

iLearn:The Digital Learning Environment

Submitted by:

J Deekshika (20VV1A1222)

J Saikiran (20VV1A1223)

K Subramanyam (20VV1A1224)

K Aditya (20VV1A1225)

Under the supervision of

Dr.B Tirumala Rao sir

DEPT. OF Information Technology

JNTU-GV

Dwarapudi, Andhra Pradesh 535003

JUNE 2022

Software Requirements Specification

July 8, 2022

1 Introduction

1.1 Document Purpose

A digital learning environment is a framework in which a set of general-purpose and specially designed tools for learning may be embedded, plus a set of applications that are geared to the needs of the learners using the system. The framework provides general services such as an authentication service, synchronous and asynchronous communication services, and a storage service. The tools included in each version of the environment are chosen by teachers and learners to suit their specific needs. These can be general applications such as spreadsheets, learning management applications such as a Virtual Learning Environment (VLE) to manage homework submission and assessment, games, and simulations.

1.2 Product Scope

A digital learning environment is a framework in which a set of general-purpose and specially designed tools for learning may be embedded, plus a set of applications that are geared to the needs of the learners using the system. The framework provides general services such as an authentication service, synchronous and asynchronous communication services, and a storage service. The tools included in each version of the environment are chosen by teachers and learners to suit their specific needs. These can be general applications such as spreadsheets, learning management applications such as a Virtual Learning Environment (VLE) to manage homework submission and assessment, games, and simulations.

1.3 Intended Audience and Document Overview

The document is intended to read by project managers, developers, tester, users and document writers. The document is organized into 5 parts:

Introduction Overall Description Specific Requirements Other Non-Functional Requirements Other Requirements

1.3.1 Definitions, Acronyms and Abbreviations:

Author:

Person submitting an article to be reviewed. In case of multiple authors, this term refers to the principal author, with whom all communication is made.

Database:

Collection of all the information monitored by this system.

Editor:

Person who receives articles, sends articles for review, and makes final judgments for publications.

Field:

A cell within a form.

Historical Society Database:

The existing membership database (also HS database).

Member:

A member of the Historical Society listed in the HS database.

Reader:

Anyone visiting the site to read articles.

CSV:

Comma Separated Values

DNF:

Disjunctive Normal Form

ID :

Identifier

GUI:

Graphical User Interface

HTM:

1.4 Document Convention

Main Heading Titles:

oFont: Arial

oFace: Bold

oSize: 14

Sub Heading Titles:

oFont: Arial

oFace: Bold

oSize: 12

Other Text Explanations:

oFont: Times New Roman

oFace: Normal

oSize: 12

1.5 References and Acknowledgments

This casestudy is reference from the book written by sommerville .The link of the website is :“ <https://iansommerville.com/software-engineering-book/case-studies/> ”

The documentation of the case study is done using OverLeaf .The link of the website is : “<https://www.overleaf.com/> ”

The source for the UML diagrams is plantUML. The link of the website is : “ <https://www.plantuml.com/> ”

The website used to design the Usecase diagrams is plantText. It is a tool which designs daigrams of different models .The link of the website is : “ <https://www.planttext.com/> ”

2 Overall Description

2.1 Product Overview

iLearning is a digital learning environment which supports the learning of the students . One of the most important requirements for the iLearn system was that it should be an open system that could easily accommodate new features and existing services. We aimed to achieve this by designing the system so that everything was a service and that, with appropriate permissions, users could replace pre-specified services with their own service version.

2.2 Product Functionality

Utility services that provide basic application-independent functionality and that may be used by other services in the system. Utility services are usually developed or adapted specifically for this system.

Application services that provide specific applications such as email, conferencing, photo sharing, etc., and access to specific educational content such as scientific films or historical resources. Application services are external services that are either specifically purchased for the system or are available freely over the Internet.

