



RQF LEVEL 4



SWDBS401
SOFTWARE
DEVELOPMENT

Backend System Design

TRAINEE'S MANUAL

October, 2024



BACKEND SYSTEM DESIGN



AUTHOR'S NOTE PAGE (COPYRIGHT)

The competent development body of this manual is Rwanda TVET Board ©, reproduce with permission.

All rights reserved.

- This work has been produced initially with the Rwanda TVET Board with the support from KOICA through TQUM Project
- This work has copyright, but permission is given to all the Administrative and Academic Staff of the RTB and TVET Schools to make copies by photocopying or other duplicating processes for use at their own workplaces.
- This permission does not extend to making of copies for use outside the immediate environment for which they are made, nor making copies for hire or resale to third parties.
- The views expressed in this version of the work do not necessarily represent the views of RTB. The competent body does not give warranty nor accept any liability
- RTB owns the copyright to the trainee and trainer's manuals. Training providers may reproduce these training manuals in part or in full for training purposes only. Acknowledgment of RTB copyright must be included on any reproductions. Any other use of the manuals must be referred to the RTB.

© **Rwanda TVET Board**

Copies available from:

- *HQs: Rwanda TVET Board-RTB*
- *Web: www.rtb.gov.rw*
- **KIGALI-RWANDA**

Original published version: October, 2024

ACKNOWLEDGEMENTS

The publisher would like to thank the following for their assistance in the elaboration of this training manual:

Rwanda TVET Board (RTB) extends its appreciation to all parties who contributed to the development of the trainer's and trainee's manuals for the TVET Certificate IV in Software Development, specifically for the module "**SWDBS401: Backend System Design**".

We extend our gratitude to KOICA Rwanda for its contribution to the development of these training manuals and for its ongoing support of the TVET system in Rwanda.

We extend our gratitude to the TQUM Project for its financial and technical support in the development of these training manuals.

We would also like to acknowledge the valuable contributions of all TVET trainers and industry practitioners in the development of this training manual.

The management of Rwanda TVET Board extends its appreciation to both its staff and the staff of the TQUM Project for their efforts in coordinating these activities.

This training manual was developed:

Under Rwanda TVET Board (RTB) guiding policies and directives



Under Financial and Technical support of



COORDINATION TEAM

RWAMASIRABO Aimable

MARIA Bernadette M. Ramos

MUTIJIMA Asher Emmanuel

Production Team

Authoring and Review

TUYIZERE Emmanuel

HATANGIMANA John

Validation

KWIZERA Israel

MBONYISIMBI Dieudonne

Conception, Adaptation and Editorial works

HATEGEKIMANA Olivier

GANZA Jean Francois Regis

HARELIMANA Wilson

NZABIRINDA Aimable

DUKUZIMANA Therese

NIYONKURU Sylvestre

MANIRAKIZA Jean de Dieu

Formatting, Graphics, Illustrations, and infographics

YEONWOO Choe

SUA Lim

SAEM Lee

SOYEON Kim

WONYEONG Jeong

KANANI Vincent

Financial and Technical support

KOICA through TQUM Project

TABLE OF CONTENT

AUTHOR’S NOTE PAGE (COPYRIGHT)-----	ii
ACKNOWLEDGEMENTS-----	iv
TABLE OF CONTENT -----	vii
ACRONYMS-----	viii
INTRODUCTION -----	1
MODULE CODE AND TITLE: SWDBS401 BACKEND SYSTEM DESIGN-----	2
Learning Outcome 1: Analyse System Backend-----	3
Key Competencies for Learning Outcome 1: Analyze System Backend-----	4
Indicative content 1.1: Gathering FURPS Requirements -----	6
Indicative content 1.2: Identification of Main Objects of Backend System-----	28
Indicative content 1.3: Description of System Interaction-----	32
Indicative content 1.4: Report of the System Backend Requirements-----	35
Learning outcome 1 end assessment -----	38
References-----	41
Learning Outcome 2: Develop System Structure -----	42
Key Competencies for Learning Outcome 2: Develop System Structure-----	43
Indicative content 2.1: Identification of System Design Tools -----	45
Indicative content 2.2: Identification of Hardware and Software Technology-----	82
Indicative content 2.3: Application of SSADM (Structured System Analysis and Design Methods) -----	84
Indicative content 2.4: Application of Object-Oriented Analysis and Design -----	89
Learning outcome 2 end assessment -----	92
References-----	94
Learning Outcome 3: Build System Design-----	95
Key Competencies for Learning Outcome 3: Build System Design-----	96
Indicative content 3.1: Development of Data Flow Diagram-----	98
Indicative content 3.2: Application of Physical Data Model-----	106
Indicative content 3.3: Documentation of System Design -----	113
Learning outcome 3 end assessment -----	124
References-----	126

ACRONYMS

API: Application programming Interface

CPU: Central processing Unit

DB: Database

DBaaS: Database as a service

DBMS: Database Management System

DDL: Data Definition language

DFD: Data Flow Diagram

DML: Data Manipulation Language

ERD: Entity Relational Diagram

FSD: Functional Specification Document

FURPS: Functionality, Usability, Reliability, Performance, and Security.

GWT: Google Web Toolkit

ITC: Insert Time Clustering

JSON: JavaScript Object Notation

KOICA: Korean international cooperation agency

KPIS: Key performance Indicators

MDC: multitenant database containers

MVC: Model-View-Controller

MYSQL: My Structured Query Language

NICs: Network Interface Cards

OOAD: Object Oriented Analysis and Design

ORM: Object Relational Mapper

OS: Operating System

PHP: Hypertext Preprocessor

RAD: Rapid Application Development

RAM: Random Access Memory

RTB: Rwanda TVET Board

SDD: System Design Document

SDLC: System development life cycle

SPA: Single Page Application

SQL: Structure query Language

SRIS: Student Result Management System

SRS: Software Requirement Specification

SSADM: Structured System Analysis Design Methods

SSDs: Solid State Drives

SSO: Sing sign on

TQUM Project: TVET Quality Management Project

TSD: Technical Specification Document

UML: Unified Modeling language

XML: Extensible Markup Language

INTRODUCTION

This trainee's manual includes all the knowledge and skills required in software development specifically for the module of “**Backend System Design**”. Trainees enrolled in this module will engage in practical activities designed to develop and enhance their competencies. The development of this training manual followed the Competency-Based Training and Assessment (CBT/A) approach, offering ample practical opportunities that mirror real-life situations.

The trainee's manual is organized into Learning Outcomes, which is broken down into indicative content that includes both theoretical and practical activities. It provides detailed information on the key competencies required for each learning outcome, along with the objectives to be achieved.

As a trainee, you will start by addressing questions related to the activities, which are designed to foster critical thinking and guide you towards practical applications in the labor market. The manual also provides essential information, including learning hours, required materials, and key tasks to complete throughout the learning process.

All activities included in this training manual are designed to facilitate both individual and group work. After completing the activities, you will conduct a formative assessment, referred to as the end learning outcome assessment. Ensure that you thoroughly review the key readings and the 'Points to Remember' section.

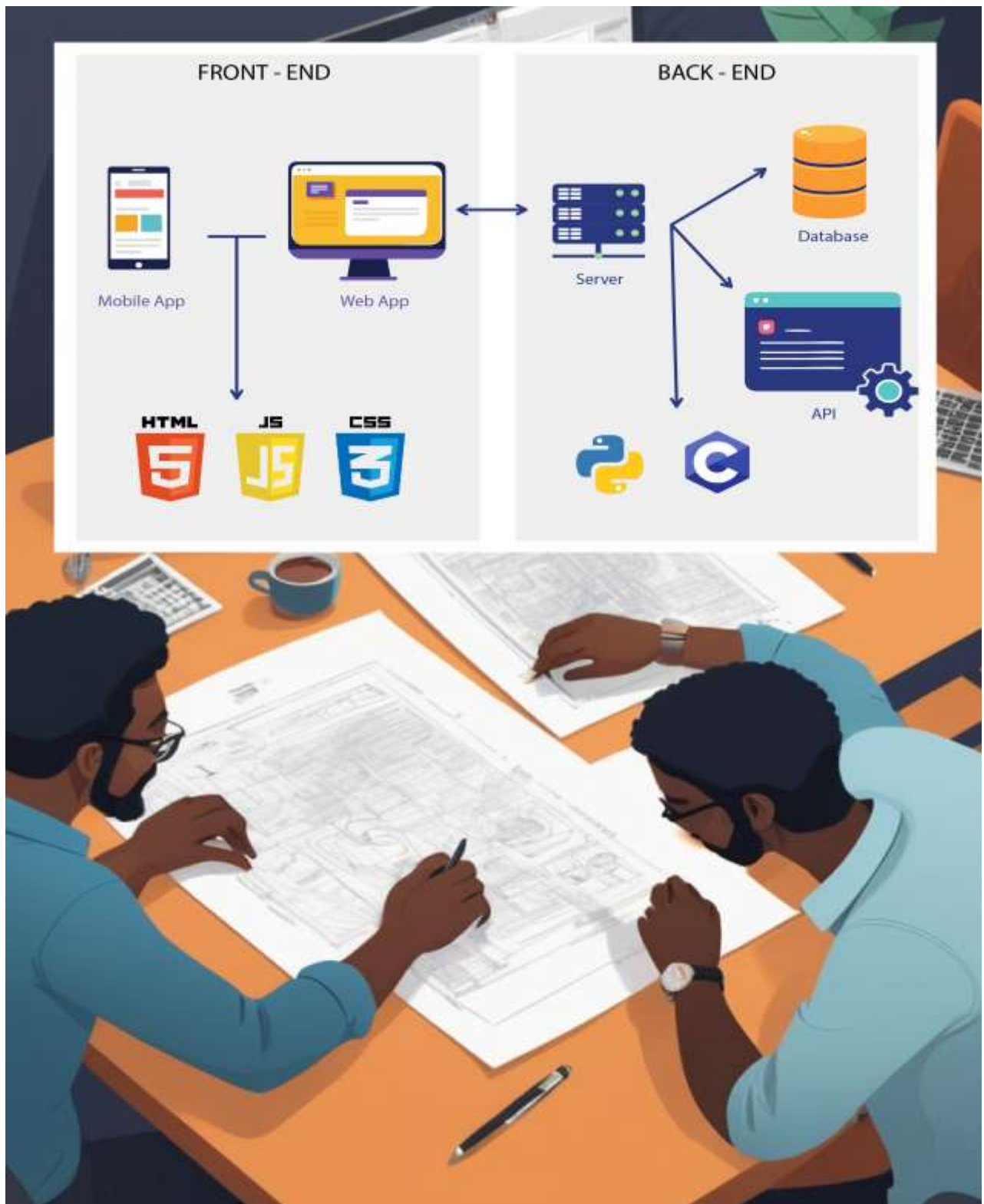
MODULE CODE AND TITLE: SWDBS401 BACKEND SYSTEM DESIGN

Learning Outcome 1: Analyse System Backend

Learning Outcome 2: Develop System Structure

Learning Outcome 3: Build System Design

Learning Outcome 1: Analyse System Backend



Indicative contents

1.1 Gathering FURPS Requirements

1.2 Identification of Main objects of Backend System

1.3 Description of System Interaction

1.4 Report of the System Backend Requirements

Key Competencies for Learning Outcome 1: Analyze System Backend

Knowledge	Skills	Attitudes
<ul style="list-style-type: none">● Description of FURPS key terms in Backend System● Description of backend development Technologies and tools.● Description System development life cycle (SDLC)● Identification of system Analysis tools● Identification of FURPS Requirements● Identification of main objects of Backend System● Description of system Interaction	<ul style="list-style-type: none">● Conducting data gathering● Documenting report of system requirements	<ul style="list-style-type: none">● Having Team work● Being critical thinker● Being Innovative● Being attentive.● Being creative● Having Curiosity and Open Mindedness



Duration: 17 hrs



Learning outcome 1 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe properly key terms of backend system according to backend system design.
2. Describe correctly backend development Technologies according to backend system design.
3. Describe properly System development life cycle (SDLC) according to backend system design
4. Identify properly system analysis tools based on backend system design
5. Identify properly main objects based on the system analysis methodology
6. Describe clearly system interactions based on identified objects
7. Report appropriately System Backend Requirements in line with software requirements document



Resources

Equipment	Tools	Materials
<ul style="list-style-type: none">• Computer	<ul style="list-style-type: none">• Microsoft Office• Visual paradigm• E- Draw• Browser	<ul style="list-style-type: none">• Internet• Electricity



Indicative content 1.1: Gathering FURPS Requirements



Duration: 5hrs



Theoretical Activity 1.1.1: Description of FURPS key terms in Backend System



Tasks:

1: Answer the following questions:

- i. Define the following terms as applied in backend system
 - a) Backend
 - b) System
 - c) Database
 - d) Operating system
 - e) System development lifecycle (SDLC)
 - f) Framework
 - g) JSON
 - h) UML
 - i) FURPS
- ii. What are the five categories of software requirements defined by FURPS?
- iii. Differentiate Server from API

2: Write your findings on papers

3: Present your findings to the trainer and colleagues

4: Ask for clarification if necessary

5: Read Key reading 1.1.1 in the manual



Key readings 1.1.1. Description of FURPS key terms in Backend System

1.Description of Key Terms of backend system

1.1 Backend

The backend refers to the server-side of an application where data is processed, and business logic is executed. It includes server technologies, databases, and other components that are not directly seen by end-users.

What does is Back-End System development mean?

Back-end development refers to the development of server-side logic that powers websites and apps from behind the scenes. It includes all the code needed to build out the database, server, and application. From database migrations to API integrations to setting up the server-side technologies that make a website tick, a

back-end web developer may be the talent you need to get your next web project off the ground.

1.2: System

A system is a set of interconnected components working together to achieve a common goal. In the context of software, a system comprises both frontend (user

Interface) and backend components.

It is an organized set of doctrines, ideas, or principles usually intended to explain the arrangement or working of a systematic whole.

Systems management is the administration of the information technology (IT) systems in an enterprise network or data center. An effective systems management plan facilitates the delivery of IT as a service and allows an organization's employees to respond quickly to changing business requirements and system activity.

Example: Shop Management systems and E-learning systems

1.3: Database

A database is an organized collection of data that is stored and accessed electronically. Databases are designed to efficiently manage, store, retrieve, and manipulate data, and they are used in a wide variety of applications, from simple record-keeping systems to complex systems that manage vast amounts of information

1.4: Operating System

The operating system is software that manages hardware and other software on a computer. It provides services for computer programs and acts as an intermediary between users and the computer hardware.

An operating system(OS) is system software that manages computer hardware and software

resources, and provides common services for computer programs

1.5: The systems development life cycle (SDLC):

The systems development life cycle (SDLC) is the overall process for developing information systems from planning and analysis through implementation and maintenance. The SDLC is the foundation for all systems development methodologies and there are literally hundreds of different activities associated with each phase in the SDLC.

Typical activities include determining budgets, gathering system requirements, and writing detailed user documentation. The activities performed during each systems development project will vary. The SDLC begins with a business need, followed by an assessment of the functions a system must have to satisfy the need, and ends when the benefits of the system no longer outweigh its maintenance costs.

This is why it is referred to as a 'lifecycle'. The SDLC is comprised of seven distinct phases: planning, analysis, design, development, testing, implementation, and maintenance. This section takes a detailed look at a few of the more common activities performed during the phases of the systems development life cycle along with common issues facing software development projects .

1.6:A framework:

A framework is a pre-built, reusable set of tools, libraries, and conventions that provide a structured way to develop, organize, and deploy backend applications. Frameworks help streamline the development process by offering standard components and features, reducing the need to write repetitive code and allowing developers to focus on the unique aspects of their applications.

1.7:JSON (JavaScript Object Notation):

JSON (JavaScript Object Notation) is a lightweight data interchange format that is widely used in backend system design. It is easy for humans to read and write, and easy for machines to parse and generate. JSON is often used to transmit data between a server and a client, as well as for configuration files, data storage, and APIs.

JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).

1.8:UML (Unified Modeling Language):

UML (Unified Modeling Language) is a standardized modeling language used in software engineering to visualize, specify, construct, and document the components of a system. In backend system design, UML helps developers and architects communicate complex ideas clearly and ensures that all stakeholders have a shared understanding of the system.

1.9 FURPS

FURPS stands for Functionality, Usability, Reliability, Performance, and Supportability.

2. Five categories of software requirements defined by FURPS

It is used to categorize software requirements into these five key areas to ensure a comprehensive assessment of a system's needs.

- ✓ **Functionality** covers features and capabilities,
- ✓ **Usability** refers to user interface and experience,
- ✓ **Reliability** relates to system stability and fault tolerance,
- ✓ **Performance** focuses on speed and efficiency, and
- ✓ **Supportability** involves maintainability and documentation.

3.API(Application Programming Interface) and Server

An API is a set of rules and protocols that allows one software application to

interact with another. It defines how software components should communicate, making it easier for developers to integrate different systems.

A server is a computer or software that provides services or resources to other computers or clients in a network. In web development, servers handle requests from clients and manage data processing.

Database servers, print servers, mail servers, file servers, application servers, web servers, and game servers are a few examples of servers.



Theoretical Activity 1.1.2: Description of System development life cycle (SDLC) models



Tasks:

- 1: Answer the following questions:
 - i. Explain briefly the Phases of System development life cycle (SDLC)
 - ii. Describe SDLCs models
- 2: Write your findings on papers
- 3: Present your findings to the trainer and colleagues
- 4: Ask for clarification if necessary
- 5: Read Key reading 1.1.2 in trainee manual



Key readings 1.1.2 Description of System development life cycle (SDLC) models

1. System Development Life Cycle (SDLC) Phases

Software Development life cycle (SDLC) is a spiritual model used in project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.

Planning

Identifying project scope, goals, risks, and resources required. Initial project planning and feasibility analysis take place in this phase.

Analysis:

Understanding and defining system requirements. This phase involves studying existing systems, identifying limitations, and proposing solutions.

Design:

Creating a blueprint for the system based on the requirements gathered. This includes architecture, user interfaces, database design, and overall system structure.

Implementation (Coding):

The actual coding of the system based on the design specifications.

Developers write code and build the system according to the approved design.

Testing:

Systematic testing of the developed system to identify and fix defects. Various testing methods, such as unit testing, integration testing, and user acceptance testing, are performed.

Deployment:

Releasing the system for public use. This involves installing the system on servers, configuring it for production, and making it available to end-users.

Maintenance:

Post-deployment activities, including ongoing support, bug fixes, updates, and improvements based on user feedback and changing requirements.

2.SDLC Models

There are different software development life cycle models specify and design, which are followed during the software development phase. These models are also called "Software Development Process Models." Each process model follows a series of phase unique to its type to ensure success in the step of software development.

Here, are some important phases of SDLC life cycle:

2.1 Waterfall Model

The waterfall is a universally accepted SDLC model. In this method, the whole process of software development is divided into various phases.

The waterfall model is a continuous software development model in which development is seen as flowing steadily downwards (like a waterfall) through the steps of requirements analysis, design, implementation, testing (validation), integration, and maintenance.

Linear ordering of activities has some significant consequences. First, to identify the end of a phase and the beginning of the next, some certification techniques have to be employed at the end of each step. Some verification and validation usually do this mean that will ensure that the output of the stage is consistent with its input (which is the output of the previous step), and that the output of the stage is consistent with the overall requirements of the system.



2.2 RAD Model

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets developing software in a short period. The RAD model is based on the concept that a better system can be developed in lesser time by using focus groups to gather system requirements.

- Business Modeling
- Data Modeling
- Process Modeling
- Application Generation
- Testing and Turnover

2.3 Spiral Model

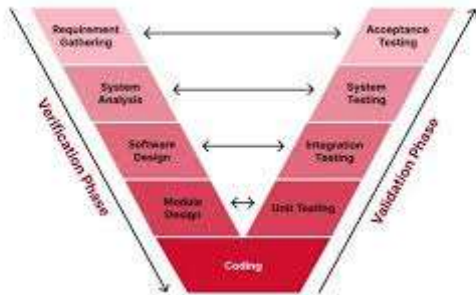
The spiral model is a risk-driven process model. This SDLC model helps the group to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc. The spiral technique is a combination of rapid prototyping and concurrency in design and development activities.

Each cycle in the spiral begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the goals, and the constraints that exist. This is the first quadrant of the cycle (upper-left quadrant). The next step in the cycle is to evaluate these different alternatives based on the objectives and constraints. The focus of evaluation in this step is based on the risk perception for the project.

The next step is to develop strategies that solve uncertainties and risks. This step may involve activities such as benchmarking, simulation, and prototyping.

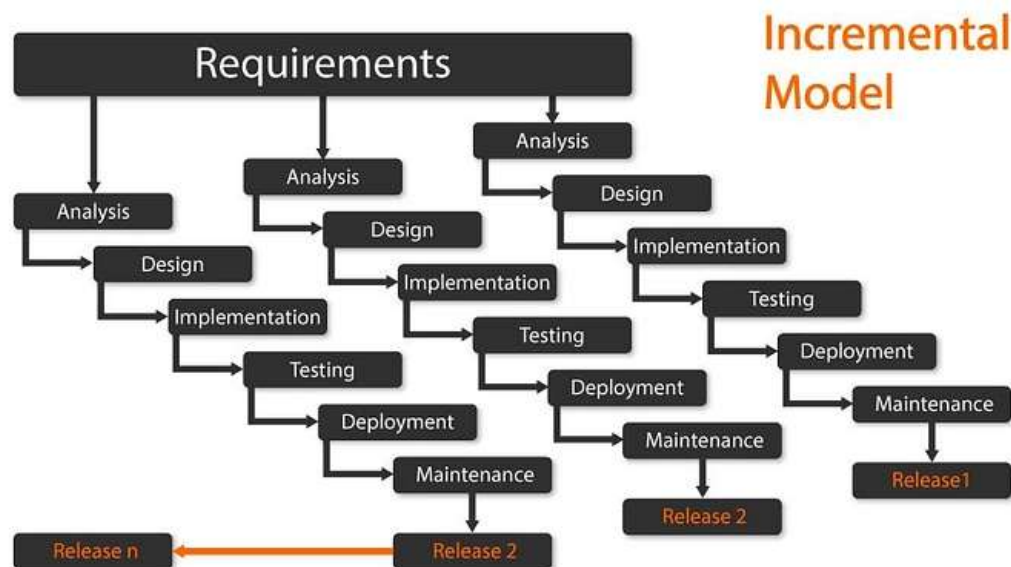
2.4 V-Model

In this type of SDLC model testing and the development, the step is planned in parallel. So, there are verification phases on the side and the validation phase on the other side. V-Model joins by Coding phase.



2.5 Incremental Model

The incremental model is not a separate model. It is necessarily a series of waterfall cycles. The requirements are divided into groups at the start of the project. For each group, the SDLC model is followed to develop software. The SDLC process is repeated, with each release adding more functionality until all requirements are met. In this method, each cycle act as the maintenance phase for the previous software release. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.



2.6 Agile Model

Agile methodology is a practice which promotes continues interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks.

Any agile software phase is characterized in a manner that addresses several key assumptions about the bulk of software projects:

1. It is difficult to think in advance which software requirements will persist and which will change. It is equally difficult to predict how user priorities will change as the project proceeds.

2. For many types of software, design and development are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to think about how much design is necessary before construction is used to test the configuration.

3. Analysis, design, development, and testing are not as predictable (from a planning point of view) as we might like.



2.7 Iterative Model

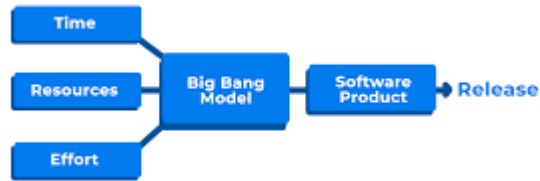
It is a particular implementation of a software development life cycle that focuses on an initial, simplified implementation, which then progressively gains more complexity and a broader feature set until the final system is complete. In short, iterative development is a way of breaking down the software development of a large application into smaller pieces.



2.8 Big bang model

Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come.

This model works best for small projects with smaller size development team which are working together. It is also useful for academic software development projects. It is an ideal model where requirements are either unknown or final release date is not given.



Theoretical Activity 1.1.3: Description of backend development technologies



Tasks:

- 1: Answer the following questions:
 - i. Describe backend development technologies
 - ii. List and explain system analysis tools
- 2: Write your findings on papers
- 3: Present your findings to the trainer and colleagues
- 4: Ask for clarification if any.
- 5: Read Key reading 1.1.3 in trainee manual



Key readings 1.1.3: Description of backend development technologies and analysis tools

1. Backend development Technologies

Backend technologies refer to the tools, frameworks, languages, and databases used to build the server side of web applications.

Framework

A programming framework is a set of reusable software components that provide a foundation for building applications. It includes pre-written code, libraries, and tools that help developers create software applications more efficiently. Therefore, consulting firms often recommend using the most popular ones to reduce development costs and speed up the time to market.

