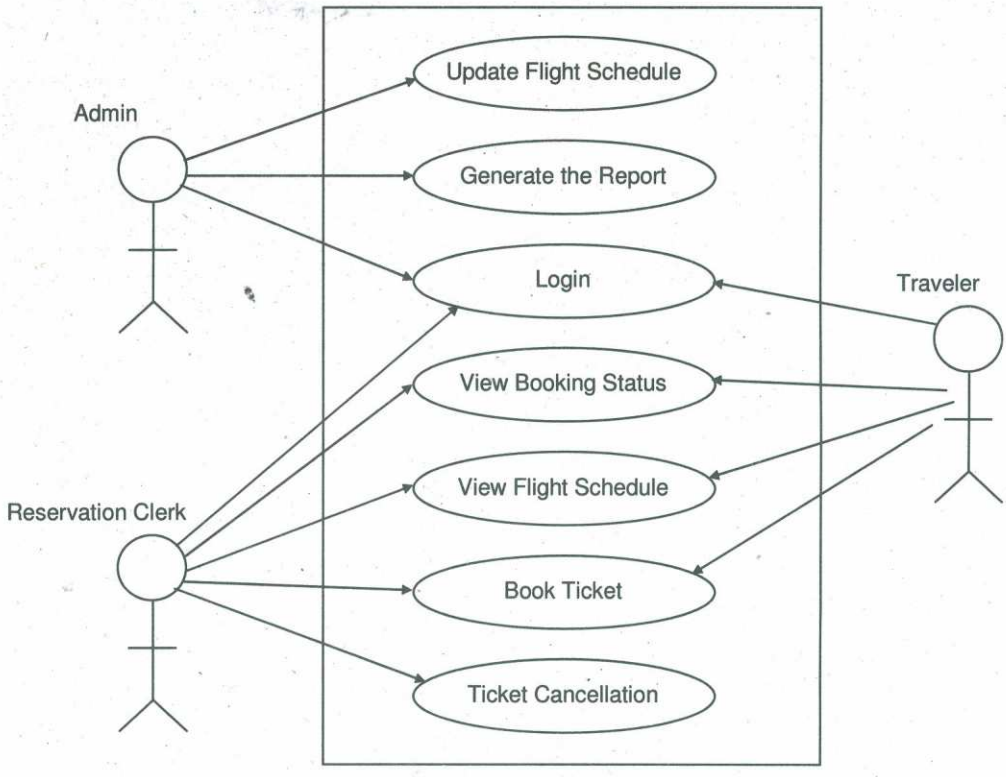


Figure 3.9 : Use Case Diagram for Flight Booking System

3.4 CLASSES AND CLASS DIAGRAM

Class diagram shows the static view of the system by showing the classes and their relationships. A class integrates the data and behavior into one unit. In UML, we can represent the class by a rectangle consisting of class name, attributes and operation. Figure 3.10 gives an example of class diagram.

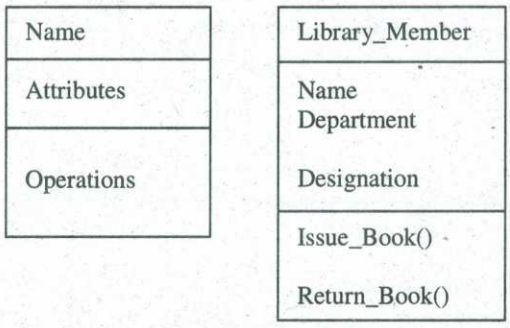


Figure 3.10 : Example of Class Diagram

Class specifies the visibility values which include public (+), private (–), Protected (#) and package (~). While writing operation specification, it must specify all arguments and their data types (separated by colon), return data type, its visibility and the constraints within {}. For example, compute_cost operation is written as + compute_cost (Order: Order) Rupees {The Total cost is sum of all items cost (unit price * quantity) mentioned in the order less the trade discount}.

A detailed class specification for Library_Member class is as follows (Figure 3.11):

Library_Member	
-	Name : String = blank
-	Member_id : String = {assigned by the system}
-	Address: string = blank
-	Security_deposit: Rupees =0
-	Semester : integer = 0
-	Branch : String = blank
-	Library_history : String = blank
+	Set_name (name: string)
+	Set_branch (name : string)
+	get_security(): rupees
+	set_Library_history (Library_history : string)
+	Update_ Library_history (Library_history : string)

Figure 3.11 : Detailed Class Classification

Three types of relations, namely, association, aggregation and generalization can exist between classes in the class diagram. Association relates a class A to class B and is shown as a link between classes. While reading textual description of requirements they correspond to verbs and a class corresponds to nouns.

An association between two classes is shown below (Figure 3.12):

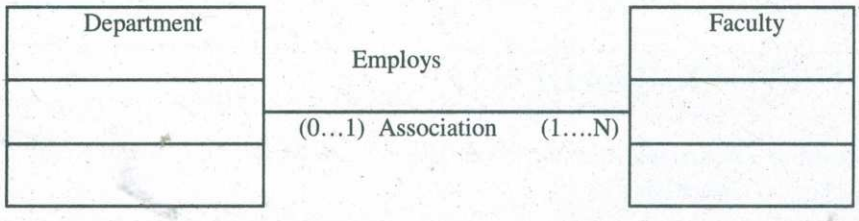


Figure 3.12 : Association between Classes

Generalization between two classes is shown by is-a relationship. The classes between which Is-a relationship is identified are called superclass and subclass respectively. Subclasses inherit the features of superclass in addition to features which are specific to subclass. A generalization is represented as in Figure 3.13.

MPDD-IGNOU/P.O. 2.0 K/Sep. 2018 (Reprint)

ISBN : 978-81-266-6610-2