Configuration services that are used to adapt the environment with a specific set of application services and to define how services are shared between students, teachers, and their parents

2.3 Design and Implementation Constraints

Software design and implementations activities are invariably interleaved .Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements. Implementation is the process of realizing the design as a program. Sometimes there is a separate design stage, and this design is modeled and documented.

Design and Implementations are closely linked and you should normally take implemen-

tations issue into the account when developing a design. One of the most important implementations decisions that has to be made at an early stage of software project is whether to build or to buy the application soft-ware.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 Interfaces:

The logical characteristics of each interface between the software product and its users. This includes the required screen formats, page or window layouts, the content of any reports or menus, or availability of programmable function keys necessary to accomplish the software requirements.

Interface optimization methods and the people responsible for them. A simple list of dos and don'ts will do the trick.

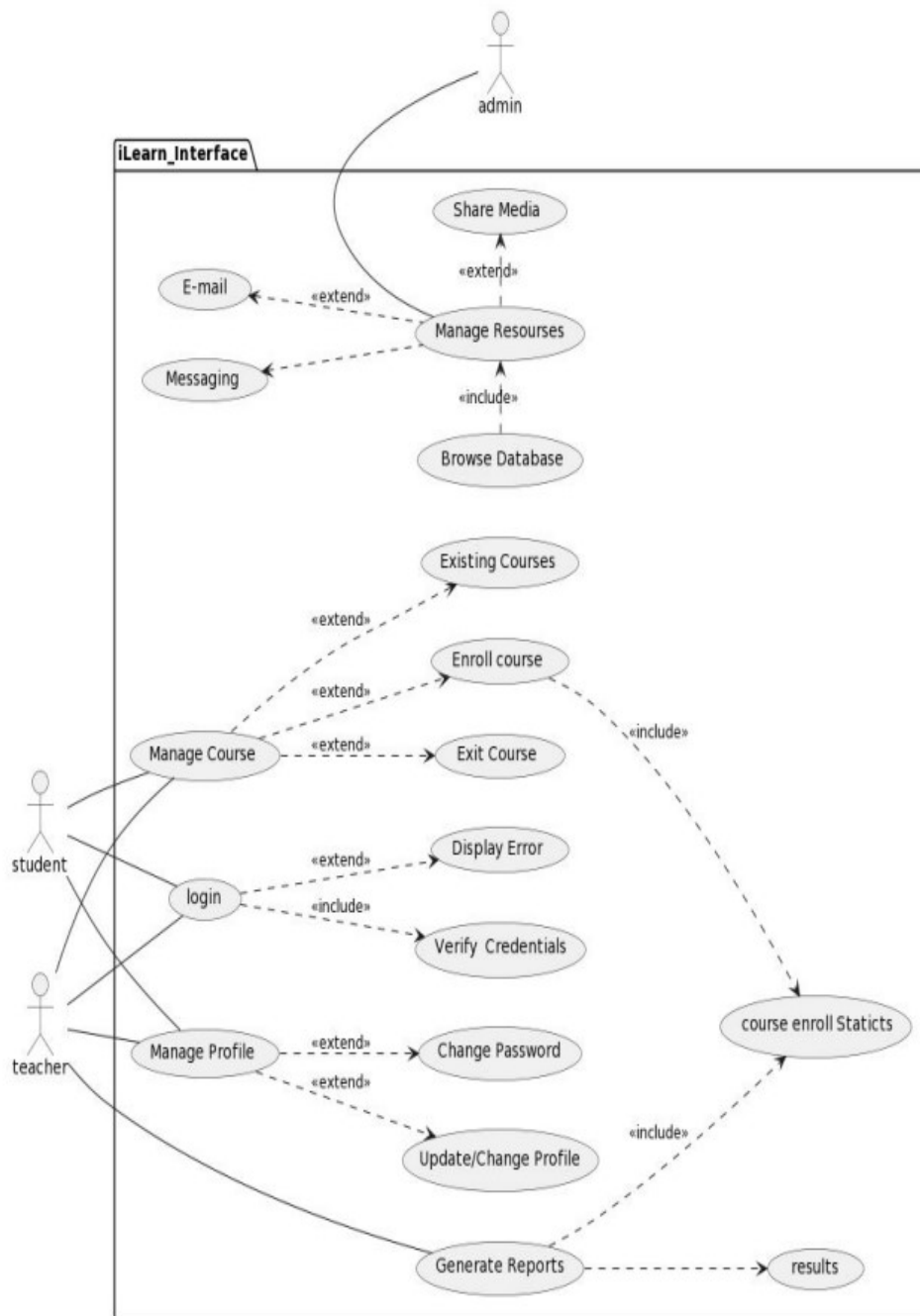
3.1.2 Software Interfaces:

The application allows import a structured MS Word document via HTML data format.

The application allows populating a MS Word document with project data via HTML data format.

The application allows import / export a list of requirements from / to MS Excel sheet via CSV data format.

3.2 Use Case Model



3.2.1 Use Case 1 (iLearn Interface Usecase diagram)

Author – Saikiran 23

Purpose - This use case is for the Interface of the learning system. Using the use case model of iLearn , the foreground work of the environment is shown . This usecase Model shows the main functions of the environment we are designing.

Requirements Traceability – To design this usecase model we require a usecase model tool. This usecase diagram is designed from the Planttext website .

Priority - The priority is High. Because this usecase shows the different functions of the casestudy. In this usecase model there are many types of usecases which shows the function of each in the casestudy. Using this usecase model the casestudy is being developed.

Preconditions - The precondition for this usecase model is

1. To have a system with internet connection to build usecase diagram .
2. We have to know the process of code to design uml diagrams.

Post conditions - The conditions that will be satisfied after the use case successfully completes

Actors – Student, Teacher, Admin.

Extends – This is not an extension usecase

Flow of Events

1. **Basic Flow** - flow of events normally executed in the use-case
2. **Alternative Flow** - a secondary flow of events due to infrequent conditions
3. **Exceptions** - Exceptions that may happen during the execution of the use case. . .

4 Other Non Functional Requirements

4.1 Performance Requirements

Startup Time

The application should display the opened document within 5s after it is started.

Edit Response Time

The application should display updated values within 1s after user triggers the edit operation.

Document Size

The application shall allow users to open documents up to 10000 objects and 100 file attachments with total size up to 100MB.

4.2 Safety and Security Requirements

Reliable Internet is the backbone of the software so for the live broadcasting of the video needs sufficient and uninterrupted internet connection.

Power is a significant feature and the power supply should be always taken care of. An uninterrupted power supply is always recommended.

The security system features from having a login for all the users to access the software. The login details will be used in the system also. So, the chances of the software getting intruded are very less.

4.3 Software Quality Attributes

the Java Virtual Machine helps the Pathshala to achieve platform independence. Hence, it can run on any environment that is available in the client computer.

5 Other Requirements

Legal Requirements:

Illegal duplication of the reports will be strictly dealt with. This is not an open source software hence source code of the product won't be open. Further modifications and improvements rights will be with the developer team.

Appendix A: Glossary

- 1) SRS: Software Requirement Specification
- 2) Client/User: Students
- 3) Server: Teacher
- 4) RAM: Random Access Memory
- 5) MB: Megabyte (Unit of Memory Storage)
- 6) SQL: Structured Query Language
- 7) HTTP: Hyper Text Transfer Protocol
- 8) User ID: Unique username issued to each user on login
- 9) Password: Unique word given to each user as a secret code

Class Diagram

Code: @startuml

left to right direction

class Student

String Name

int Age

String Education

String Username

String Password

String Email Address

Other Details

+addStudent()

searchStudent()

manageStudent()

class Course

+String CourseName

+String CourseDescription

+String CourseType

+String InstructorName

+int Fee

addCourse()

editCourse()

+enrollCourse()

deleteCourse()

+searchCourse()