1.1 Python and its Frameworks

Python is one of the most popular and effective programming languages that contain vast libraries and frameworks for almost every technical domain. Python frameworks automate the implementation of several tasks and give developers a structure for application development.

Top 5 Frameworks In Python

There are a number of Python frameworks available in the market for web development. Depending on the functionality and key features they provide to the user, Here are the top 5 frameworks available in Python:

Django

Django is a popular open-source full-stack Python framework that includes all the necessary Python features by default.

Web2Py

Web2Py is another popular open-source and full-stack Python framework. It is platform independent, which means that it can run on all the popular operating systems.

Flask

Flask is a micro-framework for Python. It is lightweight and easily adaptable to suit a developer's needs. The Flask framework comes under the BSD license and requires the Werkzeug WSGI toolkit and Jinja2 templates.

Bottle

Bottle is a micro-framework for prototyping and building simple personal applications. It was originally meant for building APIs and is considered by developers as one of the finest Python web frameworks. It also allows developers to work closely with the hardware to build small and simplistic personal use apps.

CherryPy

CherryPy is an open-source Python framework that follows a minimalist approach for building web applications. Released in 2002, it is one of the oldest Python frameworks still popular today. Unlike other frameworks, you don't need to install the apache server to run CherryPy. The best thing about this framework is that it allows you to use any type of technology for creating templates and data access.

1.2 PHP and its Frameworks

A PHP framework is built on the open-source language PHP and offers built-in features to improve your software development lifecycle. These features increase development speed and improve performance within your team by lowering the workload of your developers.

1. Laravel

On top of the list is Laravel. It is among one of the most widely used PHP frameworks. It offers various benefits for businesses, including its robust security to authentication features along with various useful libraries. This framework is known for its easy-to-read syntax, powerful features, and developer-friendly environment.

By this, it means that when you hire PHP developers, it allows them to make eye-pleasing web solutions.

Why Use Laravel?

- Its features, like routing, ORM, authentication, caching, and more, simplify complex tasks and promote code reusability.
- Its active community ensures regular updates and excellent documentation.
- Robust testing capabilities ensure bug-free web solutions.

2. CodeIgniter

It is an open-source rapid development framework suitable for developing dynamic web solutions. This PHP framework is known for its simplicity, speed, and small footprints.

Its intuitive MVC pattern simplifies web development and allows you to quickly build dynamic and scalable web applications.

If you are developing a web solution from scratch, this framework will save a lot of time for you.

Why Use CodeIgniter?

- Unlike other frameworks, it is a more lightweight PHP framework.
- Code Igniter offers features like database abstraction, caching, security, and session management, making it suitable for rapid development.
- Its extensive library collection enables developers to perform common tasks effortlessly.

3. Symfony

It is a highly flexible PHP framework that uses a modular component system. Due to its advanced features and ease of building large-scale applications with complex requirements, it is a highly recommended framework.

Major companies like BlaBlaCar, Spotify, Magento, etc., use this robust PHP framework. In 2005, it was launched, and since then, it has gained extensive enlightenment among web developers.

Why use Symfony?

- Symfony is the perfect choice for enterprise-level web projects.
- The reusable components of this framework save time while building web solutions.
- Its modular architecture makes it easier for developers to pick and choose components based on the project's requirements.
- Symfony's strong emphasis on testing and robust community support contribute to its reliability and stability.

4. Cake PHP

Cake PHP was the first PHP MVC framework created, but still, with the evolving time, it is most widely used in web development. It will help you develop web solutions loaded with visually impressive features. On top of that, the framework

emphasizes convention over the configuration approach. By this, it means that it decreases the number of decisions that a Cake PHP development company is required to make without losing flexibility and do not repeat yourself (DRY) principles.

Why use CakePHP?

- With built-in features like ORM, validation, caching, and so on, it simplifies development and promotes code reusability.
- Due to the security features of this framework, it is a good choice for commercial web applications.
- It offers a scaffolding feature that automates code generation for rapid prototyping.

5. Phalcon

It is a full-stack PHP framework and is comparatively different from others. Its source code is written in C and C++ coding styles. But you don't have to worry about that as it is implemented as a web server extension in C language.

This high-performance-oriented PHP framework is popularly known for its speed and low overhead cuts. It is very well-documented and easy to use.

Companies like KingHost, PlaceOnAir, and so on utilizes Phalcon.

Why Use Phalcon?

- Its architecture focuses on performance optimization and minimal resource consumption.
- Its ability to compile its components into PHP extensions, resulting in lightning-fast execution.
- Because of its efficient memory usage, it optimizes the performance of a web application.
- It supports MySQL, PostgreSQL, and SQLite natively.

1.3 JAVA and its Frameworks

Java is proven a predominant programming language, this is the reason for it being the chosen language for the ground-breaking software development jobs.

We will look into the features, advantages, and applications of the following Java Frameworks.

- Spring • Hibernate • Struts • Wicket • GWT • Dropwizard • Play • Vaadin • Blade • Grails

JavaScript and its frameworks

A JavaScript Framework is a very important part of modern web development. There is no one framework for the codes, as every JavaScript framework serves a different purpose. The frameworks are easy to apply as every application or website has some common features. It thus makes the task less laborious and time taking.

1. Angular

It is a framework written in TypeScript and developed by Google. It is an open-source web application framework used for developing single-page applications (SPA).

2. React

React was introduced in 2013 and was developed by Facebook. It is a reusable framework and is used for building interactive user interfaces.

3. Vue.js

Vue.js is a free, open-source JavaScript framework that is a progressive front-end framework. It is approachable, easy to learn, and can be used for both small and large applications.

4. Node.js

It is an open-source runtime environment built to execute JavaScript outside a web browser.

5. Polymer

It is a JavaScript library developed by Google. It is an open-source framework and is well suited for Single Page Applications. The polymer has a vast area of application as it supports both one way and two-way data binding. Its features help developers to create custom elements.

6. Backbone.js

Backbone JavaScript framework is used in the case of Single Page Applications. It is popular due to its features which allow complex functions with comparatively lesser codes. However, debugging could be an issue at times.

1.4 Ruby and its Framework

Ruby is a powerful object-oriented programming language with an emphasis on programmer productivity and simplicity. Ruby is a popular choice in web development due to its simplicity and helpful features.

1. Ruby on Rails

Ruby on Rails is not only one of the most popular Ruby frameworks for web development, it's also one of the most well-known web development frameworks regardless of language.

2. Sinatra

Sinatra is more of a DSL than a full-fledged framework. Sinatra lets you quickly create small web applications with minimal effort.

3. Hanami

Hanami is a Ruby on Rails alternative for building feature-rich, full-stack web apps. Like Rails, it's an MVC framework and supports many features that Rails does, like routing, controllers/actions, models, views, migrations, validations, mailers, and assets.

4. Grape

Sometimes, you might want to add REST API capabilities to an existing web

app—for example, if you have a full-stack Rails web app and you now want to offer a REST API so that you can build an Android app for it.

5. Cuba

Cuba is a micro framework for web development. It's lightweight and offers minimal features.

2. System Analysis Tools

System Analysis, “the process of studying a procedure or business in order to identify its goals and purposes and create systems and procedures that will achieve them in an efficient way.”

Tools and Techniques of System Analysis:

2.1. Grid Charts

Grid charts are used to represent the relationship between two sets of factors in a tabular method. A grid chart analysis is helpful in eradicating unnecessary reports or unnecessary data items from reports. It can also be used for identifying the responsibilities of various managers for a particular sub-system. Grid chart can be very effectively used to trace the flow of various transactions and reports in the organization.

2.2. System Flow Chart

A system flowchart is a pictorial or diagrammatic representation of the logical flow of operations and information of an organization. It depicts the clear relationship between input processing and output considering the entire system. A set of standard symbols are commonly used for the construction of system flow charts.

2.3. Decision Tree

Some decisions involve a series of steps. The outcome of the first decision guides the second; the third decision depends on the outcome of the second, and so on. In such type of situations of decision making uncertainty surrounds each step, so we face uncertainty, piled on uncertainty.

Decision trees are the model to deal with such kind of problems. They are also very important in decision making in a probabilistic situation where various opinions (or alternatives) can be drawn (as if they are the branches of a tree) and the final outcomes can be understood.

2.4. Simulation

The simulation model describes the operation of the system in terms of individual events, components of the system. Mainly, it involves the development of a model which is mostly mathematical in nature rather than directly describing the behavior of the overall system.

In particular, the system is divided into elements whose behavior is predicted in terms of probability distributions.

The inter-relationships between the elements also are built into the model. Thus, simulation provides a means of dividing the model building job into smaller component parts and then

Combining these parts in their natural order and allowing the computer to present the effect of their interaction on each other.

Simulation is nothing more or less than the technique of performing sampling experiments on the model of the system. The experiments are done on the model rather than on the real system itself only because the experiments on the real system would be too inconvenient, expensive and time-consuming.

2.5. Decision Tables

Decision tables are a graphical method of representing a sequence of logical decisions. It is prepared in a tabular form. It lists all possible conditions and associated set of actions. A decision table consists of the four parts-condition stub, condition entries, action stub, and action entries



Theoretical Activity 1.1.4: Description of data gathering and identification FURPS Requirements



Tasks:

1: Answer the following questions:

- I. Explain data gathering process
- II. Identify FURPS Requirement

2: Write your findings on papers

3: Present your findings to the trainer and colleagues

4: Ask for clarification if necessary

5: Read Key reading 1.1.4 in trainee manual



Key readings 1.1.4: Description of data gathering and identification FURPS Requirements

1. Data Gathering

Data gathering is the first and most important step in the research process, regardless of the type of research being conducted. It entails collecting, measuring, and analyzing information about a specific subject and is used by businesses to make informed decisions.

Types of data

Before we can start the discussion on data gathering, we need to review the types of data you can collect. All data can be divided into two categories,

qualitative or quantitative. Further, data can be classified as first, second, or third-party.

❖ **Qualitative data**

This type of data can't be measured or expressed as a number. It's less structured than quantitative data. Qualitative data is information acquired to understand more about a research subject's underlying motivations—answering “how” and “why” questions. It is information that is descriptive in nature and can consist of words, pictures, or symbols, which is why it isn't easily measurable.

Examples of questions that will yield qualitative data are:

How do you feel about using products from XYZ brand?

You indicated that you prefer product A. Why is that your favorite laundry detergent?

❖ **Quantitative data**

Quantitative data is structured and can be analyzed statistically. Expressed in numbers, the data can be used to measure variables. The results are objective and conclusive. Questions used to collect quantitative data are usually “how many,” “how much,” or “how often?”

Quantitative data can be measured by numerical variables, analyzed through statistical methods, and represented in charts and graphs.

Examples of quantitative research questions are:

How often do you purchase laundry detergent?

- Once weekly
- Every two weeks
- Once a month
- Other

First-party data

First-party or primary data is collected directly from your research participants. It's valuable data because it is gathered straight from your sources—which eliminates the issues of misinterpretation and errors. First-party data is the most useful and reliable data for your research.

Common sources of first-party data are:

- Survey responses
- Web analytics
- Social media analytics
- Reviews
- Email analytics
- Interviews
- Focus groups
- Experiments
- Observations

The information you can collect from first-party sources includes demographics, purchasing behaviors, interests, purchasing habits, likes, dislikes, etc.

Second-party data

Second-party or secondary data is data that has already been collected by someone else in the past. It is less reliable because you cannot be certain of the methodology of the data collection. It also was performed with a different hypothesis in mind, so analyses may not align well with your research needs.

Common second-party data sources include:

- Previous research
- Books
- Professional journal publications
- Websites
- Libraries
- Newspapers
- Public records

Second-party data may be collected before primary data to help find knowledge gaps or to augment primary research data.

Third-party data

With third-party data, you're looking at data sets that are put together from various sources. This type of data has usually been gathered by companies that don't have direct relationships with consumers and is often sold on data marketplaces.

1. Data gathering process

1.1 Define Objectives

Clearly outline the purpose of gathering data, specifying what needs to be measured or analyzed. This helps focus the data collection on relevant information.

1.2 Identify Data Sources

Determine where the data will come from. Data sources can be:

Primary Data: Collected directly from original sources through methods like surveys, interviews, observations, or experiments.

Secondary Data: Gathered from existing records, databases, or published materials, such as reports, articles, and public databases.

1.3 Choose Data Collection Methods

Select the appropriate techniques for collecting the data, which may include:

Surveys/Questionnaires: Structured or semi-structured forms with questions for respondents to answer.

Interviews: Direct questioning of individuals or groups to gather qualitative data.

Observations: Monitoring behaviors, events, or conditions in their natural setting.

Experiments: Conducting controlled tests to gather data under specific conditions.

Document/Record Review: Analyzing existing data from books, reports, and online sources.

1.4 Design Data Collection Instruments

Develop tools (e.g., questionnaires, checklists) that are valid and reliable to ensure accurate data collection.

1.5 Pilot Testing

Test the data collection instruments on a small scale to identify any issues and refine them before full-scale data collection.

1.6 Collect Data

Implement the chosen method(s) and begin gathering the actual data. This step requires organization, consistency, and adherence to ethical guidelines, especially when dealing with sensitive information.

1.7 Data Validation

Ensure the data collected is accurate, complete, and consistent. This may involve cross-checking responses or using statistical methods to detect errors or anomalies.

1.8 Store and Organize Data

Properly store the data in a structured format (e.g., databases, spreadsheets) for analysis. Security and confidentiality are important considerations in this step.

1.9 Analyze Data

After collection, the data is analyzed to derive insights, patterns, and trends. This could involve statistical analysis, qualitative coding, or other methods depending on the nature of the data.

1.10 Report Findings

Present the results of the data gathering in a clear and accessible way, through written reports, visualizations, or presentations.

2. Identification of FURPS Requirements

The FURPS model is a framework used for categorizing software quality attributes and requirements. It stands for Functionality, Usability, Reliability, and Performance, along with Supportability. Here's a description of each category:

FURPS Requirements

1. Functionality Requirements

These requirements specify what the system should do. They define the features and capabilities that the software must provide to meet user needs.

Key Aspects:

Features: Specific functionalities that must be implemented (e.g., user authentication, data processing).

Business Rules: Constraints and conditions that govern how the system

operates.

Use Cases: Scenarios detailing how users will interact with the system.

Interoperability: Requirements for how the system integrates with other systems or services.

2. Usability Requirements

Definition: Usability requirements focus on how easy and intuitive the software is for users. They encompass the user experience and interface design.

Key Aspects:

Learnability: How quickly users can learn to use the system.

Efficiency: How effectively users can perform tasks once they are familiar with the system.

Memorability: How easily users can re-engage with the system after a period of not using it.

Satisfaction: Overall user satisfaction with the interface and experience.

3. Reliability Requirements

Definition: Reliability requirements pertain to the system's ability to perform its intended functions consistently over time. This includes its stability and fault tolerance.

Key Aspects:

Availability: The proportion of time the system is operational and accessible.

Failure Rate: The frequency of system failures or errors during operation.

Recoverability: The ability of the system to recover from failures and restore data.

Consistency: The system's performance should be stable and predictable under varying conditions.

4. Performance Requirements

Definition: Performance requirements specify the speed, responsiveness, and overall efficiency of the system. They are crucial for ensuring that the system meets user expectations.

Key Aspects:

Response Time: The time it takes for the system to respond to user inputs or requests.

Throughput: The amount of work the system can handle in a given time period (e.g., transactions per second).

Resource Usage: How efficiently the system utilizes resources like CPU, memory, and bandwidth.

Scalability: The ability of the system to handle increased loads by scaling up or out.



Practical Activity 1.1.5: Conducting data gathering



Task:

- 1: By referring to the activity 1.1.4 and key reading 1.1.5 perform the following task:
Collect all data that are needed to develop a trainee registration system of your school to enrol L3 trainees.
- 2: Follow carefully trainer demonstration how to conduct data gathering
- 3: Perform the given task as trainer demonstrated
- 4: Ask clarifying question if needed
- 5: Ask trainees to read key reading 1.1.5 in their manual



Key readings 1.1.5

Performing data gathering

The data gathering process is a systematic approach to collecting information necessary for analysis and decision-making.

1. Define Objectives

Identify Purpose: Clearly outline what you want to achieve with the data collection.

Specify Questions: Formulate specific questions that the data should answer.

2. Determine Data Requirements

Types of Data: Decide on the types of data needed (qualitative, quantitative).

Sources of Data: Identify where the data will come from (primary or secondary sources).

3. Choose Data Collection Methods

Surveys/Questionnaires: Use structured forms to collect responses from a target group.

Interviews: Conduct one-on-one or group discussions to gather in-depth information.

Observations: Record behaviors or events in their natural setting.

Document Review: Analyze existing records, reports, or literature.

4. Develop Data Collection Instruments

Design Tools: Create the tools needed for data collection, such as surveys or interview guides.

Pilot Testing: Test the instruments on a small sample to identify any issues.

5. Collect Data

Implement Methods: Apply the chosen data collection methods systematically.

Maintain Consistency: Ensure that data is collected uniformly to avoid biases.

6. Data Verification

Check Accuracy: Review the collected data for completeness and correctness.

Address Issues: Resolve any identified inconsistencies or errors.

7. Analyze Data

Data Processing: Organize and prepare the data for analysis.

Statistical Analysis: Apply appropriate statistical methods to interpret the data.

8. Report Findings

Summarize Results: Compile the findings into a clear and concise report.

Visual Representation: Use charts or graphs to illustrate key points.






9. Review and Reflect

Evaluate Process: Assess the effectiveness of the data gathering process.

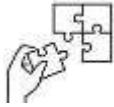
Identify Improvements: Consider how the process can be enhanced for future projects.



Points to Remember

- Data gathering is the first step in research. It involves collecting and analyzing information. Data can be qualitative or quantitative. First-party data is collected directly from sources. Second-party data is collected by others. Third-party data is collected by companies and sold on marketplaces.
- The backend refers to the server-side of web applications, which is built using various technologies, including frameworks, languages, and databases.
- Python frameworks like Django, Flask, and Web2Py are popular for backend development due to their efficiency and ease of use.
- Other popular languages include PHP and Java, each with their own set of frameworks like Laravel, Symfony, and Spring.
- The tools and techniques of system analysis include:
 -  Grid charts
 -  System flowcharts
 -  Decision trees
 -  Simulation
 -  Decision tables.
- Grid charts represent relationships between factors.
- System flowcharts depict the logical flow of operations.
- Decision trees model decision making with uncertainty.

- Simulation involves developing a mathematical model to describe system behavior.
- Decision tables represent logical decisions in a tabular form.
- FURPS is an acronym for functionality, usability, reliability, performance, and security.
 - ✚ Usability requirements relate to user interface and experience.
 - ✚ Reliability requirements focus on system dependability.
 - ✚ Performance requirements concern system efficiency.
 - ✚ Security requirements address system protection



Application of learning 1.1.

XYZ school has a problem of registering new trainees In level 3, normally it use Analog registration form (use paper) then want to change current system to digital registration system as trainee who learn data gathering you are requested to collect all data needed develop digital registration system to enroll new Level 3 trainees



Indicative content 1.2: Identification of Main Objects of Backend System



Duration: 4 hrs



Theoretical Activity 1.2.1: Description the Scope of Backend System



Tasks:

- 1: Answer the following questions:
 - i. Define the Scope of Backend System
 - ii. What do you understand by the followings:
 - a) Database
 - b) APIs
 - c) Servers
 - d) Frameworks
- 2: Write your findings on papers
- 3: Present your findings to the trainer and colleagues
- 4: Ask for clarification if any
- 5: Read Key reading 1.2.1 in trainee manual



Key readings 1.2.1.:

Description the Scope of Backend System

1. Define the Scope of Backend System

Defining the scope of a backend system involves determining the specific functionalities and features that the system will encompass. This includes identifying the data processing, storage, and retrieval capabilities, as well as the integration with other systems and services.

The scope of a backend system can vary depending on the specific requirements of the project, but generally includes tasks such as:

1. Data management and storage
2. User authentication and authorization
3. Business logic implementation
4. Integration with external APIs and services
5. Performance monitoring and optimization
6. Security and compliance measures

2. Database

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS).

Types of databases

There are many different types of databases. The best database for a specific organization depends on how the organization intends to use the data.

Relational databases

- **Relational databases** became dominant in the 1980s. Items in a relational database are organized as a set of tables with columns and rows. Relational database technology provides the most efficient and flexible way to access structured information.

Object-oriented databases

- Information in an object-oriented database is represented in the form of objects, as in object-oriented programming.

Distributed databases

- A distributed database consists of two or more files located in different sites. The database may be stored on multiple computers, located in the same physical location, or scattered over different networks.

Data warehouses

- A central repository for data, a data warehouse is a type of database specifically designed for fast query and analysis.

NoSQL databases

- A NoSQL, or nonrelational database, allows unstructured and semi structured data to be stored and manipulated (in contrast to a relational database, which defines how all data inserted into the database must be composed). NoSQL databases grew popular as web applications became more common and more complex.

Graph databases

- A graph database stores data in terms of entities and the relationships between entities.
- OLTP databases. An OLTP database is a speedy, analytic database designed for large numbers of transactions performed by multiple users.

These are only a few of the several dozen types of databases in use today. Other, less common databases are tailored to very specific scientific, financial, or other functions. In addition to the different database types, changes in technology development approaches and dramatic advances such as the cloud and automation are propelling databases in entirely new directions. Some of the latest databases include

Open-source databases

- An open-source database system is one whose source code is open source; such databases could be SQL or NoSQL databases.

Cloud databases

- A cloud database is a collection of data, either structured or unstructured, that

resides on a private, public, or hybrid cloud computing platform. There are two types of cloud database models: traditional and database as a service (DBaaS). With DBaaS, administrative tasks and maintenance are performed by a service provider.

Multimodel database

- Multimodel databases combine different types of database models into a single, integrated back end. This means they can accommodate various data types.

Document/JSON database

- Designed for storing, retrieving, and managing document-oriented information, document databases are a modern way to store data in JSON format rather than rows and columns.

Self-driving databases

- The newest and most groundbreaking type of database, self-driving databases (also known as autonomous databases) are cloud-based and use machine learning to automate database tuning, security, backups, updates, and other routine management tasks traditionally performed by database administrators.

3.APIs

API stands for Application Programming Interface. In the context of APIs, the word Application refers to any software with a distinct function. Interface can be thought of as a contract of service between two applications.

4.Servers

A server is a computer program or device that provides a service to another computer program and its user, also known as the client.

In a data centre, the physical computer that a server program runs on is also frequently referred to as a server.

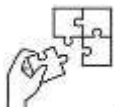
5.Frameworks

Backend frameworks are server-side frameworks designed to make tasks easier for developers. They provide tools, libraries, and other components that help developers create the framework for a website or application. Backend frameworks can automate some aspects of web development, making it faster and simpler.



Points to Remember

- Defining the scope of a backend system involves determining the specific functionalities and features that the system will encompass. This includes identifying the data processing, storage, and retrieval capabilities, as well as the integration with other systems and services.
- A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS).
- API is the acronym for application programming interface— a software intermediary that allows two applications to talk to each other.
- A server is a computer program or device that provides a service to another computer program and its user, also known as the client
- Backend frameworks are server-side frameworks designed to make tasks easier for developers. They provide tools, libraries, and other components that help developers create the framework for a website or application. Backend frameworks can automate some aspects of web development, making it faster and simpler.



Application of learning 1.2

XYZ school has a problem of registering new trainees. In level 3, normally it uses an analog registration form (uses paper) then wants to change the current system to a digital registration system as trainees who learn the main object of backend system design are requested to help your school to identify the main object of the backend system of a digital registration system.



Indicative content 1.3: Description of System Interaction



Duration: 3 hrs



Theoretical Activity 1.3.1: Description of System Interaction



Tasks:

1: Answer the following the questions:

- i. What is Purpose of System Interaction
- ii. Describe the Main components of System Interaction

2: Write your findings on papers

3: Present your findings to the trainer and colleagues

4: Ask for clarification if any

5: Read Key reading 1.3.1 in trainee manual



Key readings 1.3.1. Description of System Interaction

1. Purpose of System Interaction

Interactions between systems are a necessity, a source of opportunity, and a source of difficulty and complication in building, implementing, and maintaining IT-reliant systems in organizations.

2. Main components of System Interaction

Web server

A web server is a computer system capable of delivering web content to end users over the internet via a web browser.

How web servers work

The end user processes a request via a web browser installed on a web server. The communication between a web server or browser and the end user takes place using Hypertext Transfer Protocol (HTTP). The primary role of a web server is to store, process, and deliver requested information or webpages to end users. It uses:

Physical Storage: All website data is stored on a physical web server to ensure its safety. When an end user enters the URL of your website or searches it using a keyword on a browser, a request is generated and sent to the web server to process the data.

Application of webserver

Web servers are primarily used to process and manage HTTP/HTTPS requests and responses from the client system.

A web server can also perform several other functions, such as:

Store and protect website data: A web server can store and protect critical website data from

Unauthorized users.

Control bandwidth to regulate network traffic: A web server can help eliminate the downtime caused by high web traffic. Web hosts can set bandwidth to manage the rate of data transmission over the internet and minimize the excess network traffic.

Server-side web scripting: The server-side web scripting feature enables users to create dynamic web pages using scripting languages such as Ruby, Python, and PHP.

Virtual hosting: Web servers can also be used as virtual servers to run multiple applications, websites, data, and other services.

Web browser: The role of web browsers such as Firefox, Chrome, or Internet Explorer is to find the web server on which your website data is located. Once the browser finds your server, it reads the request and processes the information.

Application Server

An **application server** is a modern form of platform middleware. It is system software that resides between the operating system (OS) on one side, the external resources (such as a database management system [DBMS], communications and Internet services) on another side and the users' applications on the third side.

The function of the application server is to act as host (or container) for the user's business logic while facilitating access to and performance of the business application.

Database Server

A **database server** is a type of hardware that runs database software. Database software helps users or companies' store, manage, retrieve, update or change files, information logs and other forms of digital data.

External Services and API

External services mean services supplied to the Company from external suppliers, including but not limited to, suppliers of Parts, component repair or overhaul, aircraft painting, interior repair, non-destructive testing and logistics.

External Services and **API** refer to the services and interfaces that are provided by third-party applications and platforms to allow developers to integrate their own applications with these services. This allows for greater functionality and interoperability between different systems.

Some examples of popular external services and APIs include:

1. **Google Maps API** - allows developers to integrate Google Maps into their own applications

2. **Facebook API** - provides access to Facebook's social graph and other features for developers

3. **Twitter API** - allows developers to access Twitter's data and functionality

4. **Amazon Web Services** - provides a wide range of cloud-based services and APIs for developers

Message queues or event Streams

They focus on the **delivery** of messages. Once a message is delivered to its supposed recipients, the mission is accomplished, and the message is usually deleted.

Event streaming services

- Messages are stored **persistently**.
- The focus is on the **sequence** of the messages when they are saved. Messages are only appended to the end of the stream.



Points to Remember

- Interactions between systems are a necessity, a source of opportunity, and a source of difficulty and complication in building, implementing, and maintaining IT-reliant systems in organizations.
- Main components of System Interaction: Web server, Application Server, Database Server External Services and API and Message queues or event Streams



Application of learning 1.3

XYZ school has a problem of registering new trainees In level 3, normally it use Analog registration form (use paper) then want to change current system to digital registration system as trainee who learn main object of backend System design you are requested to help your school to identify the system interaction and main components of System Interaction of digital registration system.



Indicative content 1.4: Report of the System Backend Requirements



Duration: 5 hrs



Theoretical Activity 1.4 .1: Reporting the System Backend Requirements.



Tasks:

- 1: Answer this question, what do you understand reporting the System Backend Requirements
- 2: Write their findings on papers
- 3: Present their findings to the trainer and colleagues
- 4: Ask clarifying questions if necessary
- 5: Read Key reading 1.4.1 in trainee manual



Key readings 1.4.1.: Report of the System Backend Requirements.

The report on System Backend Requirements is typically structured into key sections that address different aspects of the system's infrastructure, performance, and areas for improvement.

Here's a detailed breakdown of each section:

1. Executive Summary

Purpose: This section provides a high-level overview of the entire report, summarizing the main findings, conclusions, and recommendations. It's designed to be concise and informative, allowing stakeholders to grasp the essential points without delving into the technical details.

Content: It usually includes a brief background of the project, the objectives of the analysis, a summary of the current state, key gaps or issues identified, and a summary of the proposed recommendations. It may also outline the anticipated impact of implementing the recommendations.

2. Analysis of the Current State

Purpose: This section delves into the existing backend architecture, technologies, processes, and performance metrics.

Content:

Architecture Overview: A description of the current system architecture, including hardware, software, databases, APIs, and other critical components.

Technology Stack: A list and description of the technologies and tools currently in use, such as programming languages, frameworks, and databases.

Performance Metrics: Current performance data, including load times, response times, uptime, and other key performance indicators (KPIs).

Scalability and Reliability: Analysis of the system's ability to handle growth and its reliability under various conditions.

Security Posture: An overview of the current security measures and protocols in place to protect the backend.

3. Findings on Gaps and Issues

Purpose: To identify and discuss any shortcomings, inefficiencies, or risks within the current system.

Content:

Technical Gaps: Areas where the current technology or architecture does not meet the needs or best practices, such as outdated software, insufficient capacity, or lack of integration.

Performance Issues: Specific problems related to system performance, such as slow response times, bottlenecks, or frequent downtimes.

Security Vulnerabilities: Any identified security risks or compliance issues that could expose the system to threats.

Scalability Concerns: Issues that could hinder the system's ability to scale effectively as the user base or data volume grows.

Operational Challenges: Problems in the day-to-day management or maintenance of the backend system, including workflow inefficiencies, lack of automation, or skills gaps within the team.

4. Recommendations

Purpose: To propose actionable solutions to address the gaps and issues identified in the previous section.

Content:

Technical Upgrades: Suggestions for updating or replacing technologies, software, or hardware to improve performance, scalability, or security.

Process Improvements: Recommendations for optimizing operational processes, such as automating routine tasks, improving documentation, or enhancing team collaboration.

Performance Enhancements: Specific strategies to address performance bottlenecks, such as optimizing code, improving database queries, or implementing caching.

Security Enhancements: Steps to strengthen security measures, such as implementing encryption, improving access controls, or conducting regular security audits.

Scalability Solutions: Approaches to ensure the system can scale efficiently, such as adopting cloud services, refactoring code for microservices, or optimizing load balancing.

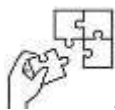
Training and Skill Development: Recommendations for training the team on new technologies or best practices to ensure they can effectively manage and

develop the backend system.



Points to Remember

- **While Reporting the System Backend Requirements you have to follow the following steps:**
 - ✚ Develop executive summary
 - ✚ Detailed analysis of the Current State
 - ✚ Findings on Gaps and issues
 - ✚ Write recommendations



Application of learning 1.4.

The xyz district in Rwanda has been experiencing rapid growth in its student population. The existing student information system (SIS) is outdated and unable to meet the increasing demands of the district's schools. This has led to inefficiencies in various administrative tasks, such as student enrolment, attendance tracking, and grade management. The district has decided to implement a new SIS to address the challenges faced by its schools. The district has commissioned a report on the system's backend requirements, You are asked to make the report of Backend system requirements that can be include Executive Summary, Detailed Analysis of the Current State, Findings on Gaps and Issues, Recommendations.



Learning outcome 1 end assessment

Written assessment

I. Multiple Choice Questions

1. Which of the following is NOT a phase in the System Development Life Cycle (SDLC)?
 - A) Implementation
 - B) Maintenance
 - C) Abstraction
 - D) Testing
2. Which framework is commonly used with Python for backend development?
 - A) Django
 - B) Laravel
 - C) Ruby on Rails
 - D) Spring
3. What does FURPS stand for in requirements gathering?
 - A) Functionality, Usability, Reliability, Performance, Supportability
 - B) Functionality, Usability, Responsiveness, Performance, Scalability
 - C) Functionality, Universality, Reliability, Performance, Supportability
 - D) Flexibility, Usability, Reliability, Performance, Supportability
4. Which of the following is an example of a system analysis tool?
 - A) JSON
 - B) Decision tree
 - C) API
 - D) UML
5. Which of the following is a key component of system interaction?
 - A) Web server
 - B) Functionality requirements
 - C) FURPS
 - D) SDLC

II. True or False Questions

- a) FURPS is an acronym used to describe the phases of the System Development Life Cycle (SDLC).
- b) PHP is a backend development technology that has its own frameworks.
- c) JSON is a data interchange format that is often used in APIs.
- d) A Decision table is a system analysis tool used to represent and analyze complex decision-making scenarios.
- b) The purpose of system interaction is to define the roles and functions of various components within a system.

III. The following table in column A it has the key terms of backend and their definition in column B, Match Key Term with the corresponding definition in the table below in column C

No	Key Terms	Definitions
.....	1.Backend	A) A structured collection of data stored and accessed electronically.
.....	2.FURPS	B) A software environment that provides foundational services for other software.
.....	3.System	C) A set of guidelines, libraries, and tools used to develop software applications.
.....	4.Server	D) The physical or virtual machine that provides services to other computers over a network.
.....	5.Database	E) The process of planning, creating, testing, and deploying an information system.
.....	6.Operating System	F) A method of representing complex systems visually through diagrams.
.....	7.System Development Life Cycle (SDLC)	G) A software component that allows two applications to communicate with each other.
.....	8.API	H) A concept that outlines the requirements for software, focusing on different quality attributes.
.....	9.JSON	I) The part of a system that handles the logic, database interactions, and server-side processing.
.....	10.Framework	J) A data format often used in web APIs for transmitting structured data.
.....	11.UML	K) An organized set of components working together to achieve a common goal.

IV. Open question

- a) What does the "Technology Stack" mean in the report?
- b) How does the report look at backend security?
- c) Why is it important to measure system performance?
- d) What problems can happen in managing backend systems?
- e) What are some examples of problems found in backend systems?
- f) Why is scalability important for a backend system?
- g) What role does training the team play in backend system improvements?

Practical assessment

A school management system (SMS) is a complex application that manages various aspects of a school's operations, including student information, attendance, grading, and communication. By applying the FURPS (Functionality, Usability, Reliability, Performance, and Supportability) framework, we can ensure that the SMS meets the specific needs of the school and provides a high-quality user experience. You are hired to make analysis, identification of FURPS requirements and make the report of FURPS requirements of SMS



References

Books:

- Bird, S. K. (2009). *Natural language processing with Python*. O'Reilly Media.
- Elgendy. (2015). *Business analysis for beginners*. Outskirts Press.
- Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley.
- Hohpe, G. &. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley.
- Kendall, K. E. (2013). *Systems analysis and design*. Pearson.
- Kotonya, G. &. (1998). *Requirements engineering: Processes and techniques*. Wiley.
- McConnell, S. (2004). *A practical handbook of software construction*. Microsoft Press.
- McConnell, S. (2004). *Documenting system backend requirements*. Microsoft Press.
- Pressman, R. S. (2014). *Software engineering*. McGraw-Hill Education.
- Satzinger, J. W. (2012). *Systems analysis and design*. Course Technology, Cengage Learning.
- Sommerville. (2015). *Software engineering*. Pearson.

Web links:

- Ambler, S. W. (n.d.). *Agile requirements change management*. Retrieved from <http://www.agilemodeling.com/essays/requirementsChangeManagement.htm>
- Backend developer*. (n.d.). Retrieved from Guru99: <https://www.guru99.com/backend-developer.html>
- How to gather requirements*. (n.d.). Retrieved from Lucidchart: <https://www.lucidchart.com/blog/how-to-gather-requirements>
- Mozilla Developer Network*. (n.d.). Retrieved from JSON: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>
- Stackify*. (n.d.). Retrieved from What is backend: <https://stackify.com/what-is-backend/>
- Techopedia*. (n.d.). Retrieved from FURPS: <https://www.techopedia.com/definition/25949/furps>
- Tutorials Point*. (n.d.). Retrieved from SDLC overview: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

Learning Outcome 2: Develop System Structure



Indicative contents

2.1 System structure design tools are properly identified based on analysis report and design methodology

2.2 The hardware & software technologies are properly identified based on system requirements

2.3 The system structure is neatly drawn based on the system requirements and design methodology

Key Competencies for Learning Outcome 2: Develop System Structure

Knowledge	Skills	Attitudes
<ul style="list-style-type: none">● Identification of system design tools● Identification of hardware and software technologies● Identification of application of SSADM (Structured System Analysis and Design Methods)● Description of application of Object-Oriented Analysis and Design● Identification of stages for drawing● Description of application Phases in Object-Oriented Software development	<ul style="list-style-type: none">● Implementing stages for drawing SSADM	<ul style="list-style-type: none">● Having Team work● Being critical thinker● Being Innovative● Being attentive.● Being creative● Having curiosity and Open Mindedness



Duration: 31 hrs



Learning outcome 2 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Identify properly system design tools based on analysis report and design methodology
2. Identify properly hardware and software technologies accurately based on system requirements.
3. Identify the application of SSADM (Structured System Analysis and Design Methods) based on the backend system based on system requirements
4. Apply correctly object-Oriented Analysis and Design based on system requirements
5. Implement correctly stages for drawing SSADM based on system requirements
6. Apply correctly Phases in Object-Oriented Software development report and design methodology



Resources

Equipment	Tools	Materials
<ul style="list-style-type: none">• Computer	<ul style="list-style-type: none">• Microsoft Office• Browser• E- Draw• Visual paradigm• Draw io	<ul style="list-style-type: none">• Internet• Electricity



Indicative content 2.1: Identification of System Design Tools



Duration: 5 hrs



Theoretical Activity 2.1.1: Identification of system design tools



Tasks:

1: Answer the following questions:

- i. What is the purpose of system design in software development
- ii. What are the key components involved in the system design process
- iii. What roles do system design tools play in the development process, and what are some examples
- iv. Define the following terms:
 - a) UML (Unified Modelling Language)
 - b) Algorithm
 - c) Flowchart
 - d) Data flow Diagram (DFD)
 - e) Entity Relation diagram (ERD)
 - f) Context diagram
 - g) Decision table
 - h) Use case diagram
 - i) Class Diagram
 - j) Decision tree

2: Provide the answer for the asked questions and write them on papers.

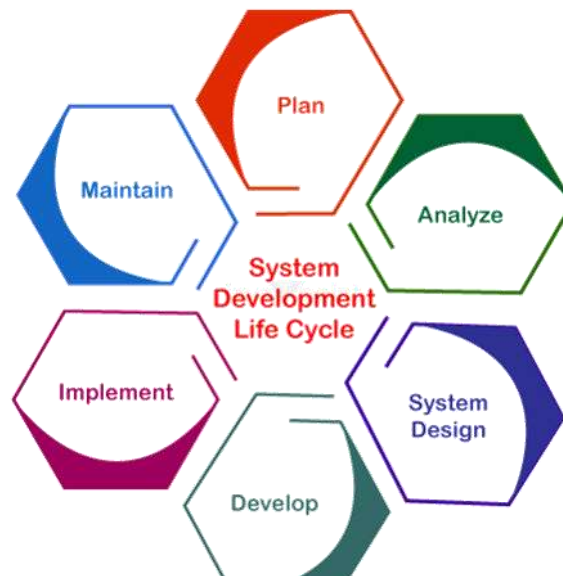
3: Present the findings/answers to the whole class or trainer

4: For more clarification, read the key readings 2.1.1. In addition, ask questions where necessary.



Key readings 2.1.1. Identification of system design tools

System Design: Is the full process of thinking, developing, and creating the architecture, structure, and components of a system to meet specified criteria, functions, or purposes is known as system design. It specifies the structure, behaviour, interfaces, and interactions between the system's different pieces.



The translation of user needs and limitations into a blueprint or Model that specifies how different components or modules of the system will work together cohesively to accomplish the intended results is included in this design phase. While developing the system's development and implementation framework, system designers must consider elements such as performance, scalability, dependability, security, and usability.

The following are important components of system design:

- 1. Understanding and documenting the system's needs,** goals, and restrictions through stakeholder interactions.
- 2. Architectural Design:** Creating the system's high-level structure and components and describing the links and interactions between these aspects.
3. Elaborating on the architectural design by describing precise requirements for individual components, algorithms, databases, interfaces, etc.
4. Building prototypes or models to validate the design, uncover any flaws, and ensure the system's functioning matches the requirements is known as prototyping and testing.

5. Implementation: Creating the system based on the completed design, including coding, integration, and testing, to guarantee it functions properly.

Several major steps are usually involved in the system design process:

1. Requirements analysis is all about Understanding and obtaining needs from stakeholders. This includes determining the system's goals, features, restrictions, and user demands.
2. System Architecture: Creating the system's high-level structure or blueprint. This includes specifying the system's components, their relationships, and how they interact with one another to provide the intended functionality.
3. Precise Design: Extending the system architecture by giving more precise specifications for each component. This involves creating algorithms, databases, interfaces, and other aspects required for system implementation.
4. Building prototypes or models to validate the system's design and functionality. Testing assists in identifying and correcting any potential errors or defects in the design.

The following are common system design principles:

1. Meeting needs: Understanding and fulfilling the needs of stakeholders is the fundamental purpose of system design. This entails collecting and converting user demands, functions, restrictions, and objectives into a system design that completely meets these factors.
2. Scalability refers to the capacity of a system to grow and meet new workloads or user demands without major change or loss of performance. The system's scalability guarantees it can adapt to changing demands and surroundings.
3. Building trustworthy, robust, resilient systems that work consistently and accurately under various scenarios. Systems should be designed to reduce the likelihood of failure and elegantly recover from faults or mistakes.
4. Performance optimization is the process of ensuring that the system operates efficiently and satisfies performance standards. To accomplish ideal execution levels, speed, response times, asset use, and complete framework throughput should be streamlined.
5. Client Experience and Convenience: developing user-friendly, intuitive, and simple technologies.

While planning, it should be important to give clients connection points and communications that address their issues and upgrade their involvement in the framework.

6. Safety efforts are set up to protect the framework against unapproved access, information breaks, and other potential risks. Data integrity, confidentiality, and system availability are all security goals.

System Design Advantages:

1. Efficiency: Well-designed systems are often more efficient, maximizing resource use and enhancing overall performance.
2. Scalability: Properly built systems may grow to handle rising workloads or user demands without requiring considerable reengineering.
3. Reliability: System design seeks to produce dependable and durable systems, lowering the risk of breakdowns and assuring consistent performance.
4. Customization: Systems created with specific requirements may be modified to match individual demands, improving usability and functionality.

System Design Disadvantages:

1. Complexity: Complex system designs can be difficult to understand, lengthening development time and raising the possibility of mistakes.
2. Cost: More resources, in terms of time, money, and experience, may be required to build and maintain elaborate system designs.
3. Over-engineering: Designing a system to be extremely complicated or feature-rich might result in inefficiencies or the use of excessive functions.
4. Rigidity: Without considerable revisions, systems created with specified specifications may fail to adapt to unanticipated changes or developing demands.
5. Dependency: System components in a closely integrated design may become unduly dependent on one another, presenting problems when one fails or has to be modified.

System design tools are essential for planning, visualizing, and structuring software systems.

System design tools

1.UML (Unified Modeling Language):

Overview: UML is a standardized modeling language used to visualize the design

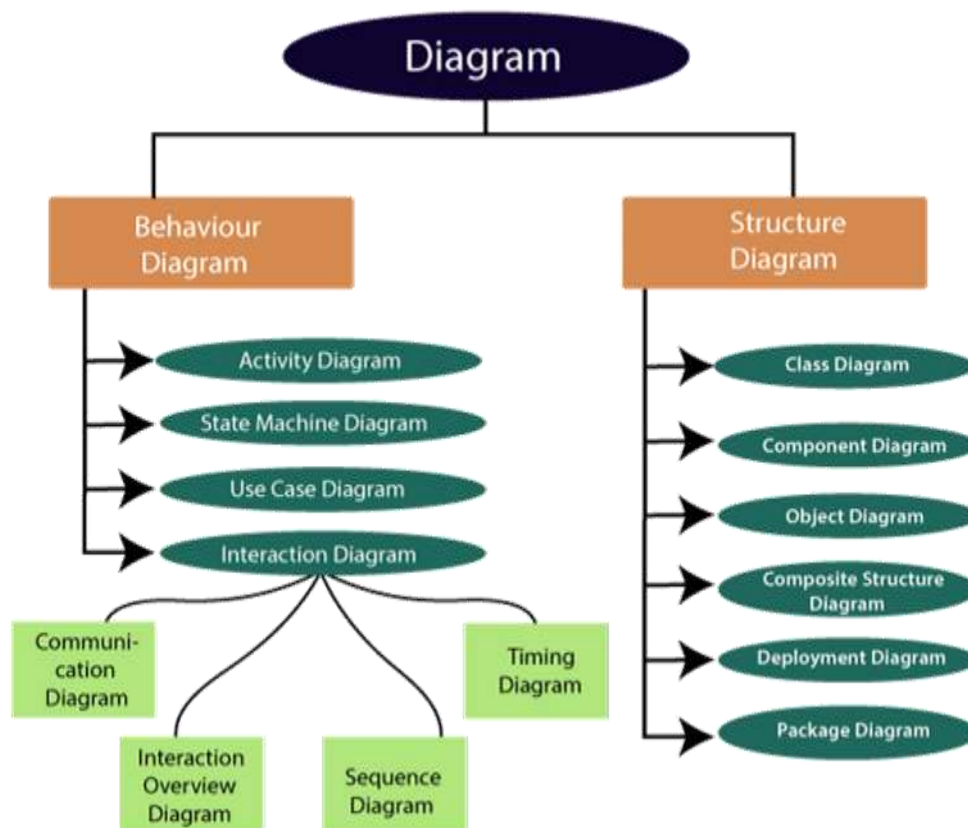
of a system. It includes various diagrams like class diagrams, sequence diagrams, and use case diagrams, providing a comprehensive way to model software systems from different perspectives.

Use Cases: Software architecture design, object-oriented programming, and system blueprinting.

UML tools example: Lucidchart, Draw.io, StarUML, Visual Paradigm, PlantUML, Microsoft Vision ,..... etc

UML-Diagrams

The UML diagrams are categorized into structural diagrams, behavioral diagrams, and also interaction overview diagrams. The diagrams are hierarchically classified in the following figure:



Structural Diagrams

Structural diagrams depict a static view or structure of a system. It is widely used in the documentation of software architecture. It embraces class diagrams, composite structure diagrams, component diagrams, deployment diagrams, object diagrams, and package diagrams. It presents an outline for the system. It stresses the elements to be present that are to be modeled.

Class Diagram: Class diagrams are one of the most widely used diagrams. It is the backbone of all the object-oriented software systems. It depicts the static structure of the system. It displays the system's class, attributes, and methods. It is helpful in recognizing the relation between different objects as well as classes.

Composite Structure Diagram: The composite structure diagrams show parts within the class. It displays the relationship between the parts and their configuration that ascertain the behavior of the class. It makes full use of ports, parts, and connectors to portray the internal structure of a structured classifier. It is similar to class diagrams, just the fact it represents individual parts in a detailed manner when compared with class diagrams.

Object Diagram: It describes the static structure of a system at a particular point in time. It can be used to test the accuracy of class diagrams. It represents distinct instances of classes and the relationship between them at a time.

Component Diagram: It portrays the organization of the physical components within the system. It is used for modeling execution details. It determines whether the desired functional requirements have been considered by the planned development or not, as it depicts the structural relationships between the elements of a software system.

Deployment Diagram: It presents the system's software and its hardware by telling what the existing physical components are and what software components are running on them. It produces information about system software. It is incorporated whenever software is used, distributed, or deployed across multiple machines with dissimilar configurations.

Package Diagram: It is used to illustrate how the packages and their elements are organized. It shows the dependencies between distinct packages. It manages UML diagrams by making it easily understandable. It is used for organizing the class and use case diagrams.

ii. Behavioral Diagrams:

Behavioral diagrams portray a dynamic view of a system or the behavior of a system, which describes the functioning of the system. It includes use case diagrams, state diagrams, and activity diagrams. It defines the interaction within the system.

State Machine Diagram: It is a behavioral diagram. it portrays the system's behavior utilizing finite state transitions. It is also known as the State-charts

diagram.

It models the dynamic behavior of a class in response to external stimuli.

Activity Diagram: It models the flow of control from one activity to the other. With the help of an activity diagram, we can model sequential and concurrent activities. It visually depicts the workflow as well as what causes an event to occur.

Use Case Diagram: It represents the functionality of a system by utilizing actors and use cases. It encapsulates the functional requirement of a system and its association with actors. It portrays the use case view of a system.

iii. Interaction Diagrams

Interaction diagrams are a subclass of behavioral diagrams that give emphasis to object interactions and also depicts the flow between various use case elements of a system. In simple words, it shows how objects interact with each other and how the data flows within them. It consists of communication, interaction overview, sequence, and timing diagrams.

Sequence Diagram: It shows the interactions between the objects in terms of messages exchanged over time. It delineates in what order and how the object functions are in a system.

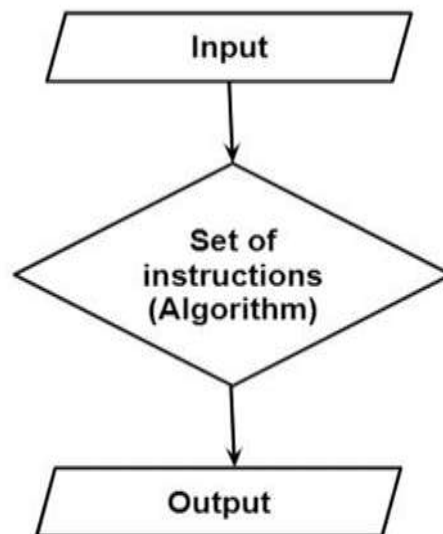
Communication Diagram: It shows the interchange of sequence messages between the objects. It focuses on objects and their relations. It describes the static and dynamic behavior of a system.