+makePayment()

class Instructor +String InstructorName

+String InstructorRating

+String InstructorData

+String CoursesTeaching

-addInstructor()
-editInstructorDetails()
-removeInstructor()
+searchInstructor()

class User
+int *userId*
+ *String**userName*
+ *String**userRole*
+ *String**userEmail*
+ *String**userDob*
+ *String**userAddress*
+ *addUser()*
+ *editUser()*
deleteUser()
+ *searchUser()*

classAdmin
– *int**UserId*
– *String**AdminName*
– *String**AdminRole*
– *manageUser()*
– *managePayments()*
– *manageCourses()*
– *manageDatabase()*

classPermission
*int**PermissionId*
*String**Description*
givePermission()
denyPermission()
managePermissionRequests()

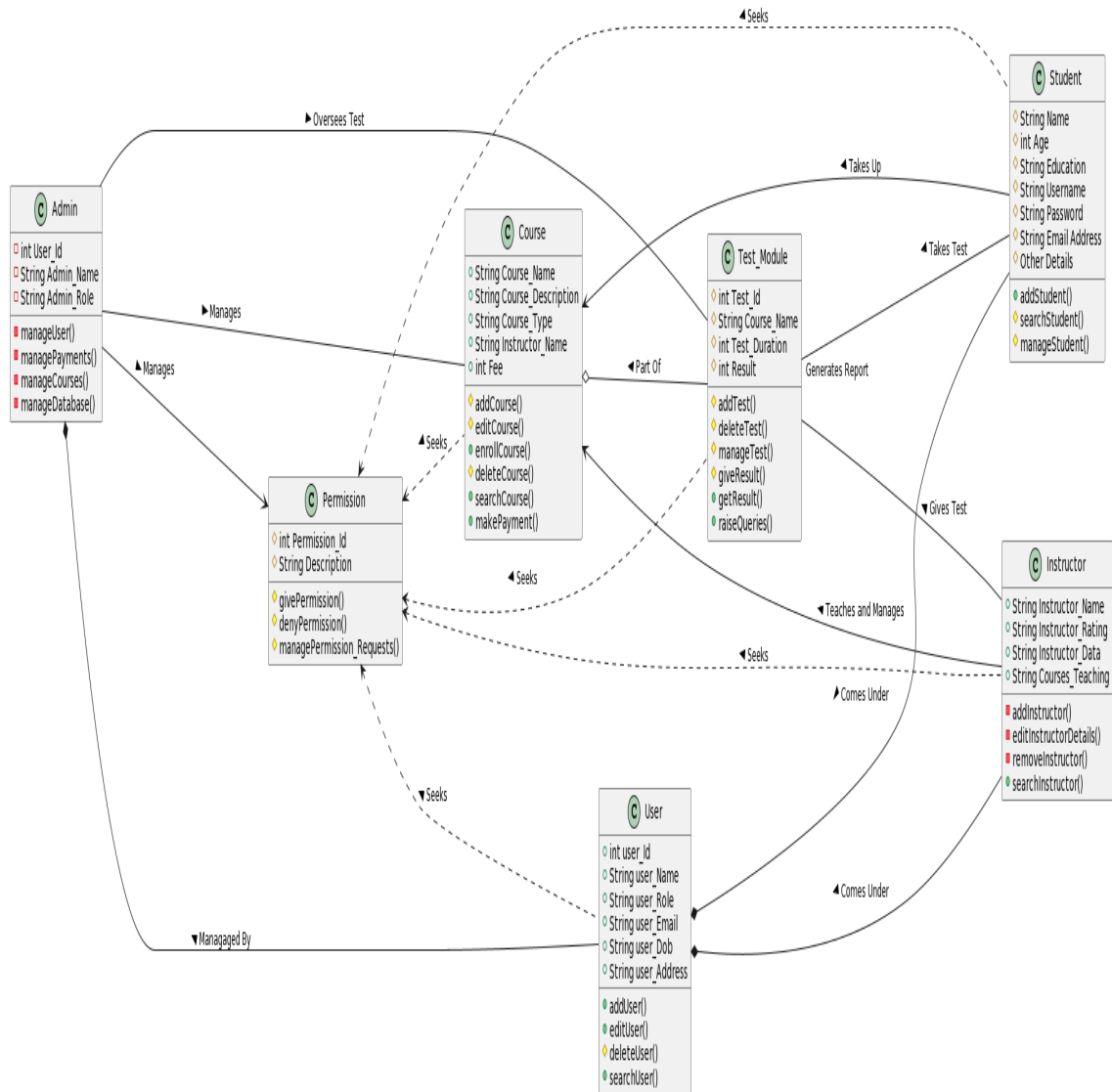
```

classTestModule
intTestId
StringCourseName
intTestDuration
intResult
addTest()
deleteTest()
manageTest()
giveResult()
+ getResult()
+ raiseQueries()

User * -- Student :< ComesUnder
User * -- Instructor :< ComesUnder
Admin * -- User :< ManagagedBy
Admin --- Course : Manages >
Permission < ..User :< Seeks
Permission < ..TestModule :< Seeks
Permission < ..Instructor :< Seeks
Permission < ..Student :< Seeks
Admin --- > Permission :> Manages
Permission < ..Course :< Seeks
Courseo -- TestModule : PartOf <
TestModule --- Instructor :< GivesTest
TestModule"GeneratesReport" --- Student :< TakesTest
TestModule --- Admin :< OverseesTest
Course < --- Instructor :< TeachesandManages
Course < --- Student :< TakesUp
@enduml