Timing Diagram: It is a special kind of sequence diagram used to depict the object's behavior over a specific period of time. It governs the change in state and object behavior by showing the time and duration constraints.

Interaction Overview diagram: It is a mixture of activity and sequence diagram that depicts a sequence of actions to simplify the complex interactions into simple interactions.

An algorithm is a set of defined steps designed to perform a specific objective. This can be a simple process, such as a recipe to bake a cake, or a complex series of operations used in machine learning to analyze large datasets and make predictions.

Programming Algorithm Example:



Characteristics of an Algorithm



You probably wish you could see an example, right? So, what exactly does an algorithm in programming look like? Well, asking a user for an email address is probably one of the most common tasks a web-based program might need to do, so that is what we will use here for an example. An algorithm can be written as a list of steps using text or as a picture with shapes and arrows called a flowchart. We will make one of each which you will see here:

Step 1: Start

Step 2: Create a variable to receive the user's email address

Step 3: Clear the variable in case it's not empty

Step 4: Ask the user for an email address

Step 5: Store the response in the variable

Step 6: Check the stored response to see if it is a valid email address

Step 7: Not valid? Go back to Step 3.

Step 8: End

3.A flowchart:

is a diagram that represents a process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. Here's an example flowchart for a basic login process for a web application:

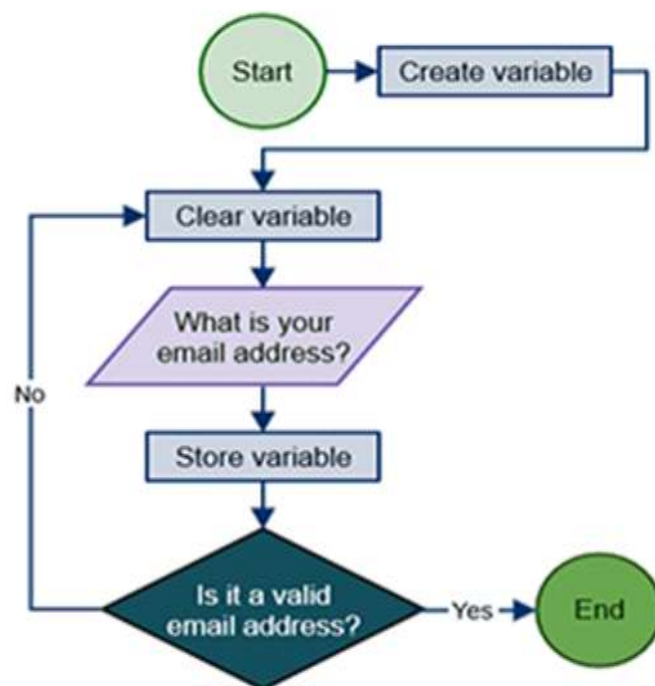


Figure: Flowchart example

4.DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.


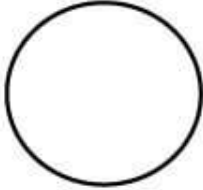

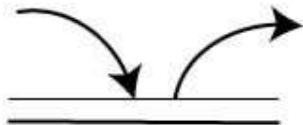
It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

The following observations about DFDs are essential:

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows in a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

Symbol	Name	Function
	Data flow	Used to Connect Processes to each other, to sources or Sinks; the arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Circle: A circle (bubble) shows a process that transforms data inputs into data outputs.

Data Flow: A curved line shows the flow of data into or out of a process or data store.

Data Store: A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

Source or Sink: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

Levels in Data Flow Diagrams (DFD)

The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail.

Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFDM

It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the program at hand is well understood. It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x1 and x2 and one output y, then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

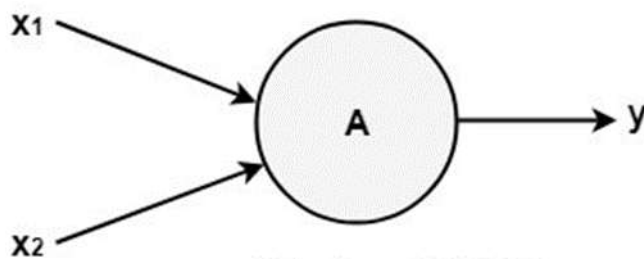


Fig: Level-0 DFD.

The Level 0 DFD: also called Context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed.

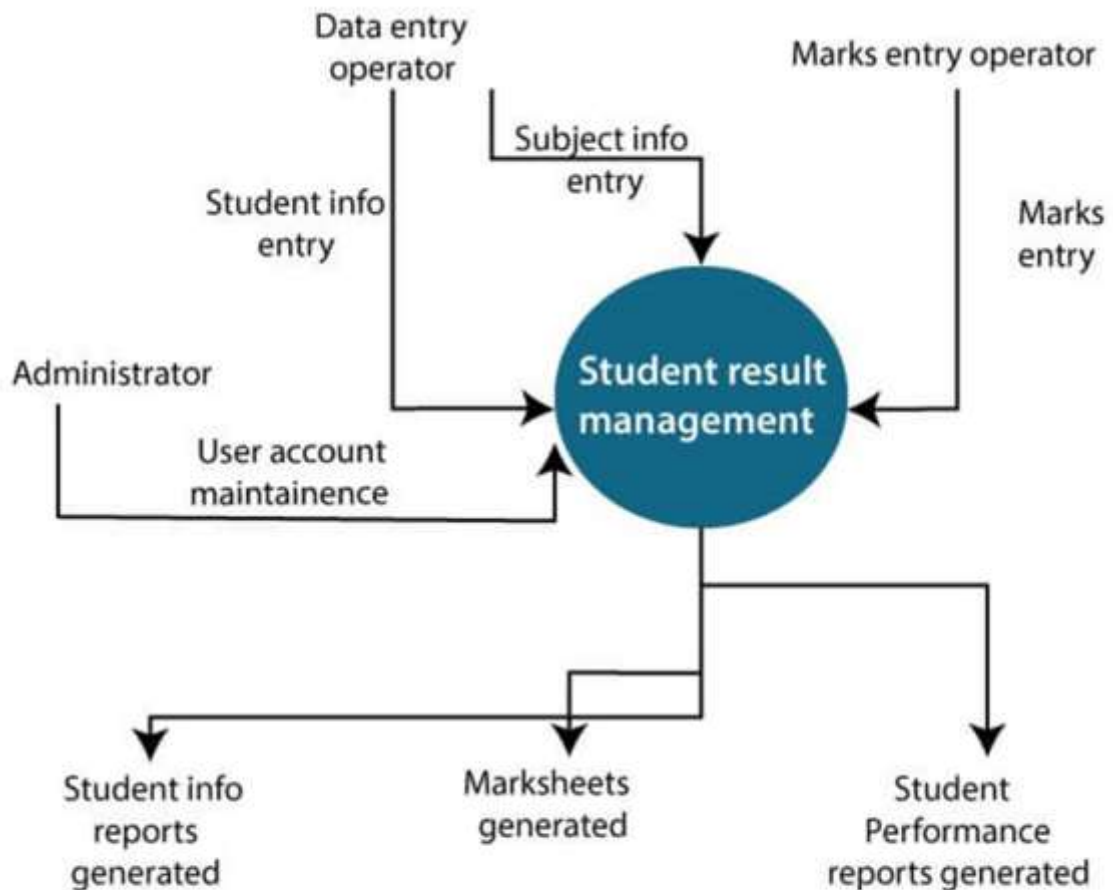


Fig: Level-0 DFD of result management system

1-level DFD

In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into sub processes.

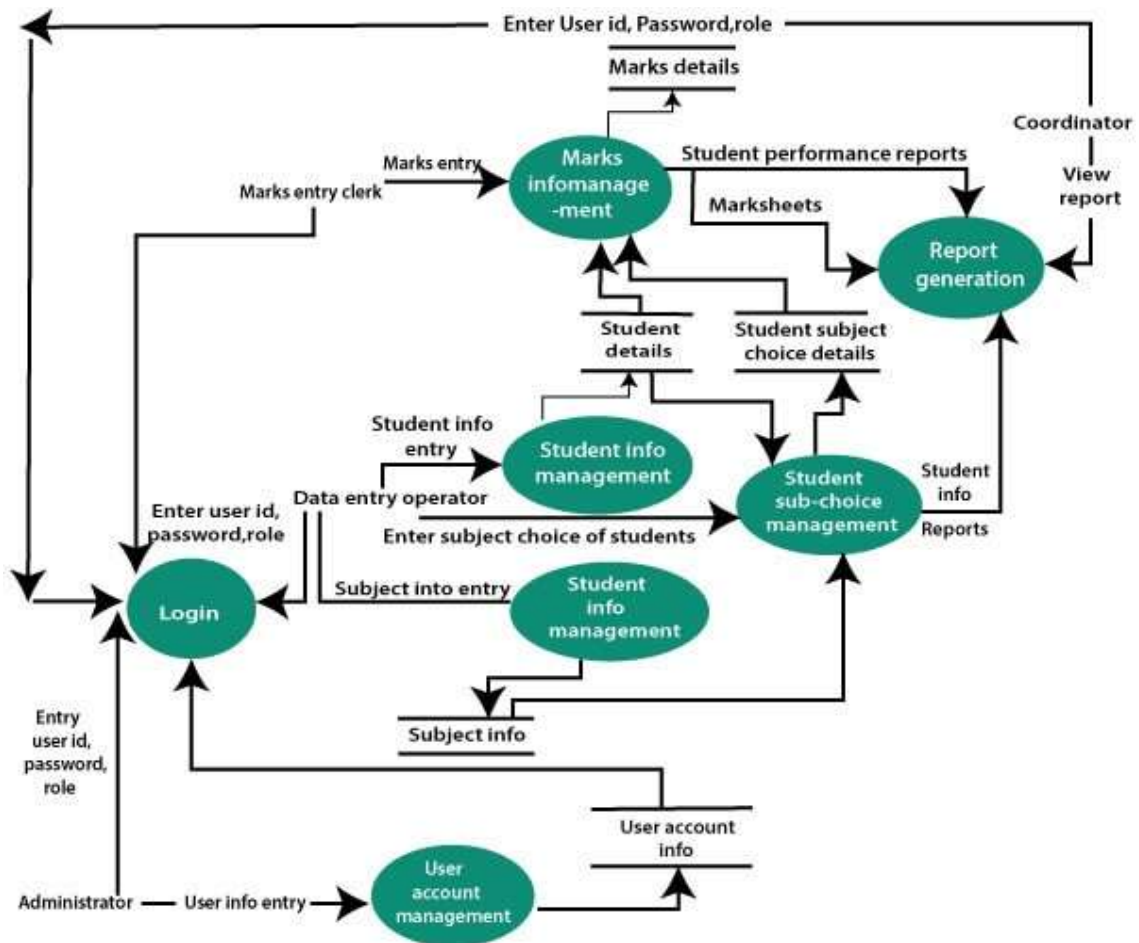
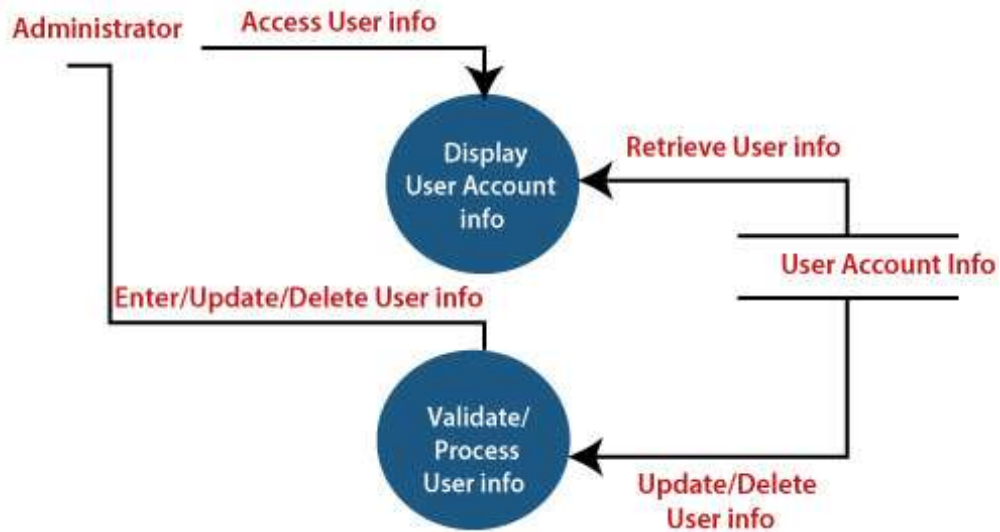


Fig: Level-1 DFD of result management system

2-Level DFD

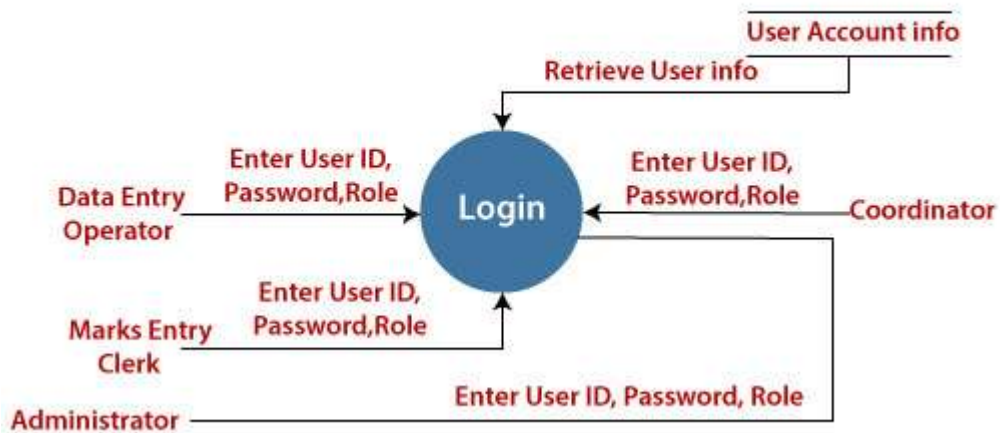
2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

1. User Account Maintenance

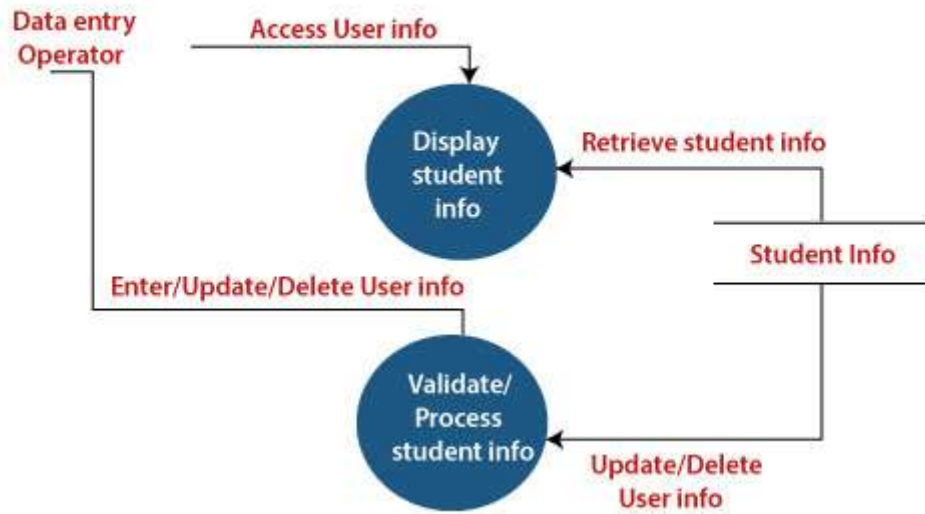


2. Login

The level 2 DFD of this process is given below:

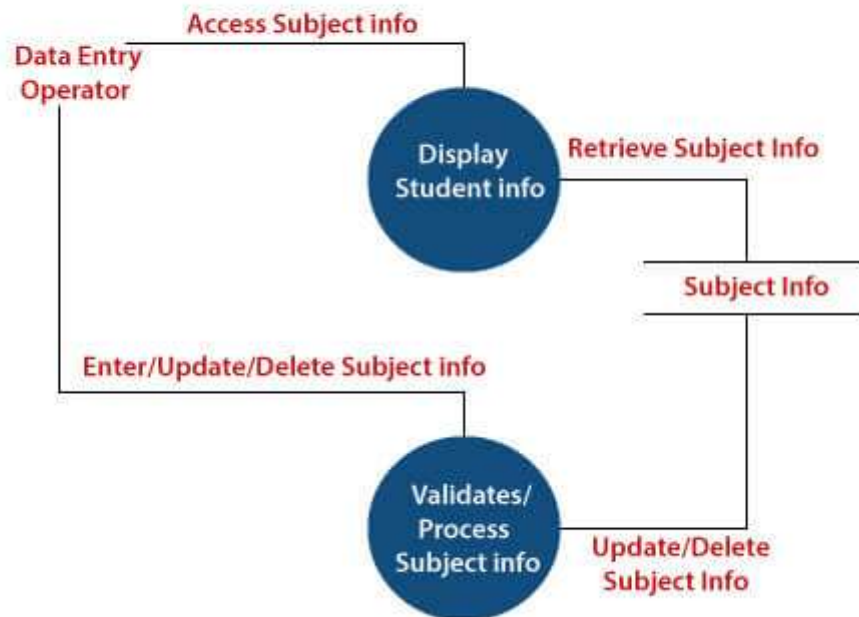


3. Student Information Management



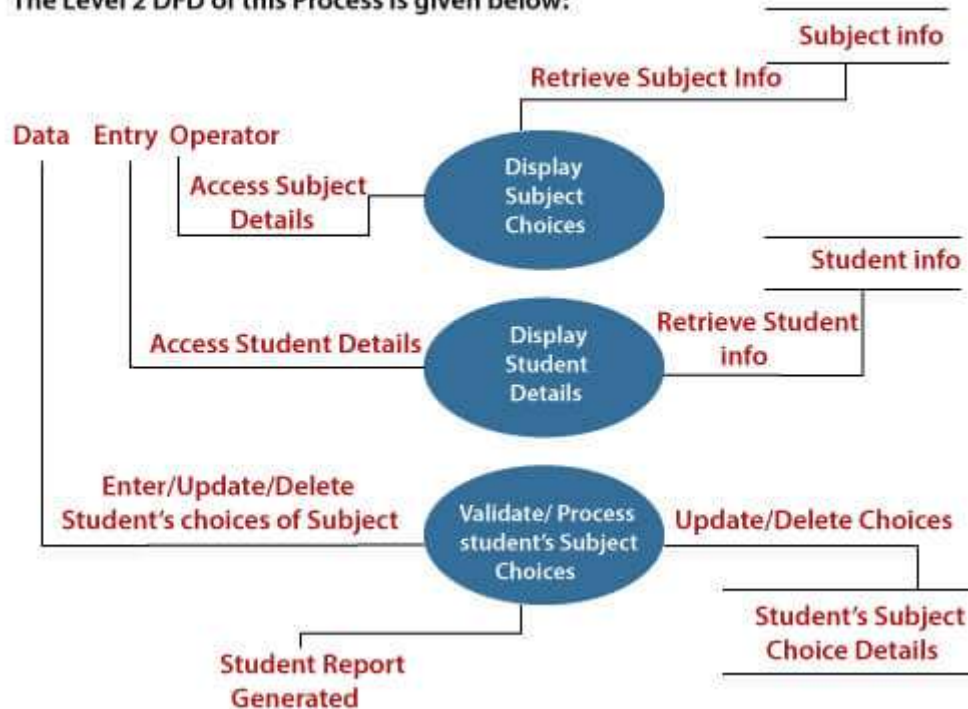
4. Subject Information Management

The level 2 DFD of this process is given below:



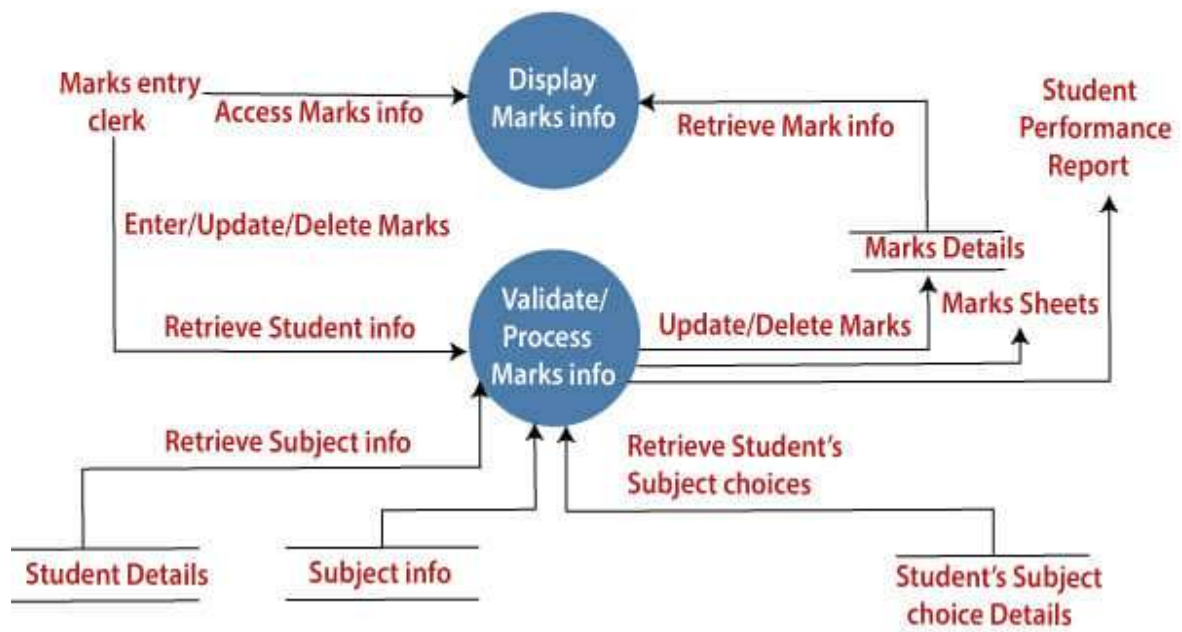
5. Student's Subject Choice Management

The Level 2 DFD of this Process is given below:



6. Marks Information Management

The Level 2 DFD of this Process is given below:



Entity-Relationship Diagrams

ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this

ER-modeling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

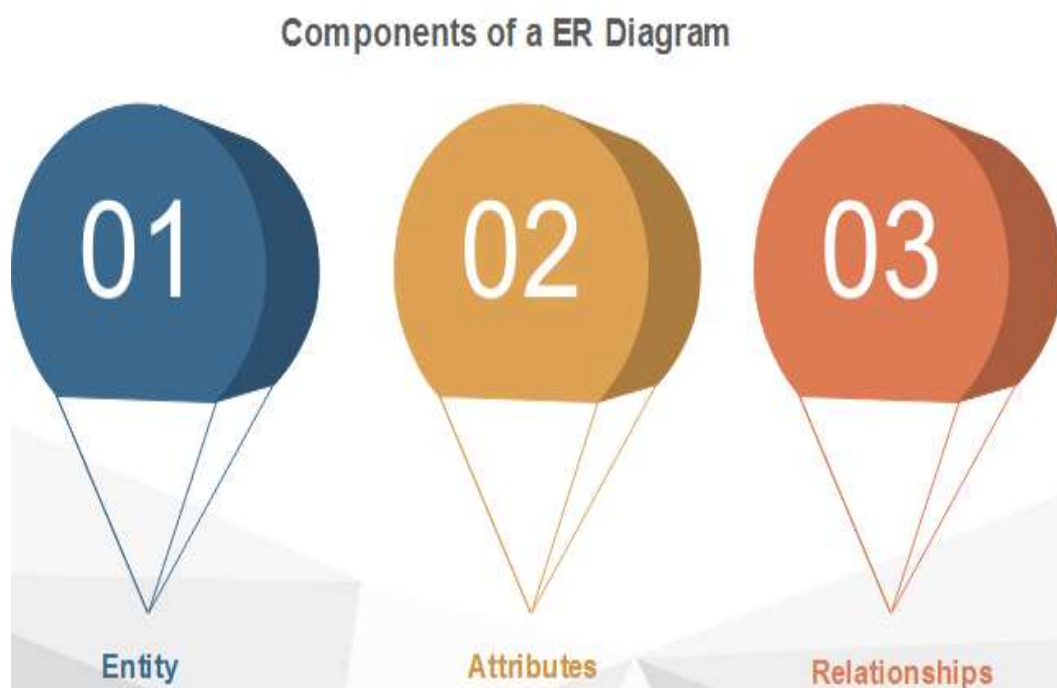
Purpose of ERD

The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.

The ERD serves as a documentation tool.

Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

Components of an ER Diagrams



. Entity

An entity can be a real-world object, either animate or inanimate, that can be merely identifiable. An entity is denoted as a rectangle in an ER diagram. For example, in a school database, students,

teachers, classes, and courses offered can be treated as entities. All these entities have some attributes or properties that give them their identity.

Entity Set

An entity set is a collection of related types of entities. An entity set may include entities with attribute sharing similar values.

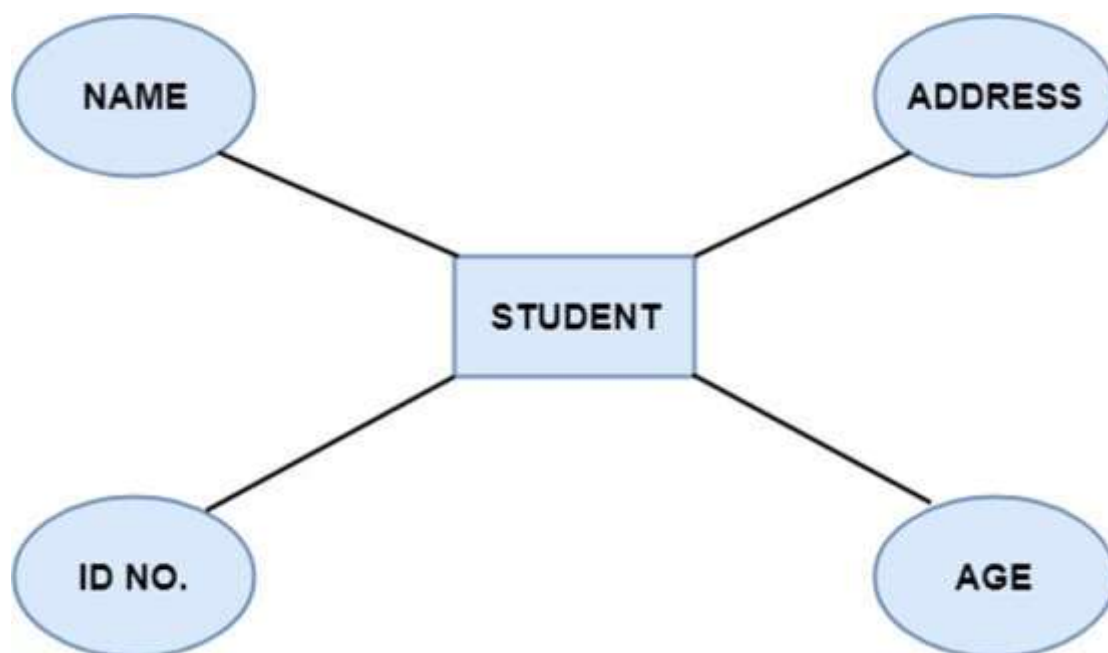
For example, a Student set may contain all the students of a school; likewise, a Teacher set may include all the teachers of a school from all faculties. Entity set need not be disjoint.



Attributes

Entities are denoted utilizing their properties, known as attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

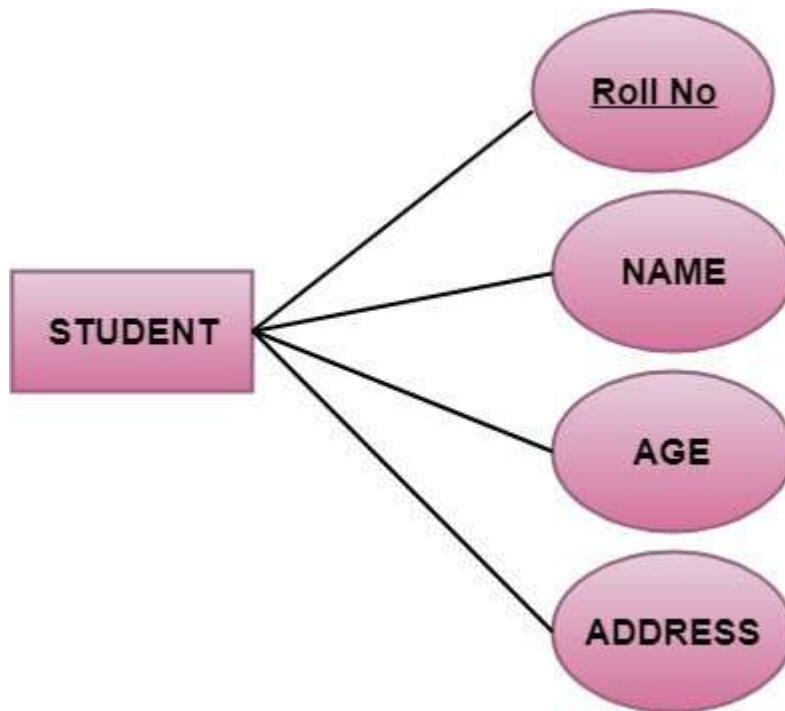
There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



There are four types of Attributes:

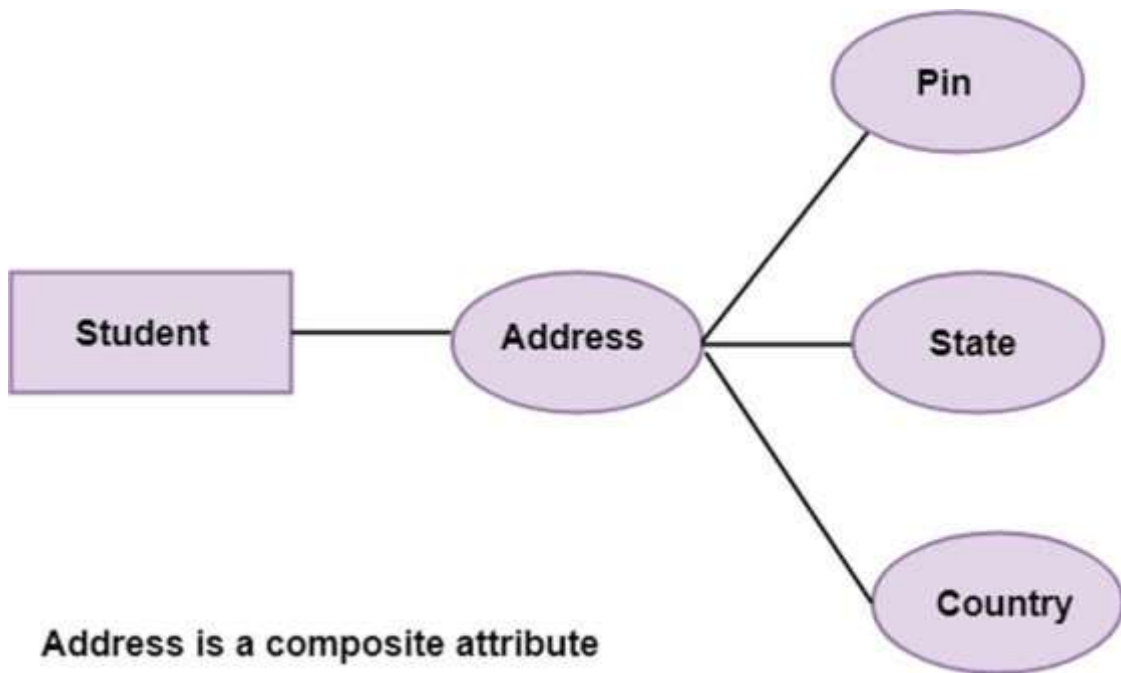
- 1.Key attribute
- 2.Composite attribute
- 3.Single-valued attribute
- 4.Multi-valued attribute
- 5.Derived attribute

Key attribute: Key is an attribute or collection of attributes that uniquely identifies an entity among the entity set. For example, the roll_number of a student makes him identifiable among students.



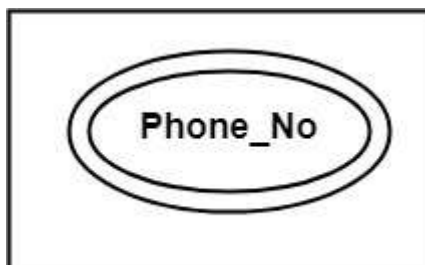
There are mainly three types of keys:

1. **Super key:** A set of attributes that collectively identifies an entity in the entity set.
 2. **Candidate key:** A minimal super key is known as a candidate key. An entity set may have more than one candidate key.
 3. **Primary key:** A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.
2. **Composite attribute:** An attribute that is a combination of other attributes is called a composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other characteristics such as pin code, state, country.

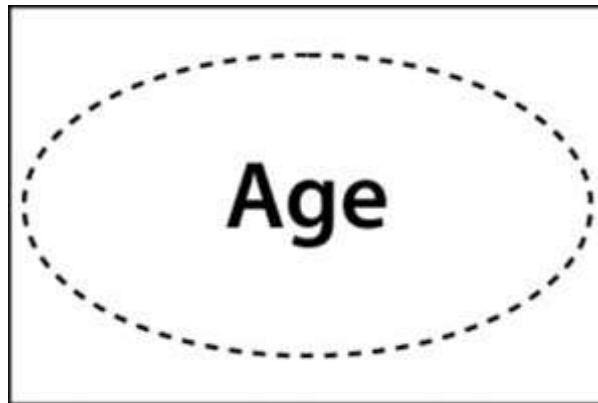


3. Single-valued attribute: Single-valued attribute contain a single value. For example, Social_Security_Number.

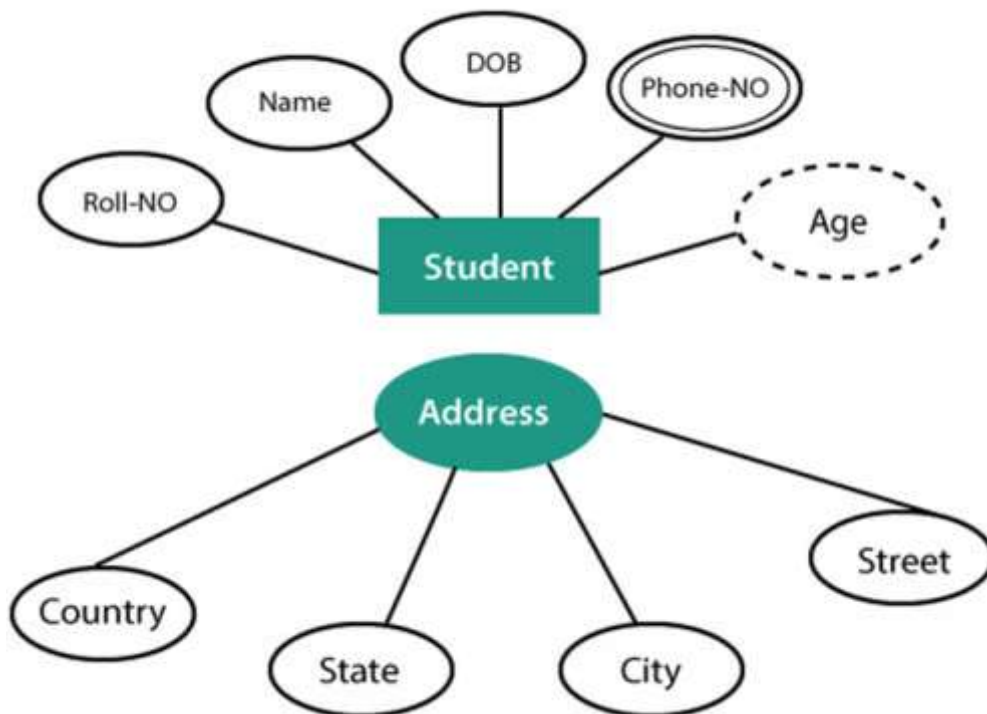
4. Multi-valued Attribute: If an attribute can have more than one value, it is known as a multi-valued attribute. Multi-valued attributes are depicted by the double ellipse. For example, a person can have more than one phone number, email-address, etc.



5. Derived attribute: Derived attributes are the attribute that does not exist in the physical database, but their values are derived from other attributes present in the database. For example, age can be derived from date_of_birth. In the ER diagram, Derived attributes are depicted by the dashed ellipse.



The Complete entity type Student with its attributes can be represented as:



3. Relationships

The association among entities is known as relationship. Relationships are represented by the diamond-shaped box. For example, an employee works_at a department, a student enrolls in a course. Here, Works_at and Enrolls are called relationships.



Fig: Relationships in ERD

6.DECISION TABLE

In UML (Unified Modeling Language), a Decision Table is a tabular representation that captures complex decision logic by mapping conditions to corresponding actions. It helps in clearly defining and documenting business rules. The table typically consists of:

- Conditions: Rows that list all possible scenarios or criteria.
- Actions: Rows that list the possible actions or outcomes.
- Rules: Columns that define specific combinations of conditions and their corresponding actions.

Decision Tables									
		R1	R2	R3	R4	R5	R6	R7	R8
Condition	Printer prints	N	N	N	N	Y	Y	Y	Y
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is recognized by computer	N	Y	N	Y	N	Y	N	Y
Action	Check the power cable			x					
	Check the printer-computer cable	x		x					
	Ensure printer software is installed	x		x		x		x	
	Check/replace ink	x	x			x	x		
	Check for paper jam		x		x				

This method simplifies understanding and ensures all possible decision outcomes are considered, reducing ambiguity in the system's behavior.

Conditions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valid email	0	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1
Email in database	0	0	1	0	0	1	0	0	1	1	0	1	1	0	1	1
User is blocked	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	1
Valid and correct password	0	0	0	0	1	0	0	1	0	1	1	0	1	1	1	1
Action	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Error message	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Login password	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

7. UML Use Case Diagram

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

Purpose of Use Case Diagrams

The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system.

Following are the purposes of a use case diagram given below:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

How to draw a Use Case diagram?

It is essential to analyze the whole system before starting with drawing a use case diagram, and then the system's functionalities are found. And once every single functionality is identified, they are then transformed into the use cases to be used in the use case diagram.

After that, we will enlist the actors that will interact with the system. The actors are the person or a thing that invokes the functionality of a system. It may be a system or a private entity, such that it requires an entity to be pertinent to the functionalities of the system to which it is going to interact.

Once both the actors and use cases are enlisted, the relation between the actor and use case/ system is inspected. It identifies the no of times an actor communicates with the system. Basically, an actor can interact multiple times with a use case or system at a particular instance of time.

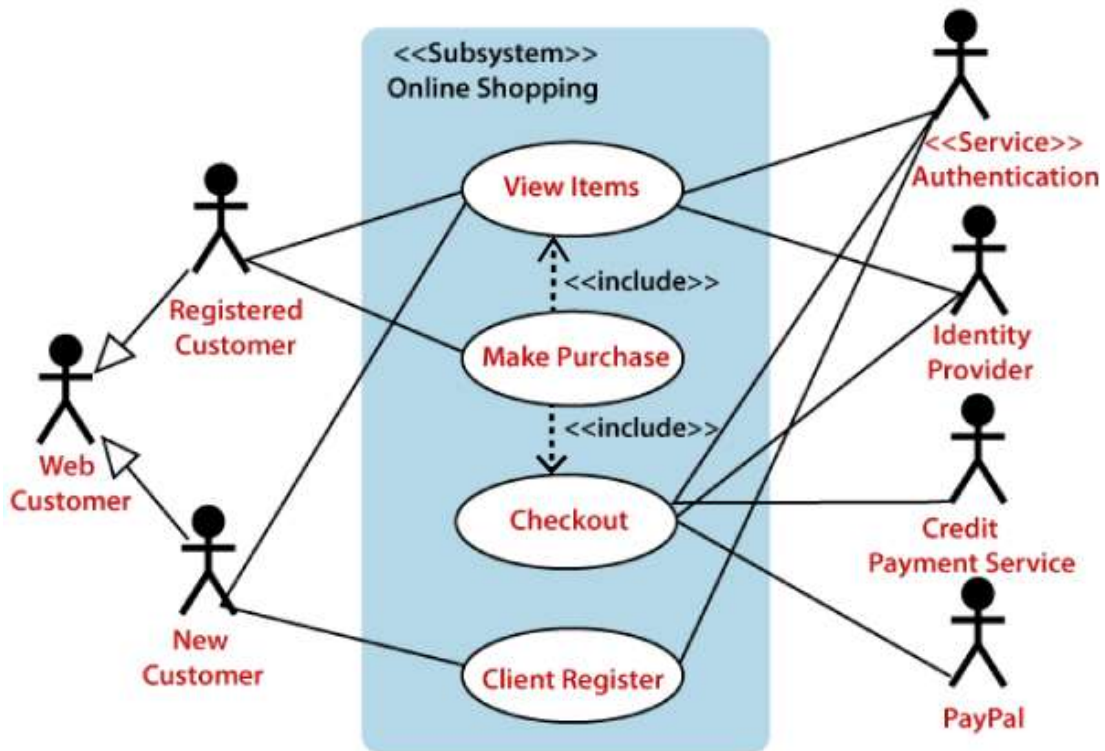
Following are some rules that must be followed while drawing a use case diagram:

1. A pertinent and meaningful name should be assigned to the actor or a use case of a system.
2. The communication of an actor with a use case must be defined in an understandable way.
3. Specified notations to be used as and when required.
4. The most significant interactions should be represented among the multiple no of interactions between the use case and actors.

Example of a Use Case Diagram

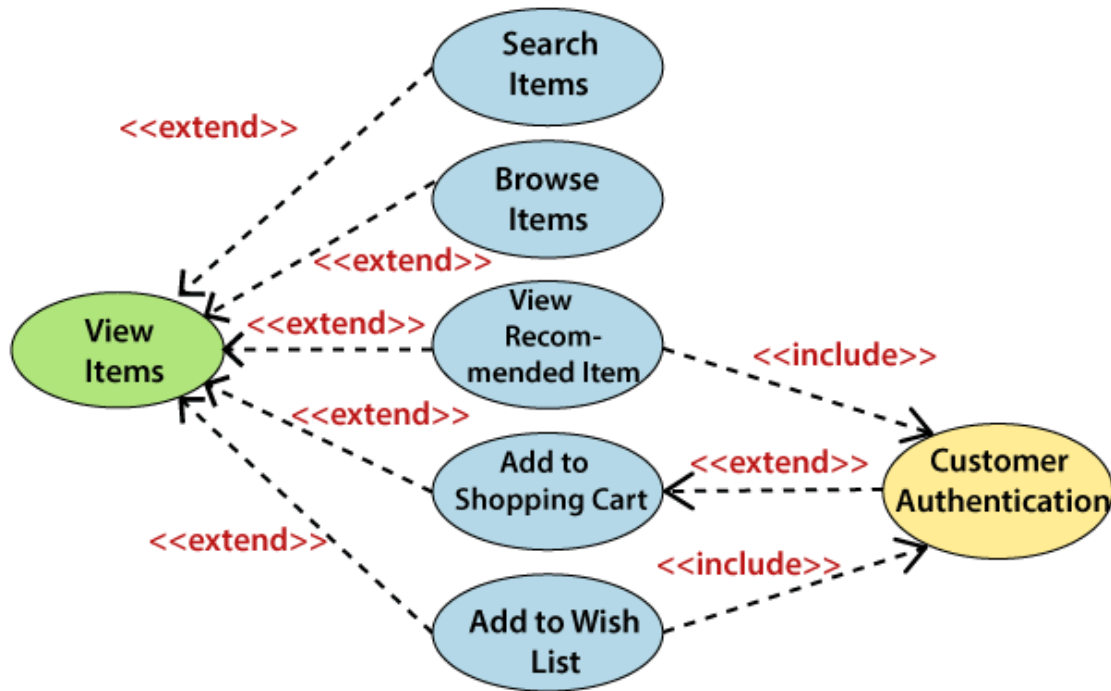
A use case diagram depicting the Online Shopping website is given below.

Here the Web Customer actor makes use of any online shopping website to purchase online. The top-level uses are as follows; View Items, Make Purchase, Checkout, Client Register. The View Items use case is utilized by the customer who searches and view products. The Client Register use case allows the customer to register itself with the website for availing gift vouchers, coupons, or getting a private sale invitation. It is to be noted that the Checkout is an included use case, which is part of Making Purchase, and it is not available by itself.



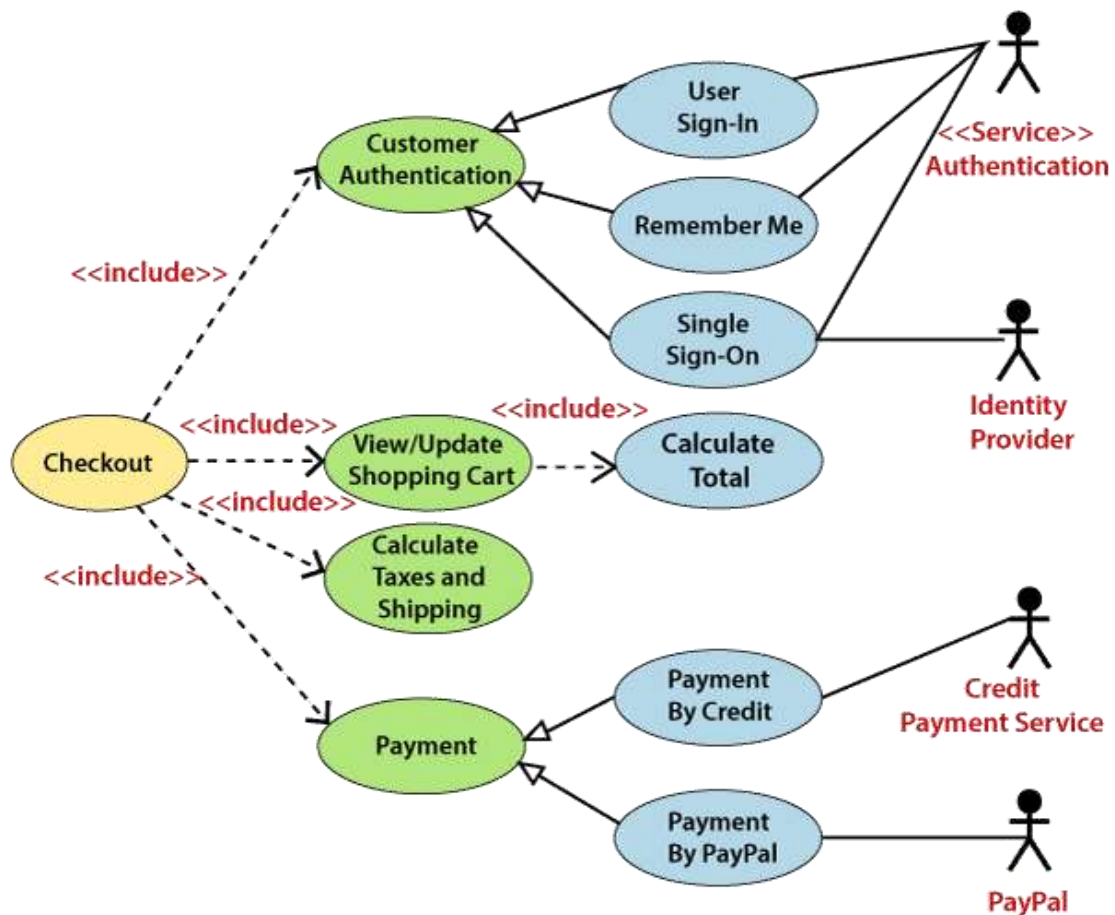
The View Items is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allows them to search for an item. The View Items is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allows them to search for an item.

Both View Recommended Item and Add to Wish List include the Customer Authentication use case, as they necessitate authenticated customers, and simultaneously item can be added to the shopping cart without any user authentication.



Similarly, the Checkout use case also includes the following use cases, as shown below. It requires an authenticated Web Customer, which can be done by login page, user authentication cookie ("Remember me"), or Single Sign-On (SSO). SSO needs an external identity provider's participation, while Web site authentication service is utilized in all these use cases.

The Checkout use case involves Payment use case that can be done either by the credit card and external credit payment services or with PayPal.



Important tips for drawing a Use Case diagram

Following are some important tips that are to be kept in mind while drawing a use case diagram:

1. A simple and complete use case diagram should be articulated.
2. A use case diagram should represent the most significant interaction among the multiple interactions.
3. At least one module of a system should be represented by the use case diagram.
4. If the use case diagram is large and more complex, then it should be drawn more generalized.

8.UML Class Diagram

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes.

A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams

The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

1. It analyses and designs a static view of an application.
2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagrams.
4. It incorporates forward and reverse engineering.

Benefits of Class Diagrams

1. It can represent the object model for complex systems.
2. It reduces the maintenance time by providing an overview of how an application is structured before coding.
3. It provides a general schematic of an application for better understanding.
4. It represents a detailed chart by highlighting the desired code, which is to be programmed.
- 5, It is helpful for the stakeholders and the developers.

Vital components of a Class Diagram

The class diagram is made up of three sections:

Upper Section: The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:

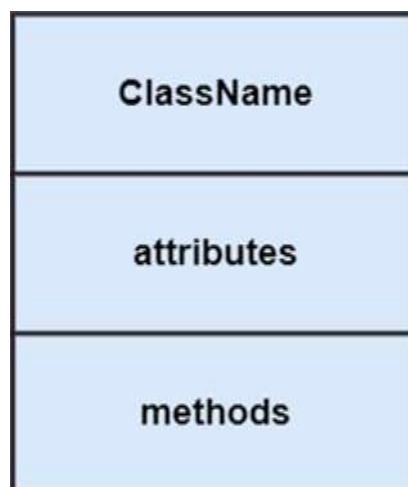
- a. Capitalize the initial letter of the class name.
- b. Place the class name in the center of the upper section.
- c. A class name must be written in bold format.
- d. The name of the abstract class should be written in italics format.

Middle Section: The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:

The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).

- a. The accessibility of an attribute class is illustrated by the visibility factors.
- b. A meaningful name should be assigned to the attribute, which will explain its usage inside the class.

Lower Section: The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.

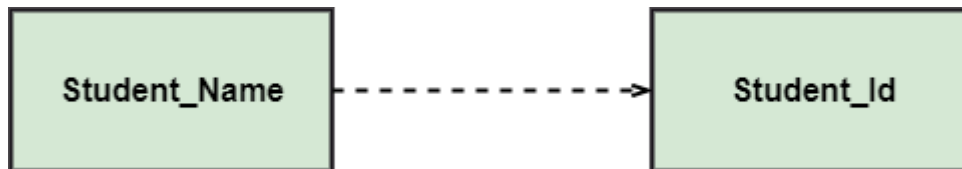


Relationships

In UML, relationships are of three types:

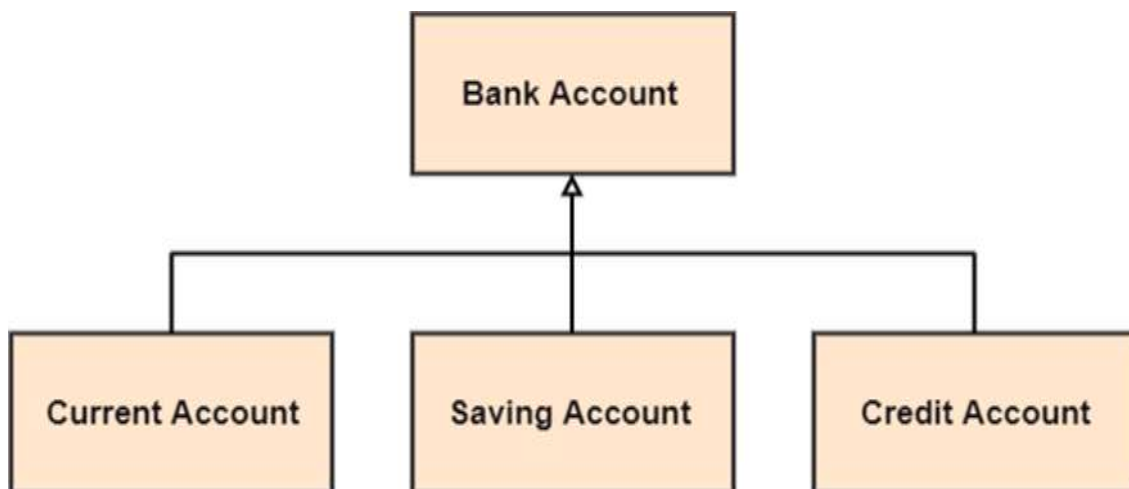
Dependency: A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship.

In the following example, Student_Name is dependent on the Student_Id.



Generalization: A generalization is a relationship between a parent class (superclass) and a child class (subclass). In this, the child class is inherited from the parent class.

For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.



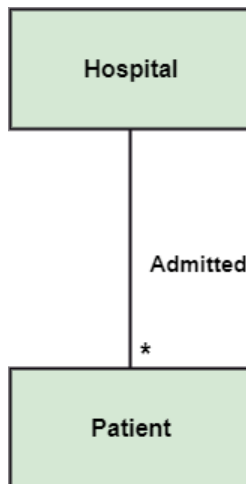
Association: It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship.

For example, a department is associated with the college.



Multiplicity: It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity.

For example, multiple patients are admitted to one hospital.



Aggregation: An aggregation is a subset of association, which represents has a relationship. It is more specific than association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.

The company encompasses a number of employees, and even if one employee resigns, the company still exists.



Composition: The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship.

A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost.

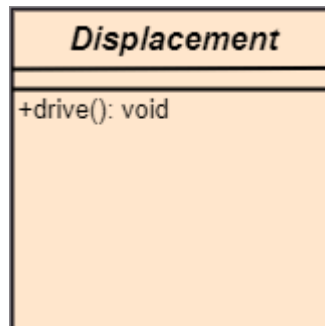


Abstract Classes

In the abstract class, no objects can be a direct entity of the abstract class. The abstract class can neither be declared nor be instantiated. It is used to find the functionalities across the classes. The notation of the abstract class is similar to that of class; the only difference is that the name of the class is written in italics. Since it does not involve any implementation for a given function, it is best to use

the abstract class with multiple objects.

Let us assume that we have an abstract class named displacement with a method declared inside it, and that method will be called as a drive (). Now, this abstract class method can be implemented by any object, for example, car, bike, scooter, cycle, etc.



How to draw a Class Diagram

The class diagram is used most widely to construct software applications. It not only represents a static view of the system but also all the major aspects of an application. A collection of class diagrams as a whole represents a system.

Some key points that are needed to keep in mind while drawing a class diagram are given below:

To describe a complete aspect of the system, it is suggested to give a meaningful name to the class diagram.

The objects and their relationships should be acknowledged in advance.

The attributes and methods (responsibilities) of each class must be known.

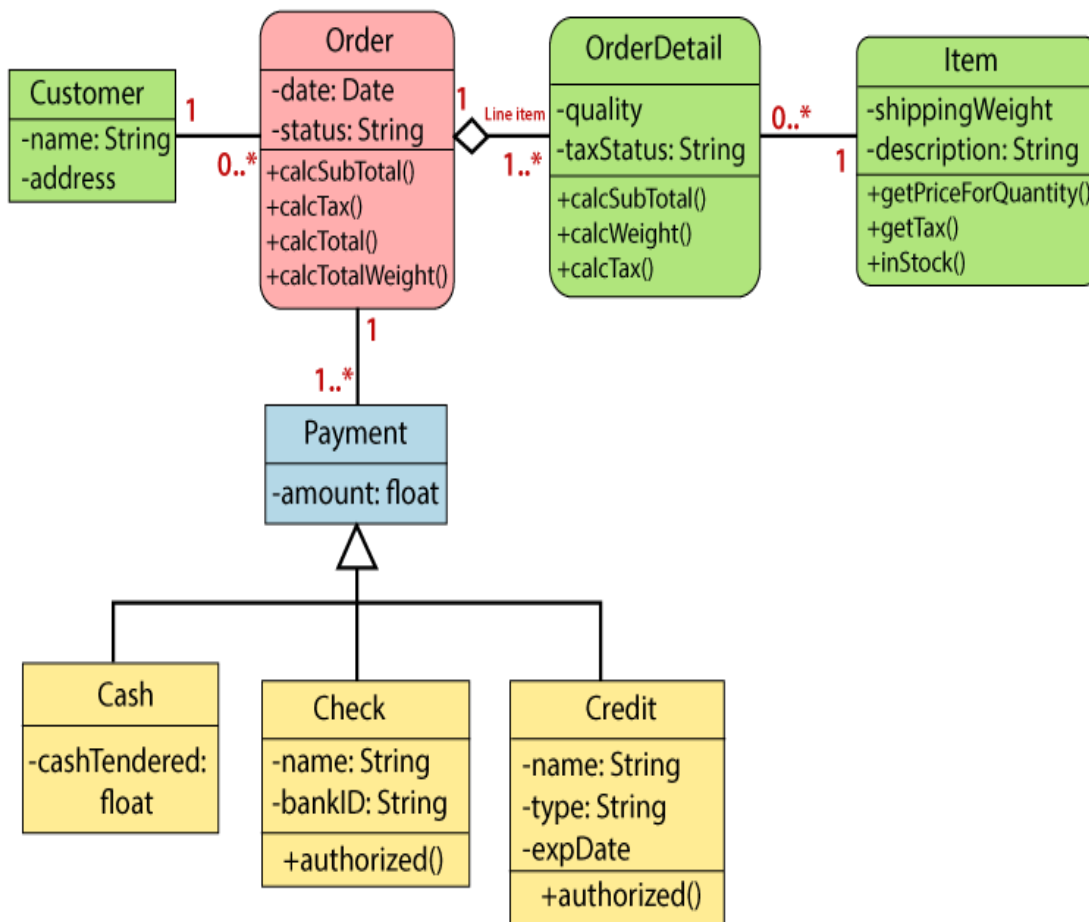
A minimum number of desired properties should be specified as more number of the unwanted property will lead to a complex diagram.

Notes can be used as and when required by the developer to describe the aspects of a diagram.

The diagrams should be redrawn and reworked as many times to make it correct before producing its final version.

Class Diagram Example

A class diagram describing the sales order system is given below.



Usage of Class diagrams

The class diagram is used to represent a static view of the system. It plays an essential role in the establishment of the component and deployment diagrams. It helps to construct an executable code to perform forward and backward engineering for any system, or we can say it is mainly used for construction. It represents the mapping with object-oriented languages that are C++, Java, etc. Class diagrams can be used for the following purposes:

1. To describe the static view of a system.
2. To show the collaboration among every instance in the static view.
3. To describe the functionalities performed by the system.
4. To construct the software application using object-oriented languages.

9. DECISION TREE

A UML Decision Tree is a decision-making tool represented as a tree-like structure in Unified Modeling Language (UML). It shows various decision points (nodes), possible actions (branches), and the outcomes (leaves) based on specific

conditions or rules. Each path from the root to a leaf represents a possible decision path. It's used to model complex decision processes in software systems, helping in visualizing choices and their potential results.

A UML (Unified Modeling Language) decision tree isn't a standard UML diagram type, but you can represent decision-making processes using several UML diagrams. Here are some ways you might represent decision-making processes in UML:

Activity Diagram

- **Decision Nodes:** Use decision nodes to show points in the process where decisions are made.
- **Merge Nodes:** Merge nodes are used to combine multiple branches back into a single path.
- **Example:** For a decision on whether to approve or reject a request, an activity diagram would show the initial action, the decision node with branches for approval and rejection, and the subsequent actions for each branch.

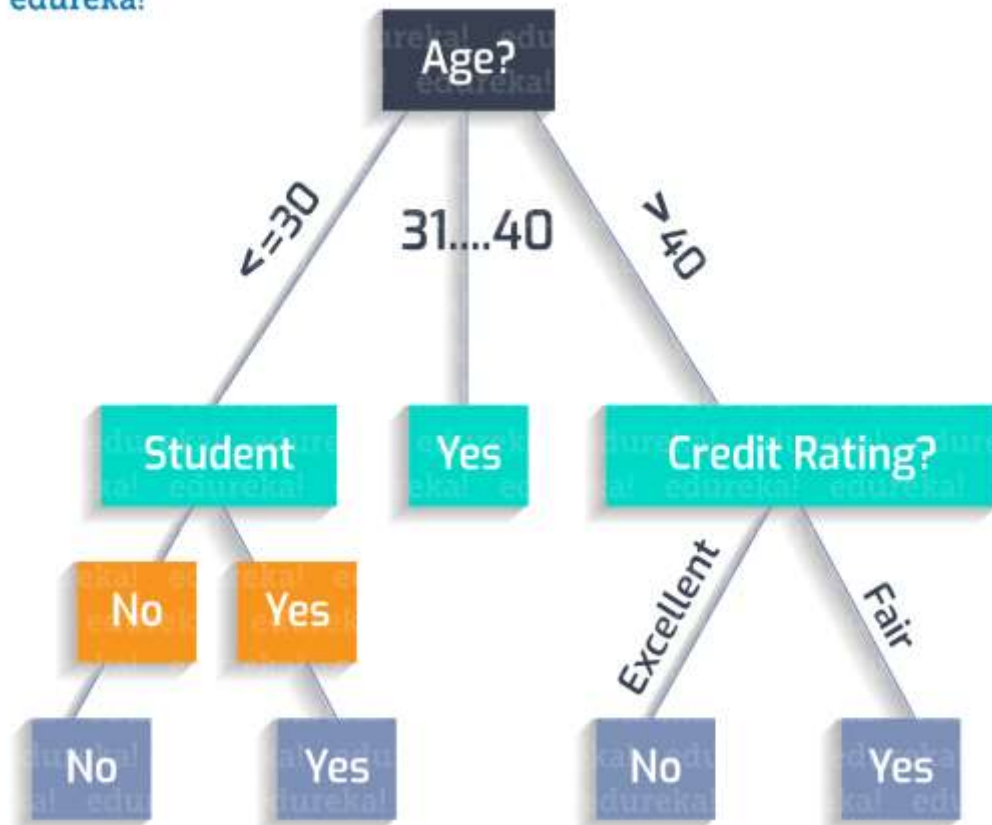
State Machine Diagram

- **States and Transitions:** Represent different states of an object and the transitions between states based on decisions.
- **Example:** A state machine diagram might show a process where an object moves through states based on certain conditions being met, such as "Submitted," "Under Review," and "Approved."

Flowchart (Optional)

- **Decision Points:** Use diamonds to represent decision points in a flowchart, which is a type of diagram commonly used to represent decision trees.

Example: A simple flowchart could show a decision point where a choice leads to different actions based on the outcome.



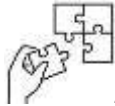
Decision Tree Figure



Points to Remember

- Designing a system application involves a structured approach to ensure that the final product meets the requirements and functions efficiently. Here are the key starting steps to follow when designing a system application:
 - ✚ Requirement Gathering and Analysis
 - ✚ Feasibility Study
 - ✚ System Design Planning
 - ✚ Prototyping
 - ✚ System Modelling
- Download and Install software design tools that will be used
- Check if software used online or offline are installed

- Create UML Diagrams: Use Unified Modeling Language (UML) diagrams such as class diagrams, sequence diagrams, and state diagrams to model the system's structure and behaviour.
- After designed your diagram then after export it
- Design Database Schema: Create a detailed database schema that defines tables, relationships, keys, and data types based on the ERD.
- Develop a Project Plan
- Review and Finalize the Design



Application of learning 2.1.

The Student Result Management System (SRIS) is a software application designed to manage and automate the student result processing for a school. The system allows for the entry, storage, processing, and reporting of student grades for various courses. Based on this project scenario, you can create the UML diagrams using standard symbols and notations.



Indicative content 2.2: Identification of Hardware and Software Technology



Duration: 8 hrs



Theoretical Activity 2.2.1: Identification of hardware and software technology



Tasks:

- 1: You are requested to answer the following questions:
 - i. Discuss on computer hardware as used in system design
 - ii. Describe system software as used in system design
 - iii. Explain application software needed to design backend
- 2: Write your answers on papers.
- 3: Present the findings/answers to the whole class or trainer.
- 4: For more clarification, read the key readings 2.2.1. In addition, ask questions where necessary.



Key readings 2.2.1. Identification of Hardware and software technology

To develop a system structure, it's important to understand how different types of hardware and software technologies contribute to the overall architecture.

Computer Hardware

1. **Central Processing Unit (CPU):** The brain of the computer, executing instructions from programs.
2. **Memory (RAM):** Temporary storage for data and instructions that the CPU needs while performing tasks.
3. **Storage Devices:** Hard drives or SSDs for long-term data storage.
4. **Networking Equipment:** Routers, switches, and network interface cards (NICs) to connect and communicate over networks.
5. **Input/Output Devices:** Keyboards, mice, monitors, and printers used for interaction with the system.

System Software

1. **Operating System (OS):** Manages hardware resources and provides services for application software. Examples include Windows, mac OS, and Linux.
2. **Device Drivers:** Specialized software that allows the OS to communicate with hardware components (e.g., graphics drivers, printer drivers).
3. **Utilities:** Tools for system maintenance and optimization (e.g., antivirus software, disk clean up tools).

Application Software

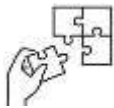
1. Microsoft Office: A suite of productivity applications including Word, Excel, PowerPoint, and Outlook. Useful for document creation, data analysis, presentations, and email.

2. Visual Paradigm: A UML tool for modeling and designing system architectures, including diagrams such as use case diagrams, class diagrams, and sequence diagrams.

3. E-Draw Max: A diagramming tool that allows you to create flowcharts, network diagrams, and other visual representations of systems and processes.

Browser

•**Web Browser:** Software such as Chrome, Firefox, or Edge that allows users to access and interact with web-based applications and resources. Essential for cloud-based tools and online collaboration.



Application of learning 2.2.

APC Ltd want to develop a system that records information of clients in their shop that sells products including delivery and payment, you are hired as backend developer responsible for identifying hardware and software to be used while developing that web application.



Indicative content 2.3: Application of SSADM (Structured System Analysis and Design Methods)



Duration: 10 hrs



Theoretical Activity 2.3.1: Description of SSADM



Tasks:

- 1: You are requested to answer the following questions:
 - i. Explain the objective of SSADM
 - ii. Describe SSADM Techniques
 - iii. List advantages and disadvantages of SSADM
- 2: Write your answers on papers.
- 3: Present the findings/answers to the whole class or trainer.
- 4: For more clarification, read the key readings 2.2.1. In addition, ask questions where necessary.



Key readings 2.3.1. Description of SSADM

SSADM is a systematic approach to software development, particularly useful in the analysis and design of information systems.

1. Objective of SSADM

Structured Approach: SSADM aims to provide a clear and structured methodology for developing systems to ensure all requirements are captured accurately.

Requirements Analysis: It emphasizes thorough analysis of user requirements to create systems that meet business needs.

Documentation: SSADM promotes detailed documentation throughout the development process, facilitating better communication among stakeholders.

Quality Assurance: By following a structured methodology, SSADM helps ensure the quality and reliability of the developed systems.

2. SSADM Techniques

Logical Data Modeling

To define and represent the data requirements of the system independently of any physical implementation.

Components:

Entities: Represent objects or concepts within the system.

Attributes: Describe the properties of entities.

Relationships: Define how entities interact with one another.

Outcome: A logical data model that serves as a blueprint for the data structure of the system.

Data Flow Modeling

Purpose: To visualize how data moves through the system, identifying inputs, processes, and outputs.

Components:

Processes: Activities that transform inputs into outputs.

Data Stores: Repositories where data is held.

External Entities: Sources or destinations of data outside the system.

Outcome: Data flow diagrams (DFDs) that depict the flow of information and help identify inefficiencies or redundancies in processes.

Entity Behavior Modeling

Purpose: To capture the dynamic behavior of entities within the system.

Components:

State Transitions: Define how an entity changes states in response to events.

Events: Triggers that cause changes in the state of an entity.

Outcome: State transition diagrams that illustrate how entities behave over time, helping to clarify requirements related to system interactions.

Conclusion

SSADM provides a comprehensive framework for analyzing and designing information systems. By utilizing techniques like logical data modeling, data flow modeling, and entity behavior modeling, developers can ensure that the resulting system is well-structured, meets user needs, and is of high quality.

3. Advantages

1. Separation of logical and physical aspects of the system
2. Well-defined techniques and documentation
3. User involvement

4. Disadvantages

1. The size of SSADM (in some circumstances)
2. Cost and time in training people
3. The learning curve can be considerable if the full method is used.



Practical Activity 2.3.2: Implementing stages for drawing SSADM



Task:

1: Read the task below:

Your school need software that will be used while recording marks of students, as backend application developer you are requested to implement stages for drawing SSADM.

2: Refers to key reading 2.3.2, implement stages for drawing SSADM.

3: present your work to whole class or trainer

4: Ask for clarification if any.



Key readings 2.3.2: Implementing stages for drawing SSADM

Implementing SSADM (Structured Systems Analysis and Design Method) involves several stages that guide the development process from initial analysis through to design.

Here's a structured approach to the stages of SSADM:

Stages for Drawing SSADM

1. Feasibility Study

Assess whether the proposed system is viable and aligns with business objectives.

Activities:



Conduct initial discussions with stakeholders.



Identify constraints, costs, and potential benefits.



Produce a feasibility report outlining findings.

2. Requirements Analysis

Objective: Gather and document detailed user requirements.

Activities:

Interviews and Surveys: Engage with users to understand their needs.

Workshops: Facilitate group discussions to gather comprehensive requirements.

Documentation: Create a requirements specification document.

Outcome: Clear and detailed understanding of what the system must achieve.

3. Logical System Specification

Objective: Develop a model of the system that reflects the identified requirements.

Activities:

Logical Data Modeling: Create entity-relationship diagrams.

Data Flow Modeling: Develop data flow diagrams to represent how data moves through the system.

Entity Behavior Modeling: Construct state transition diagrams to illustrate the behavior of entities.

Outcome: A set of models that define the system's functionality and data structure.

4. System Design

Objective: Translate the logical specifications into a physical design for implementation.

Activities:

Architecture Design: Define the overall system architecture.

Database Design: Develop a physical data model based on the logical data model.


Interface Design: Create design mockups for user interfaces.


Outcome: A detailed design specification that guides development.


5. Implementation


Objective: Build the system according to the design specifications.

Activities:

 **Coding:** Develop the software based on the design.

 **Testing:** Conduct unit tests and system tests to ensure functionality.

 **Documentation:** Produce user manuals and technical documentation.

 **Outcome:** A fully functional system ready for deployment.

6. Testing and Validation

Objective: Ensure the system meets all specifications and is ready for live use.

Activities:

User Acceptance Testing (UAT): Engage end-users to validate the system against their requirements.

Bug Fixing: Address any issues identified during testing.


Outcome: A validated system that meets user expectations.

7. Deployment

Objective: Roll out the system to users.

Activities:

 **Installation:** Set up the system in the production environment.

 **Training:** Provide training sessions for users.

 **Support:** Establish a support process for ongoing user assistance.

 **Outcome:** The system is live and operational.

8. Maintenance

Objective: Ensure the system continues to function effectively over time.

Activities:

 **Monitoring:** Regularly check system performance.



Updates and Upgrades: Implement changes based on user feedback and technological advancements.



Documentation: Keep system documentation up to date.

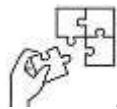


Outcome: A sustainable system that evolves with user needs.



Points to Remember

- By following these stages, organizations can effectively implement SSADM, ensuring that their systems are well-analyzed, designed, and maintained to meet the evolving needs of users and the business:
 - Feasibility Study
 - Requirements Analysis
 - Logical System Specification
 - System Design
 - Implementation
 - Testing and Validation
 - Deployment
 - Maintenance



Application of learning 2.3.

APC Ltd want to develop a system that records information of clients in their shop that sells products including delivery and payment, you are hired as backend developer responsible for implementing stages for drawing SSADM while developing that web application.



Indicative content 2.4: Application of Object-Oriented Analysis and Design



Duration: 8 hrs



Theoretical Activity 2.4.1: Description of Object-Oriented Analysis and Design



Tasks:

- 1: You are requested to answer the following questions:
 - i. Explain the term OOAD
 - ii. Describe the Advantages and disadvantages of OOAD
 - iii. Describe the phases in object-oriented software development
- 2: Write your answers on papers.
- 3: Present the findings/answers to the whole class or trainer.
- 4: For more clarification, read the key readings 2.4.1. In addition, ask questions where necessary.



Key readings 2.4.1.: Application of Object-Oriented Analysis and Design (OOAD)

Object-Oriented Analysis and Design (OOAD) is a software development approach that emphasizes the use of objects—instances of classes that encapsulate data and behavior. This methodology is widely used in software engineering to create modular, reusable, and maintainable code.

Introduction

OOAD focuses on identifying and organizing the system's components (objects) based on real-world entities and their interactions. It utilizes principles of object orientation, such as encapsulation, inheritance, and polymorphism, to create models that reflect both the static (structure) and dynamic (behaviour) aspects of systems.

Advantages and Disadvantages

2.1 Advantages

Modularity: Objects can be developed, tested, and maintained independently, promoting code reuse.

Encapsulation: Data and functions are bundled together, reducing complexity and increasing security.

Reusability: Existing objects can be reused in different systems, saving time and effort.

Flexibility: Changes can be made to individual objects without affecting the entire system.

Real-World Modeling: OOAD mirrors real-world entities, making it easier for stakeholders to understand the system.

2.2 Disadvantages

Complexity: The initial design can be more complex due to the abstraction and relationships between objects.

Learning Curve: Developers may require training to fully understand OO principles and patterns.

Performance Overhead: Object-oriented systems can be slower due to the overhead of object management and messaging.

Overhead of Design: The need for detailed design can lead to longer development times in some cases.

Phases in Object-Oriented Software Development

3.1 Analysis

Understand and document the requirements of the system.

Activities:

Identify Objects: Determine the key objects that represent real-world entities.

Define Relationships: Establish how these objects interact with each other.