```

Class Diagram



Admin.java

```
import java.util.Vector;

public class Admin
public Integer UserId;
public String AdminName;
public String AdminRole;
public Vector Maneges;
public Vector Manages;
public Vector Maneged By;
public Vector Oversees Test;
public void manageUser()

public void managePayments()

public void manageCourses()

public void manageDatabase()
```


Coourse.java

```
import java.util.Vector;

public class Course extends TestModule, TestModule, TestModule
public String CourseName;
publicStringCourseDescription
; publicStringCourseType;
publicStringInstructorName;
publicIntegerFee;
publicVectorManeges;
publicVectormyTestModule;
publicVectormyTestModule;
publicVectorPartOf;
publicvoidaddCourse()
publicvoideditCourse()
publicvoidenrollCourse()
publicvoiddeleteCourse()
publicvoidsearchCourse()
publicvoidmakePayment()
```

Instructor.java

```
import java.util.Vector;

public class Instructor
public String InstructorName;
publicStringInstructorRating;
publicStringInstructorData;
publicStringCoursesTeaching;
publicVectorTeachesandManages;
publicVectorGiveTest;
publicVectormyUser;
publicVectorComesUnder;
publicvoidaddInstructor()
publicvoideditInstructorDetails()

    public void removeInstructor()

    public void searchInstructor()
```

Permission.java

```
public class Permission
public Integer PermissionId;
publicStringDescription;
publicvoidgivePermission()
publicvoiddenyPermission()
publicvoidmanagePermissionRequests()
```

Student.java

```
import java.util.Vector;

public class Student
public String Name;
public Integer Age;
public String Education;
public String Username;
public String Password;
public String EmailAddress;
publicStringDetails;
publicVectorTakesUp;
publicVectorTakeTest;
publicVectormyUser;
publicVectormyUser;
publicVectorComesUnder;
publicvoidaddStudent()
publicvoidsearchStudent()
publicvoidmanageStudent()
```