Use Case Development: Create use cases that describe system functionalities from a user perspective.

Outcome: A comprehensive analysis model that outlines the system's requirements and object interactions.

3.2 Design

Create a blueprint for the system based on the analysis model.

Activities:

Class Design: Define classes, their attributes, and methods.

Relationship Design: Specify relationships among classes, including inheritance and associations.

Architecture Design: Develop an overall system architecture, including design patterns where applicable.

Outcome: A detailed design model that provides the specifications needed for implementation.

3.3 Implementation

Build the system according to the design specifications.

Activities:

Coding: Write code based on the defined classes and methods.

Testing: Conduct unit testing to ensure individual components work as intended, followed by integration testing.

Documentation: Create user documentation and technical documentation for

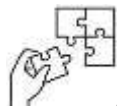
future reference.

Outcome: A functional software system that meets the specified requirements.



Points to Remember

- Object-Oriented Principles: Focus on encapsulation, inheritance, polymorphism, and abstraction.
- OOAD in SDLC: Iteratively analyze, design, and refine using object-oriented concepts throughout the software life cycle.
- System Modeling: Use UML diagrams to represent different perspectives external, interaction, structural, and behavioral.
- Key Phases: Move from requirements gathering to analysis, design, implementation, testing, deployment, and maintenance.



Application of learning 2.4.

As a trainee, you have been assigned to develop a Digital Library Management System for a school. The school's existing system for managing books, borrowing records, and member information is manual, making it challenging for students and staff to keep track of available books, issue/return dates, and overdue notices.

Your task is to design and develop a digital system that will allow:

- Students and staff to search, borrow, and return books digitally.
- Library staff to manage book inventories and user records efficiently.
- The system to send automatic notifications about due dates and new arrivals.

You are expected to apply your knowledge of Object-Oriented Analysis and Design (OOAD) to complete this project.



Learning outcome 2 end assessment

Theoretical assessment

1. Match the following terms with their definitions:

Answer	Term	Definition
.....	A) Data Flow Diagram	1) A diagram representing a sequence of operations
.....	B) Class Diagram	2) A diagram depicting object interactions over time
.....	C) Sequence Diagram	3) A diagram showing the static structure of a system
.....	D) Activity Diagram	4) A diagram representing data flow within a system

2. Match the following UML diagrams with their purpose:

Answer	UML Diagram	Purpose
.....	A) State Machine Diagram	1) Represents system interactions and message exchange
.....	B) Use Case Diagram	2) Shows dynamic behavior using state transitions
.....	C) Communication Diagram	3) Shows high-level system functionality with actors
.....	D) Component Diagram	4) Describes the physical components of a system

3. Evaluate the Following Statements as True or False:

- A sequence diagram shows the interactions between objects in terms of the messages exchanged over time.
- An activity diagram portrays the static structure of the system by showing its classes and methods.
- A use case diagram encapsulates the functional requirements of a system by utilizing actors and use cases.
- The composite structure diagram displays the relationships between the parts within a class, focusing on its internal structure.
- A package diagram is used to describe the dynamic behavior of a system and how objects interact over time.

- What are the main advantages and disadvantages of using UML diagrams in system design

Practical assessment

Imagine you are a software system designer for a student-owned project. Each student should manage their own project tasks and progress. Begin by creating a use case diagram to identify the primary actors (such as students, project mentors, and system administrators) and their interactions with the system. Next, develop a class diagram that outlines the system's structure, including the classes (like Student, Task, Project, and Mentor), attributes (such as task description, deadline, and status), methods (such as addTask and updateProgress), and relationships (like Student works on Task).

Using this foundation, write an algorithm to manage the project tasks and progress tracking, ensuring it efficiently handles various scenarios, including task assignment and status updates. Then, create a flowchart to represent this algorithm visually, considering the algorithm's efficiency in terms of time complexity.

Following this, draw a Level 0 Data Flow Diagram (DFD) to illustrate the main processes (such as task management and progress tracking), data flows (like task details and student updates), and external entities (such as students and mentors) involved in the system. Develop an Entity-Relationship Diagram (ERD) that captures the data requirements, detailing the entities (like Tasks, Projects, and Students), attributes, and relationships. Explain how the DFD and ERD complement each other regarding data flow and storage.

Finally, for a task prioritization process within this system (for students managing multiple tasks), create a decision table that includes conditions such as task urgency, student workload, and deadlines. Construct a decision tree from this table to visualize the decision-making process. Discuss how the decision table and decision tree help in making structured decisions for this scenario.

END



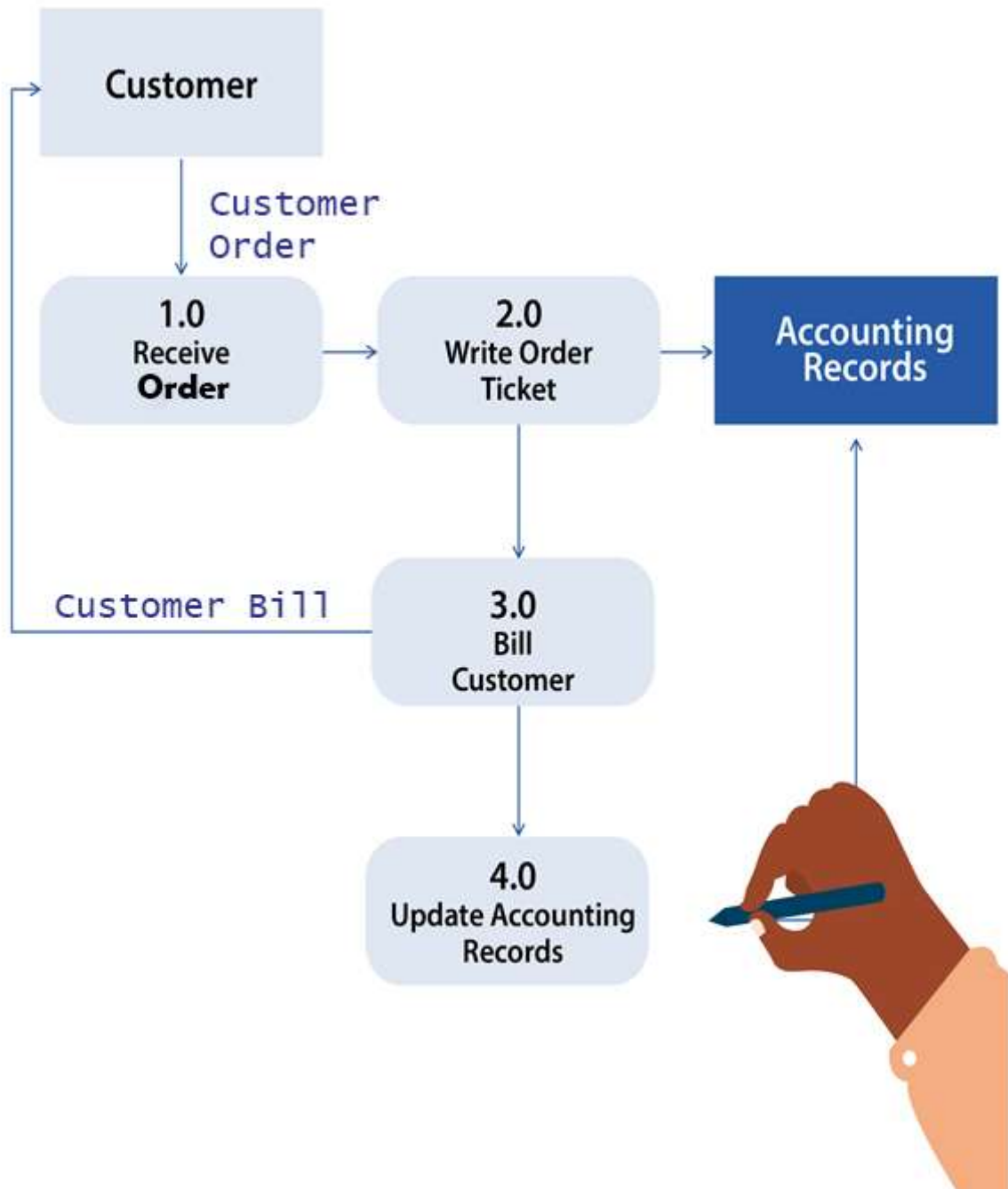
References

Books:

- Bird, S. K. (2009). *Natural language processing with Python*. O'Reilly Media.
- Elgendy. (2015). *Business analysis for beginners*. Outskirts Press.
- Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley.
- Hohpe, G. &. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley.
- Kendall, K. E. (2013). *Systems analysis and design*. Pearson.
- Kotonya, G. &. (1998). *Requirements engineering: Processes and techniques*. Wiley.
- McConnell, S. (2004). *A practical handbook of software construction*. Microsoft Press.
- McConnell, S. (2004). *Documenting system backend requirements*. Microsoft Press.
- Pressman, R. S. (2014). *Software engineering*. McGraw-Hill Education.
- Satzinger, J. W. (2012). *Systems analysis and design*. Course Technology, Cengage Learning.
- Sommerville. (2015). *Software engineering*. Pearson.
- Whitten, J. L. (2004). *Systems analysis and design methods*. McGraw-Hill.

Web links:

- Ambler, S. W. (n.d.). *Agile requirements change management*. Retrieved from <http://www.agilemodeling.com/essays/requirementsChangeManagement.htm>
- Backend developer*. (n.d.). Retrieved from Guru99: <https://www.guru99.com/backend-developer.html>
- How to gather requirements*. (n.d.). Retrieved from Lucidchart: <https://www.lucidchart.com/blog/how-to-gather-requirements>
- Mozilla Developer Network*. (n.d.). Retrieved from JSON: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>
- Stackify*. (n.d.). Retrieved from What is backend: <https://stackify.com/what-is-backend/>
- Techopedia*. (n.d.). Retrieved from FURPS: <https://www.techopedia.com/definition/25949/furps>
- Tutorials Point*. (n.d.). Retrieved from SDLC overview: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm



Indicative contents

3.1 Development of Data Flow

3.2 Application of Physical Data Model

3.3 Documentation of system design

Key Competencies for Learning Outcome 3: Build System Design

Knowledge	Skills	Attitudes
<ul style="list-style-type: none">● Identification of database Objects● Identification of Software tools● Description of Types of documentation of system design	<ul style="list-style-type: none">● Developing Data flow diagram of system● Designing Database● Developing System documentation● Developing User documentation	<ul style="list-style-type: none">● Having Team work● Being critical thinker● Being Innovative● Being attentive.● Being creative● Having Curiosity and Open Mindedness



Duration:32 hrs



Learning outcome 3 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Identify correctly software design tools based on the system design Requirements
2. Identify clearly database Objects based on the system design Requirements
3. Describe correctly types of documentation of system design based on the system design Requirements
4. Develop properly data Flow diagram based on the system design Requirements
5. Design correctly Database based on the system design Requirements
6. Develop correctly System documentation based on system analysis and architecture
7. Prepare correctly user documentation based on system analysis and architecture



Resources

Equipment	Tools	Materials
<ul style="list-style-type: none">• Computer	<ul style="list-style-type: none">• Microsoft office• Visual paradigm• E- Draw max• Browser	<ul style="list-style-type: none">• Internet



Indicative content 3.1: Development of Data Flow Diagram



Duration: 12 hrs



Theoretical Activity 3.1.1: Description of Data Flow Diagram



Tasks:

- 1: You are requested to answer the following questions:
 - i. Elements of DFD
 - ii. Rules of Drawing a DFD
 - iii. Describe types of DFD
 - iv. Explain software tools used to design DFD
- 2: Write your answers on papers.
- 3: Present the findings/answers to the whole class or trainer.
- 4: For more clarification, read the key readings 3.1.1. In addition, ask questions where necessary.



Key readings 3.1.1: Description of Data flow diagram

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both

1. Elements of DFD

2. **Processes:** Represented by circles or ovals, processes transform inputs into outputs.
3. **Data Flows:** Arrows that show the direction of data movement between processes, data stores, and external entities.
4. **Data Stores:** Represented by open-ended rectangles, these are repositories where data is stored for later use.
5. **External Entities:** Depicted as squares or rectangles, these are sources or destinations of data outside the system.

6. Rules of Drawing a DFD

7. Rules of Drawing a DFD

8. **Use Consistent Symbols:** Stick to standard DFD symbols for clarity and understanding.
9. **One Input, One Output:** Each process should have at least one input and one output.
10. **Label Everything:** Clearly label processes, data stores, and data flows for easy understanding.
11. **No Control Flows:** DFDs focus on data movement, not control flows (decisions or sequences).
12. **Maintain Hierarchy:** Higher-level DFDs should provide a broad overview, while lower-level diagrams (like Level 1 and Level 2) should detail processes.

13. Types of Data flow Diagram

3.1 Level 0 (Context)

Level 0 DFD: This is the highest-level DFD, which provides an overview of the entire system. It shows the major processes, data flows, and data stores in the system, without providing any details about the internal workings of these processes.

- **0-level DFD:** It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.

3.2 Level 1

Level 1 DFD: This level provides a more detailed view of the system by breaking down the major processes identified in the level 0 DFD into sub-processes. Each sub-process is depicted as a separate process on the level 1 DFD. The data flows and data stores associated with each sub process are also shown.

- **1-level DFD:** In 1-level DFD, the context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into sub processes.

3.3 Level 2 (Function decomposition)

Level 2 DFD: This level provides an even more detailed view of the system by breaking down the sub-processes identified in the level 1 DFD into further sub-processes. Each sub-process is depicted as a separate process on the level 2 DFD. The data flows and data stores associated with each sub-process are also shown.

- **2-level DFD:** 2-level DFD goes one step deeper into parts of 1-level DFD. It can be used to plan or record the specific/necessary detail about the system's functioning.

1. Software tools used to design DFD

4.1 Microsoft Office

Tools like Microsoft Visio or PowerPoint can be used to create simple DFDs with shapes and connectors.

4.2 Visual Paradigm

A dedicated modeling tool that offers robust features for creating DFDs along with other UML diagrams, complete with templates and collaboration options.

4.3 E-Draw

A diagramming tool that allows users to create various types of diagrams, including DFDs, with predefined shapes and easy drag-and-drop functionality.



Practical Activity 3.1.2: Develop Data Flow diagram



Task:

1: Read the given task:

Your school need to develop a library system that will be used while managing the process of renting books and returning them in library, as backend developer you are requested to develop the Data flow diagram level 1 that will be used while developing the backend of that system.

2: Refers to provided steps and rules in key reading 3.1.2, Design that data flow diagram.

3: Present your work to whole class or trainer

4: Ask for clarification if any.



Key readings 3.1.2: Develop Data Flow diagram

10 simple steps to draw a data flow diagram online with Lucidchart

Now that you know what makes up a data flow diagram, let's see how easy it is to make one using our powerful, online tool. We provide a ton of templates to use as a starting point. In this how-to, we're going to create a Level 0 DFD for an online shopping experience. Log in to your account (if you don't have one, sign up to try Lucid chart free for a week) and follow the steps below to make a DFD.

1. Select a data flow diagram template

In the Documents section, click on the orange +Document button and double-click on the Blank ERD & Data Flow diagram.

2. Name the data flow diagram

Click on the Blank ERD & Data Flow header in the top left corner of the screen. A pop-up screen opens, type the name of your diagram in the text box and click OK. The name of your DFD appears in the top left corner of the screen.

3. Add an external entity that starts the process

In the left column of the screen, you'll notice a lot of shapes and symbols. We've already created the four symbols you'll need to make a DFD. You can also add images to the diagram. Scroll through the list of symbols until you get to the bottom and see the heading Data Flow.

These are all the DFD symbols you need. (Note: Mouse over each shape to see what they represent: process, data stores, data flow, and external entities). We have symbols for Yourdon and Coad, Yourdon and DeMarco, and Gane and Sarson methods. Click and hold External Entity and drag it onto the workspace.

Click the highlighted text in the box and type the name of the external entity. For our example, we're typing "customer." You can use the curved arrow in the top left corner of the square to rotate the symbol. Delete a symbol by clicking it and pressing delete on your keyboard.

4. Add a Process to the DFD

Click and hold on a process symbol and drag it to where you want it on the workspace. Type the name of the process. We're calling this process "add product to cart."

5. Add a data store to the diagram

Click and hold on a data store symbol and drag it to where you want it on the workspace. Type the data store name. We're naming ours "shopping cart."

6. Continue to add items to the DFD

Drag-and-drop the appropriate symbols to add all the external entities, processes, and data stores to your diagram. Move symbols around by clicking and holding on them, and then drag them to a new location.

Click on a symbol to resize it, then click and hold the blue box in one of the corners and drag the corner to make the shape bigger or smaller. Use the background graph as a guide for alignment and sizing.

7. Add data flow to the DFD

Double-click on an entity, process, or data store, and then click and hold one of the orange circles and drag the line to the appropriate symbol.


Tip: If you prefer to create the data flow process as you complete the diagram, click on an entity, process, or data store and then click and hold one of the orange circles and drag the line to draw an arrow. Release the mouse button and a box with DFD symbols will appear. Click on the shape you want to add and it will automatically be created.

8. Name the data flow


Add a name to describe the data flow by double-clicking on the arrow line. An option to type text will appear, type the data flow name.


9. Customize the DFD with colors and fonts

Once you have the basic design of your diagram, you can add colors to symbols, change fonts, and adjust arrows. Here's how to:

 **Add colors to symbols:** Click on a symbol on the diagram and then click the color-fill icon and choose a color.

Tip: To make multiple symbols the same color, click the first item and then hold the shift key and click the remaining shapes. Next, click the color-fill icon and choose a color.

 **Change the font:** Choose Select All from the Edit option in the menu. Click the font box, choose a new font, and click it. All text in the diagram will be updated. You can use the other shortcuts (font color, size, bold, italic, underline, and alignment) to customize the font even more.

 **Adjust arrow style:** Click an arrow to select it. Next, click the arrow icon in the menu bar and choose one of the nine other styles.

NB: To change the style of all the arrows, choose Select All from the Edit menu.

10. Add a title and share your data flow diagram

At the top of the symbols column, you'll see a large letter T. Click it and drag it to where you want to add a title to the diagram. Type the title, and if you'd like to, adjust the font and type size using the shortcut keys at the top of the screen.

You can easily share your DFD with others either via email, link, social media (Facebook, Twitter, Google+, and LinkedIn), or embed it on a website. Click the blue Share button in the top right corner of the screen and a pop-up will appear. Choose how you'd like to share the DFD and enter the appropriate information.

When you add a collaborator by sending a link to the DFD via email, you can work on the data flow diagram simultaneously and use the chat feature (the yellow quote icon in the top right corner of the screen) to have discussions.

The easiest way to make a data flow diagram online

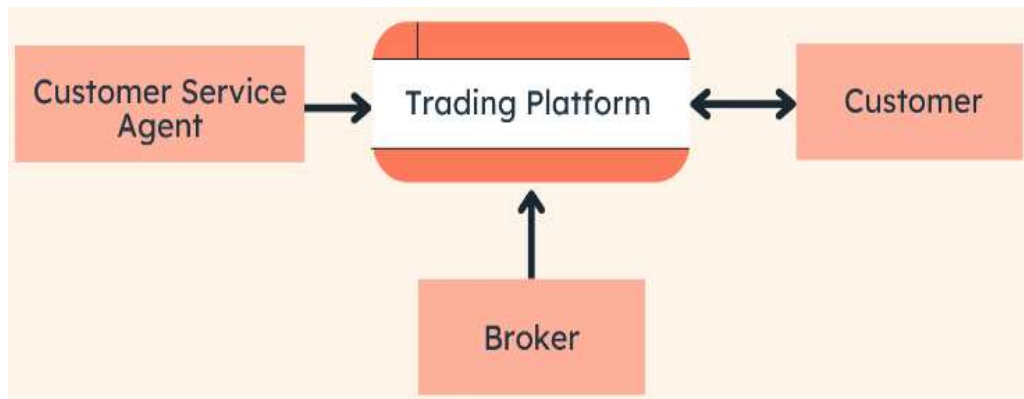
There you have it, the easiest way to make a data flow diagram online. Since Lucidchart is a web-based app, your diagram is automatically saved, and you can access it virtually anywhere you have an Internet connection. When you share your DFD with colleagues, you'll never have to wonder if they can open the file or access it. Best of all, if you give them permission to edit they can make changes to the DFD and add comments. Sign up for a free trial and see how easy it is to use Lucid chart.

Data Flow Diagram Examples

Professionals in various industries, like software engineering, IT, ecommerce, and product management & design, can use DFDs to better understand, refine, or implement a new system or process.

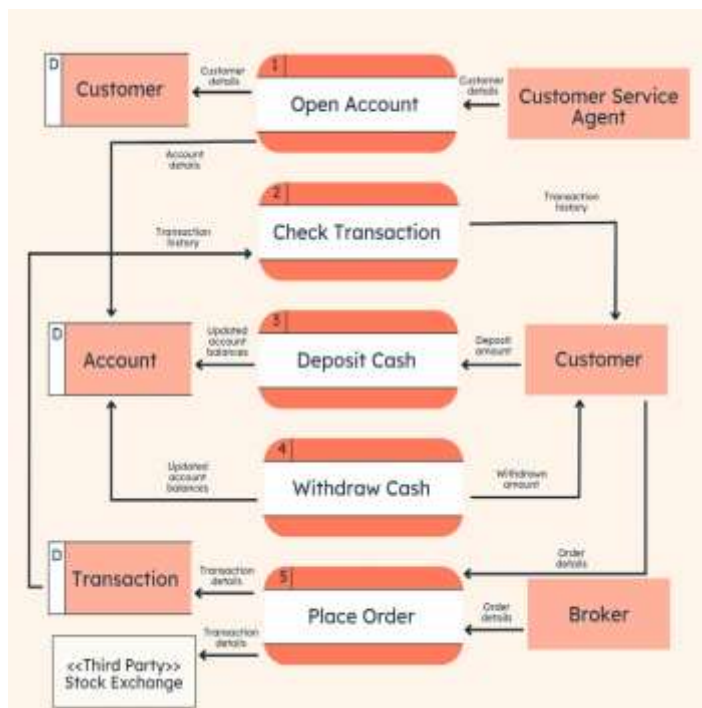
But what does a data flow diagram look like in practice — and how does it help your business? Here are three examples to help you contextualize DFDs' impact.

1. Level 0 DFD



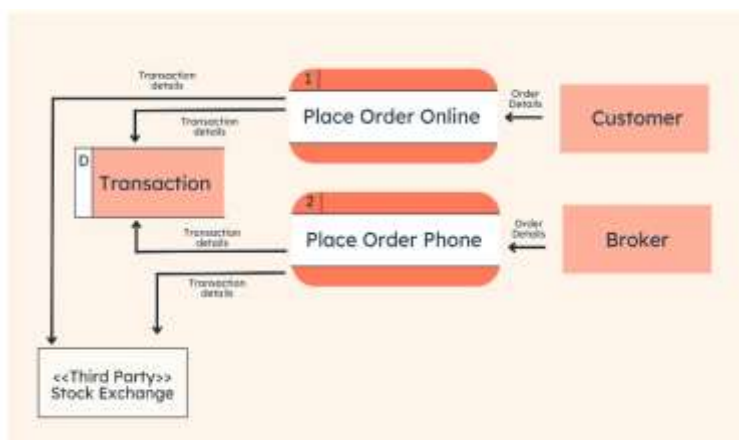
This Level 0 DFD provides a contextual map of a securities trading platform. Data flows in one direction from the customer service assistant and the broker to the platform. It also flows in two directions from customers to the platform and back again.

2. Level 1 DFD



This Level 1 DFD breaks down the customer process in more detail, expanding it to include account creation, cash withdrawals, and eventual securities transactions.





3. Level 2 DFD

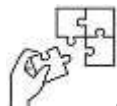


This Level 2 DFD decomposes the “Place Order” process to contextualize the steps required to place an order — either by a customer or by a broker. It even accounts for a third-party stock exchange center where transaction details are forwarded after an order is placed.



Points to Remember

- **A Data Flow Diagram (DFD)** is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both
- **Elements of DFD**
 -  Processes
 -  Data Flows
 -  Data Stores
 -  External Entities
- **DFD rules and tips**
 - Each process should have at least one input and an output.
 - Each data store should have at least one data flow in and one data flow out.
 - Data stored in a system must go through a process.
 - All processes in a DFD go to another process or a data store.



Application of learning 3.1.

Your school need to develop a library system that will be used while managing the process of renting books and returning them in library, as backend developer you are requested to develop the Data flow diagram level 1 that will be used while developing the backend of that system.



Indicative content 3.2: Application of Physical Data Model



Duration : 8 hrs











Theoretical Activity 3.2.1: Description of database Objects



Tasks:

1: You are requested to answer the following questions:

Describe the following database objects:

-  Tables
-  Constraints
-  Indexes
-  Triggers
-  Sequences
-  Views
-  Schemas
-  Synonyms

2: Write your answers on papers.

3: Present the findings/answers to the whole class or trainer.

4: For more clarification, read the key readings 3.2.1. In addition, ask questions where necessary.





Key readings 3.2.1.: Database objects

In a database management system (DBMS), various objects are used to store, manage, and manipulate data. Here's a breakdown of the primary database objects:

1. Tables

Definition: The fundamental objects that store data in rows and columns.

Components:

-  **Columns:** Define the attributes of the data (e.g., name, age).
-  **Rows:** Each row represents a single record or entry.

2. Views


Definition: Virtual tables that represent the result of a query. They do not store data themselves but provide a way to present data from one or more tables.


Use: Simplify complex queries and enhance security by restricting access to specific data.

3. Indexes

Definition: Objects that improve the speed of data retrieval operations on a table by providing quick access paths to the data.

Types:

 **Unique Indexes:** Ensure that the indexed columns do not contain duplicate values.

 **Composite Indexes:** Involve multiple columns to increase retrieval efficiency.

4. Stored Procedures

Definition: Precompiled collections of one or more SQL statements that can be executed as a single unit.

Use: Encapsulate business logic, perform complex calculations, and improve performance by reducing network traffic.

5. Functions

Definition: Similar to stored procedures, but they return a single value and can be used in SQL expressions.

Use: Perform calculations, manipulate data, and return results for use in queries.

6. Triggers

Definition: Special types of stored procedures that automatically execute in response to certain events on a particular table (e.g., insert, update, delete).

Use: Enforce business rules, maintain audit trails, and ensure data integrity.

7. Schemas

Definition: Logical containers that hold database objects such as tables, views, and procedures.

Use: Organize and group related objects, enhancing security and manageability.

8. Sequences

Definition: Objects that generate unique numeric values, often used for primary key columns.

Use: Automatically generate sequential numbers for new records.

9. Synonyms





Definition: Aliases for database objects that simplify access to them.

Use: Allow users to refer to objects using different names, often used for simplifying references to complex object names.

10. Constraints

Definition: Rules applied to table columns to enforce data integrity.

Types:

-  **Primary Key:** Uniquely identifies each record in a table.
-  **Foreign Key:** Ensures referential integrity between tables.
-  **Unique Constraint:** Ensures that all values in a column are distinct.
-  **Check Constraint:** Enforces domain integrity by limiting the values that can be entered in a column.



Practical Activity 3.2.1: Designing Database in E drawMax



Task:

1: Read the given task:

Your school need to develop a library system that will be used while managing the process of renting books and returning them in library, as backend developer you are requested to develop the database that will be used while developing the backend of that system.

2: Refers to provided steps and rules in key reading 3.1.2, Design that database.

3: Present your work to whole class or trainer

4: Ask for clarification if any.



Key readings 3.2.2: Designing Database in E Draw Max

To design a database in E-Draw Max, including tables and relationships, follow these steps. Here's a detailed guide tailored for creating an Entity-Relationship Diagram (ERD) for a library management system.

Step-by-Step Guide to Design a Database in E-Draw Max

Step 1: Open E-Draw Max

Launch E-Draw Max on your computer.

Step 2: Create a New Document

Click on "**File**" > "**New**" to create a new drawing.

Step 3: Select ER Diagram Template




- ✓ Look for the "**Templates**" section and search for "**ER Diagram**".
- ✓ Choose an appropriate template to start.

Step 4: Add Tables (Entities)

1. Authors Table

- ✓ Drag a **rectangle shape** to represent the table.
- ✓ Label it "**Authors**".

✓ Inside, add attributes:

-  AuthorID (PK)
-  Name
-  Birthdate





2. Books Table

Repeat the process to create a "**Books**" table:

-  BookID (PK)
-  Title
-  AuthorID (FK)
-  Genre
-  PublishedYear






3. Members Table

Create a "**Members**" table:

-  MemberID (PK)
-  Name
-  Email
-  PhoneNumber



4. Loans Table

Finally, create a "**Loans**" table:

-  LoanID (PK)
-  BookID (FK)
-  MemberID (FK)
-  LoanDate
-  ReturnDate

Step 5: Define Relationships

1. Authors to Books

-  Draw a line from the "**Authors**" table to the "**Books**" table.
-  Add a label indicating the relationship: "**1**" on the **Authors** side and "**M**" on the **Books** side.

2. Members to Loans

- ✚ Draw a line from the "**Members**" table to the "**Loans**" table.
- ✚ Label it "**1**" on the **Members** side and "**M**" on the **Loans** side.

3. Books to Loans

- ✚ Draw a line from the "**Books**" table to the "**Loans**" table.
- ✚ Label it "**1**" on the **Books** side and "**M**" on the **Loans** side.

Step 6: Arrange and Format

- ✚ Adjust the layout for clarity, ensuring that the tables and relationships are well spaced and easy to read.
- ✚ Use formatting options to style the tables and lines as needed.

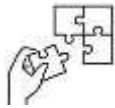
Step 7: Save Your Diagram

- ✚ Click on "**File**" > "**Save As**" to save your work in your desired format (e.g., E-Draw Max format, PDF, or image).



Points to Remember

- Physical database design consists of defining database objects and their relationships.
- The main objects of a database are tables, indexes, sequences, saved searches, and views.
- Databases are collections of tables, and those tables have fields (also known as columns). Every table contains a field known as an entity (or primary) key, which identifies the rows within that table. By telling your database that the key values in one table correspond to key values in another, you create a relationship between those tables; these relationships make it possible to run powerful queries across different tables in your database. When one table's entity key gets linked to a second table, it's known as a foreign key in that second table



Application of learning 3.2.

Your school need to develop a library system that will be used while managing the process of renting books and returning them in library, as backend developer you are requested to develop the database that will be used while developing the backend of that system.



Indicative content 3.3: Documentation of System Design



Duration: 12 hrs








Theoretical Activity 3.3.1: Description of Documentation of system design



Tasks:

1: You are requested to answer the following questions:

Describe the following types of documentation of system design:

-  System Design document (SDD)
-  Functional Specification Document (FSD)
-  Technical Specification Document (TSD)
-  Database design Document
-  Use case Document

2: Write your answers on papers.

3: Present the findings/answers to the whole class or trainer.

4: For more clarification, read the key readings 3.3.1. In addition, ask questions where necessary.



Key readings 3.3.1: Documentation of system design

Definition

The System Design Document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces.

Types of documentation of system design

1. System Design document (SDD)

The System Design Document (SDD) describes how the functional and nonfunctional requirements recorded in the Requirements Document, the preliminary user-oriented functional design recorded in the High-Level Technical Design Concept/Alternatives document, and the preliminary data design documented in the Logical Data

2. Functional Specification Document (FSD)

The Functional Specification Document (FSD) is written by the project's Business Analyst and provides detailed information on how the system solution will function based on what the requested behavior is.

3. Technical Specification Document(TSD)

This specification outlines the details of a product's design and the technical requirements for its development. It is a roadmap for engineers and developers to follow during the design and implementation phase.

The document specifies the technical attributes and requirements of the product, including the tools, technologies, and programming languages that will be used. It also describes the intended user experience, including the product's features and functionality.

4. Database design Document

The Database Design Document maps the logical data model to the target database management system with consideration to the system's performance requirements.

5. Use case Document

A use case document is a standardized document that describes a use case in detail. It includes the steps, preconditions, assumptions and expected outcomes or results. On the other hand, a use case is a general term that describes a specific scenario or interaction between a user or system and a product or service.

System documentation



Practical Activity 3.3.2: Developing system documentation for a backend system



Task:

1: Read the given task:

Your school developed a library system that will be used while managing the process of renting books and returning them in library, as backend developer you are requested to develop the system documentation that will be used as guide including the following files

- ✓ /_config.yml
- ✓ /index.md
- ✓ /architecture.md
- ✓ /api.md
- ✓ /database.md
- ✓ /deployment.md
- ✓ /troubleshooting.md.

2: Refers to provided steps in key reading 3.3.2, Develop the system documentation.

3: Present your work to whole class or trainer

4: Ask for clarification if any.

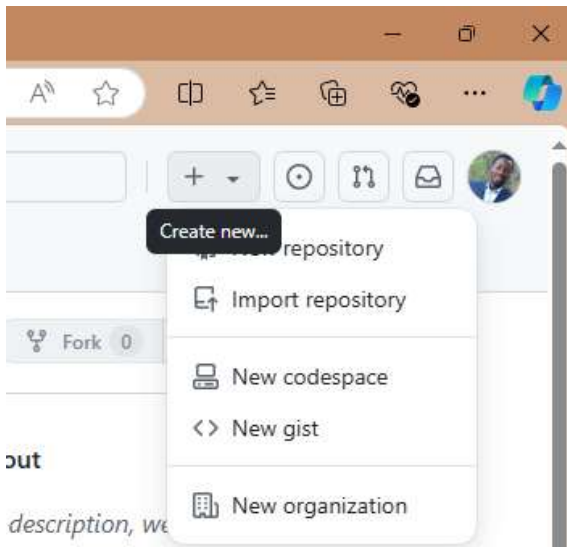


Key readings 3.3.2: Developing system documentation for a backend system

1. Developing system documentation for a backend system using GitHub Pages involves several steps.

1. Create a GitHub Repository

2. Go to GitHub and create a new repository.



3. description, we
4. Name it appropriately (e.g., backend-system-docs).

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * Repository name *

☒ Public Anyone on the internet can see this repository. You choose who can commit.

☐ Private You choose who can see and commit to this repository.

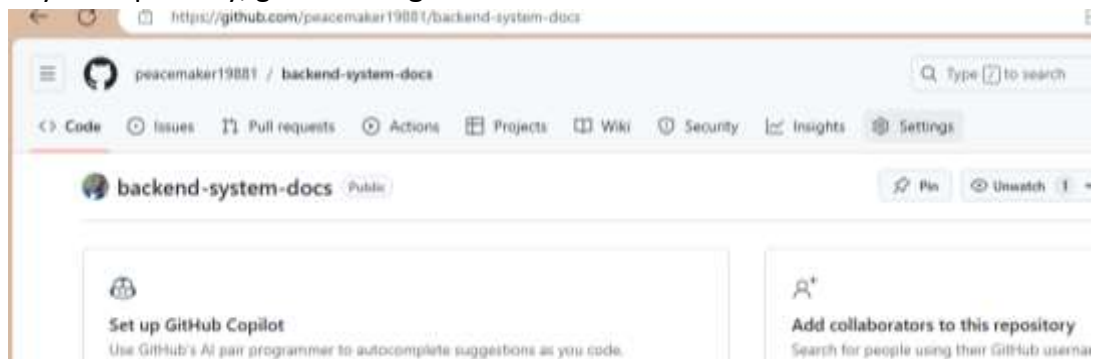
Initialize this repository with:

- ☐ Add a README file This is where you can write a long description for your project. [Learn more about READMEs.](#)

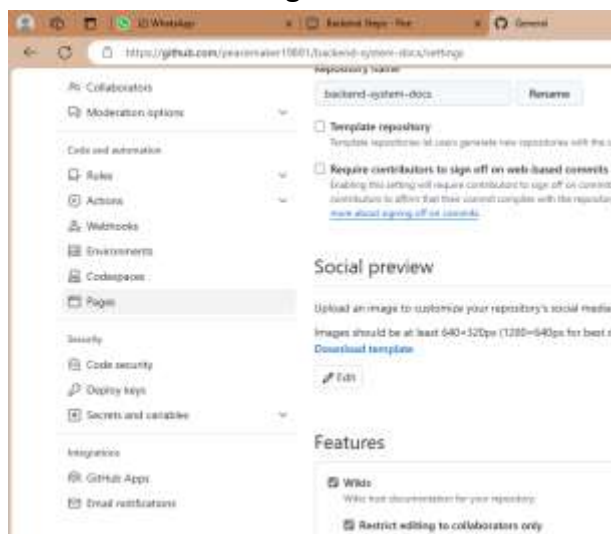
- 5.

2. Enable GitHub Pages

6. In your repository, go to **Settings**.



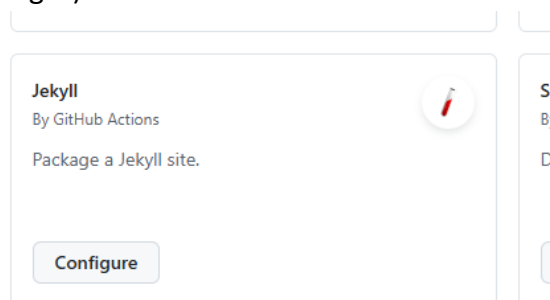
7. Scroll down to the **Pages** section.
8. Scroll down to the **Pages** section.



9. Select the branch (usually main or master) and a folder (typically /root or /docs) to publish your documentation.
10. Select the branch (usually main or master) and a folder (typically /root or /docs) to publish your documentation.

3. Choose a Documentation Framework

11. Consider using a static site generator like **Jekyll** (which is built into GitHub Pages).



12. Alternatively, you can use other frameworks like **MkDocs** or **Docusaurus**.
13. Alternatively, you can use other frameworks like **MkDocs** or **Docusaurus**.

14. Commit changes by clicking on **Commit changes**

4. Set Up Your Local Environment

15. If using Jekyll:

- Install Ruby and Jekyll on your local machine.

16. For Ruby

17. Download the RubyInstaller from RubyInstaller for Windows.

18. Run the installer and make sure to check the option to add Ruby to your PATH.

19. After installation, open a new Command Prompt window and verify the installation:

```
ruby -v
```

20.

21. For Jekyll

22. In the Command Prompt, run

```
gem install jekyll bundler
```

23.

- Create a new Jekyll site:

```
24. jekyll new my-docs
```

```
25. cd my-docs
```

5. Organize Your Documentation Structure

26. Create directories and markdown files for each section of your documentation:

```
27. /_config.yml
```

```
28. /index.md
```

```
29. /architecture.md
```

```
30. /api.md
```

```
31. /database.md
```

```
32. /deployment.md
```

```
33. /troubleshooting.md
```

6. Write Documentation

34. Use Markdown to document each section:

35. **Overview:** Describe the system and its purpose.

36. **Architecture:** Include diagrams and descriptions.

37. **API Documentation :** Detail endpoints, request/response formats.

38. **Database Schema:** Outline tables and relationships.

39. **Deployment:** Provide setup and configuration instructions.

40. **Troubleshooting:** Common issues and solutions.

7. Add Visuals

- 41. Include diagrams (e.g., flowcharts, UML) in your documentation.
- 42. Store images in an /assets directory and reference them in your Markdown.

8. Customize Your Site

- 43. Modify the _config.yml file to change site settings (title, description, theme).
- 44. Choose a theme that fits your documentation needs. GitHub Pages supports several themes.

9. Preview Locally

- 45. Run the Jekyll server locally to preview your documentation:

```
46. bundle exec jekyll serve
```

- 47. Visit <http://localhost:4000> to see your site.

10. Push Changes to GitHub

- 48. Commit your changes and push them to your GitHub repository:

```
49. git add .
```

```
50. git commit -m "Initial documentation"
```

```
51. git push origin main
```

11. Access Your Documentation

- 52. Go to the GitHub Pages URL provided in the settings (usually <https://<username>.github.io/<repository-name>>).
- 53. Verify that everything displays correctly.

12. Maintain and Update

- 54. Regularly update your documentation with new information.
- 55. Use version control to track changes and collaborate with team members.

13. Promote Documentation Usage

- 56. Share the link to your documentation with team members and stakeholders.
- 57. Encourage feedback and contributions.



Practical Activity 3.3.3: Developing User documentation



Task:

1: Read the given task:

Your school have developed a library system that used while managing the process of renting books and returning them in library, as backend developer you are requested to develop the user documentation for backend in GitHub pages containing the following files:

- ✓ /_config.yml
- ✓ /index.md
- ✓ /getting-started.md
- ✓ /features.md
- ✓ /user-interface.md
- ✓ /troubleshooting.md
- ✓ /faqs.md

2: Refers to provided steps in key reading 3.3.3, Develop the user documentation.

3: Present your work to whole class or trainer



4: Ask for clarification if any.



Key readings 3.3.3: Developing User documentation

Developing user documentation using GitHub Pages involves several steps. Here's a structured approach to help you create and publish your documentation effectively:

1. Create a GitHub Repository

-  Go to GitHub and create a new repository (e.g., user-documentation).
-  Choose whether to make it public or private based on your needs.

2. Enable GitHub Pages

-  Navigate to the **Settings** of your repository.
-  Scroll down to the **Pages** section.

- ✚ Select the branch (typically main or master) and choose the folder (usually /root or /docs) to publish your documentation.

3. Choose a Documentation Framework

- ✚ You can use Jekyll (which is integrated with GitHub Pages) or other static site generators like MkDocs or Docusaurus.
- ✚ For this guide, we'll focus on Jekyll.

4. Set Up Your Local Environment

- ✚ **Install Ruby and Jekyll** (follow the previous instructions if you haven't done this yet).
- ✚ Create a new Jekyll site:

```
jekyll new my-docs  
cd my-docs
```

5. Organize Your Documentation Structure

- ✚ Create directories and Markdown files for each section of your user documentation:

```
/_config.yml  
/index.md  
/getting-started.md  
/features.md  
/user-interface.md  
/troubleshooting.md  
/faqs.md
```

6. Write Documentation Content

- Use Markdown to document each section:
 - ✚ **Introduction:** Overview of the product and its purpose.
 - ✚ **Getting Started:** Installation and setup instructions.
 - ✚ **Features:** Detailed explanations of key features.
 - ✚ **User Interface:** Descriptions of the UI elements.
 - ✚ **Troubleshooting:** Common issues and solutions.
 - ✚ **FAQs:** Answers to frequently asked questions.

7. Add Visuals

- ✚ Include screenshots and diagrams to enhance understanding.
- ✚ Store images in an `/assets` directory and reference them in your Markdown files:

```
![[Screenshot]](assets/screenshot.png)
```

8. Customize Your Site

- ✚ Modify the `_config.yml` file to set site settings (title, description, theme).
- ✚ Choose a theme suitable for documentation. GitHub Pages supports several built-in themes.

9. Preview Locally

- ✚ Run the Jekyll server locally to preview your documentation:

```
bundle exec jekyll serve
```

- ✚ Access your documentation at `http://localhost:4000`.

10. Push Changes to GitHub

- ✚ Commit your changes and push them to your GitHub repository:

```
git add .  
git commit -m "Initial user documentation"  
git push origin main
```

11. Access Your Documentation Online

- ✚ Go to the GitHub Pages URL provided in the settings (usually <https://<username>.github.io/<repository-name>>).
- ✚ Verify that your documentation is displayed correctly.

12. Maintain and Update Regularly

- ✚ Regularly update your documentation as new features are added or existing content changes.
- ✚ Use version control to track changes and collaborate with your team.

13. Promote Documentation Usage

- ✚ Share the link to your documentation with users.
- ✚ Encourage feedback and questions to improve the documentation continuously.

14. Gather Feedback

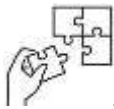
- ✚ Implement a feedback mechanism (e.g., issues in GitHub, forms) to collect user input on the documentation.



Points to Remember

- **System Design Document (SDD):** Provides a detailed description of the system architecture and design, including components, interfaces, and data flow.
- **Functional Specification Document (FSD):** Defines the system's functionality, detailing how the system should behave and what it should do.
- **Technical Specification Document (TSD):** Outlines the technical details of the system, including hardware, software, and network requirements.
- **Database Design Document:** Documents the structure of the database, including tables, relationships, and constraints.
- **Use Case Document:** Describes specific scenarios of system interaction, detailing how users will interact with the system to achieve a goal.
- **User Documentation:** Offers instructions and guides for end-users on how to use the system's features and functionalities.
- While developing system documentation in GitHub pages you follow those steps:
 - ✚ Create a GitHub Repository
 - ✚ Enable GitHub Pages
 - ✚ Choose a Documentation Framework
 - ✚ Set Up Your Local Environment
 - ✚ Organize Your Documentation Structure
 - ✚ Write Documentation
 - ✚ Add Visuals
 - ✚ Customize Your Site
 - ✚ Preview Locally
 - ✚ Push Changes to GitHub
 - ✚ Access Your Documentation
 - ✚ Maintain and Update

- ✚ Promote Documentation Usage
- While developing user documentation in GitHub pages you follow those steps:
 - ✚ Create a GitHub Repository
 - ✚ Enable GitHub Pages
 - ✚ Choose a Documentation Framework
 - ✚ Set Up Your Local Environment
 - ✚ Organize Your Documentation Structure
 - ✚ Write Documentation Content
 - ✚ Add Visuals
 - ✚ Customize Your Site
 - ✚ Preview Locally
 - ✚ Push Changes to GitHub
 - ✚ Access Your Documentation Online
 - ✚ Maintain and Update Regularly
 - ✚ Promote Documentation Usage
 - ✚ Gather Feedback



Application of learning 3.3.

Your school developed a library system that will be used while managing the process of renting books and returning them in library, as backend developer you are requested to develop the system and user documentation that will be used as guide.



Learning outcome 3 end assessment

Written assessment

I. Multiple Choice Questions

1. What is the purpose of a Data Flow Diagram (DFD)?
 - A) To display the physical layout of hardware components
 - B) To represent the flow of data within a system
 - C) To document user manuals
 - D) To visualize software code structure
2. Which of the following is NOT an element of a Data Flow Diagram (DFD)?
 - A) Data Store
 - B) Process
 - C) Entity
 - D) Network
3. What is the main difference between Level 0 and Level 1 DFD?
 - A) Level 0 DFD is more detailed than Level 1 DFD
 - B) Level 1 DFD provides a high-level overview, while Level 0 DFD is detailed
 - C) Level 0 DFD is the context diagram, while Level 1 DFD breaks down processes into more detail
 - D) There is no difference between Level 0 and Level 1 DFD
4. Which software tool is commonly used for creating Data Flow Diagrams?
 - A) Adobe Photoshop
 - B) Microsoft Office
 - C) Visual Paradigm
 - D) Oracle Database
5. In the context of database design, what is a 'relationship'?
 - A) A link between two tables that defines how data in one table relates to data in another
 - B) A connection between two software applications
 - C) A feature in Microsoft Word
 - D) A data storage unit

6. Which document details the functionality of a system?

- A) Technical Specification Document (TSD)
- B) Functional Specification Document (FSD)
- C) System Design Document (SDD)
- D) Database Design Document

II. Answer by True or False for the following Statements

1. A Data Flow Diagram (DFD) is used to represent the flow of data within a system.
2. Level 2 DFD is typically the highest level, providing a very broad overview of the system.
3. Microsoft Office is a preferred tool for creating Data Flow Diagrams.
4. In database design, identifying database objects includes defining tables and their relationships.
5. System Design Document (SDD) primarily focuses on the technical aspects of system implementation.
6. Use Case Documents are a type of system design documentation.
7. The Physical Data Model represents the logical structure of the database.
8. Technical Specification Document (TSD) includes detailed information about the system's hardware requirements.

Practical assessment

To address the development and design of the XYZ school management system located in Rubavu, Rwanda, we will go through several key aspects, including the development of a Data Flow Diagram (DFD), application of a Physical Data Model, identification and design of database objects, and documentation of the system design. You are hired to develop dataflow diagram, physical data model and design database object and make documentation of the system design.



References

Books:

- Bird, S. K. (2009). *Natural language processing with Python*. O'Reilly Media.
- Elgendy. (2015). *Business analysis for beginners*. Outskirts Press.
- Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley.
- Hohpe, G. &. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley.
- Kendall, K. E. (2013). *Systems analysis and design*. Pearson.
- Kotonya, G. &. (1998). *Requirements engineering: Processes and techniques*. Wiley.
- McConnell, S. (2004). *A practical handbook of software construction*. Microsoft Press.
- McConnell, S. (2004). *Documenting system backend requirements*. Microsoft Press.
- Pressman, R. S. (2014). *Software engineering*. McGraw-Hill Education.
- Satzinger, J. W. (2012). *Systems analysis and design*. Course Technology, Cengage Learning.
- Sommerville. (2015). *Software engineering*. Pearson.
- Whitten, J. L. (2004). *Systems analysis and design methods*. McGraw-Hill.

Web links:

- Ambler, S. W. (n.d.). *Agile requirements change management*. Retrieved from <http://www.agilemodeling.com/essays/requirementsChangeManagement.htm>
- Backend developer. (n.d.). Retrieved from Guru99: <https://www.guru99.com/backend-developer.html>
- How to gather requirements. (n.d.). Retrieved from Lucidchart: <https://www.lucidchart.com/blog/how-to-gather-requirements>
- Mozilla Developer Network. (n.d.). Retrieved from JSON: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>
- Stackify. (n.d.). Retrieved from What is backend: <https://stackify.com/what-is-backend/>
- Techopedia. (n.d.). Retrieved from FURPS: <https://www.techopedia.com/definition/25949/furps>
- Tutorials Point. (n.d.). Retrieved from SDLC overview: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm



October 2024