User.java

```
import java.util.Vector;

public class User

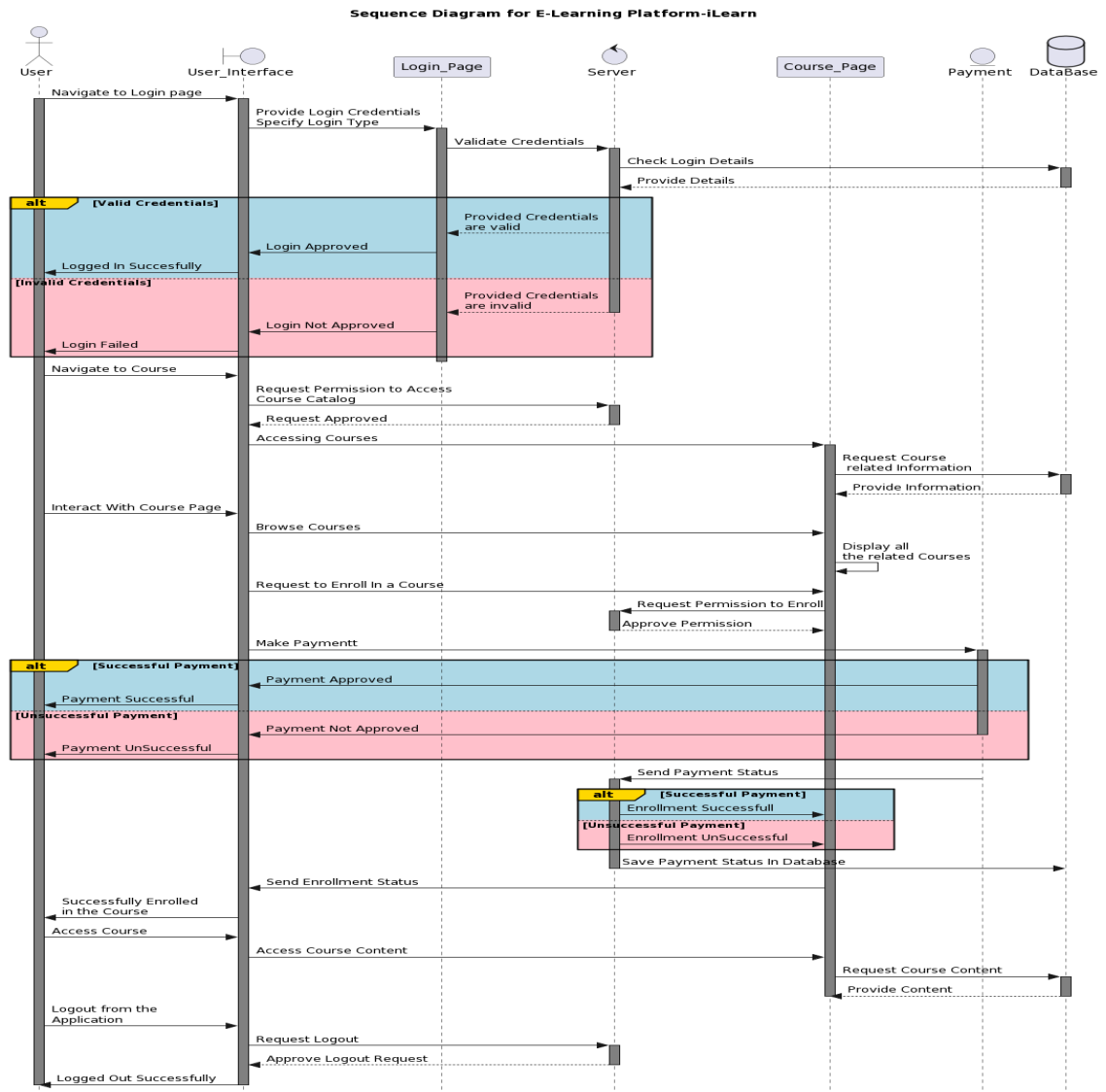
public Integer user_Id;

publicStringuser_Role;
publicStringuser_Email;
publicIntegeruser_Dob;
publicStringuser_Address;
publicVectormyStudent;
publicVectormyInstructor;
publicVectorComesUnder;
publicVectormyAdmin;
publicVectorManegedBy;
publicVectormyStudent;
publicVectorComesUnder;
publicvoidaddUser()
publicvoideditUser()
publicvoiddeleteUser()
publicvoidsearchUser()
```

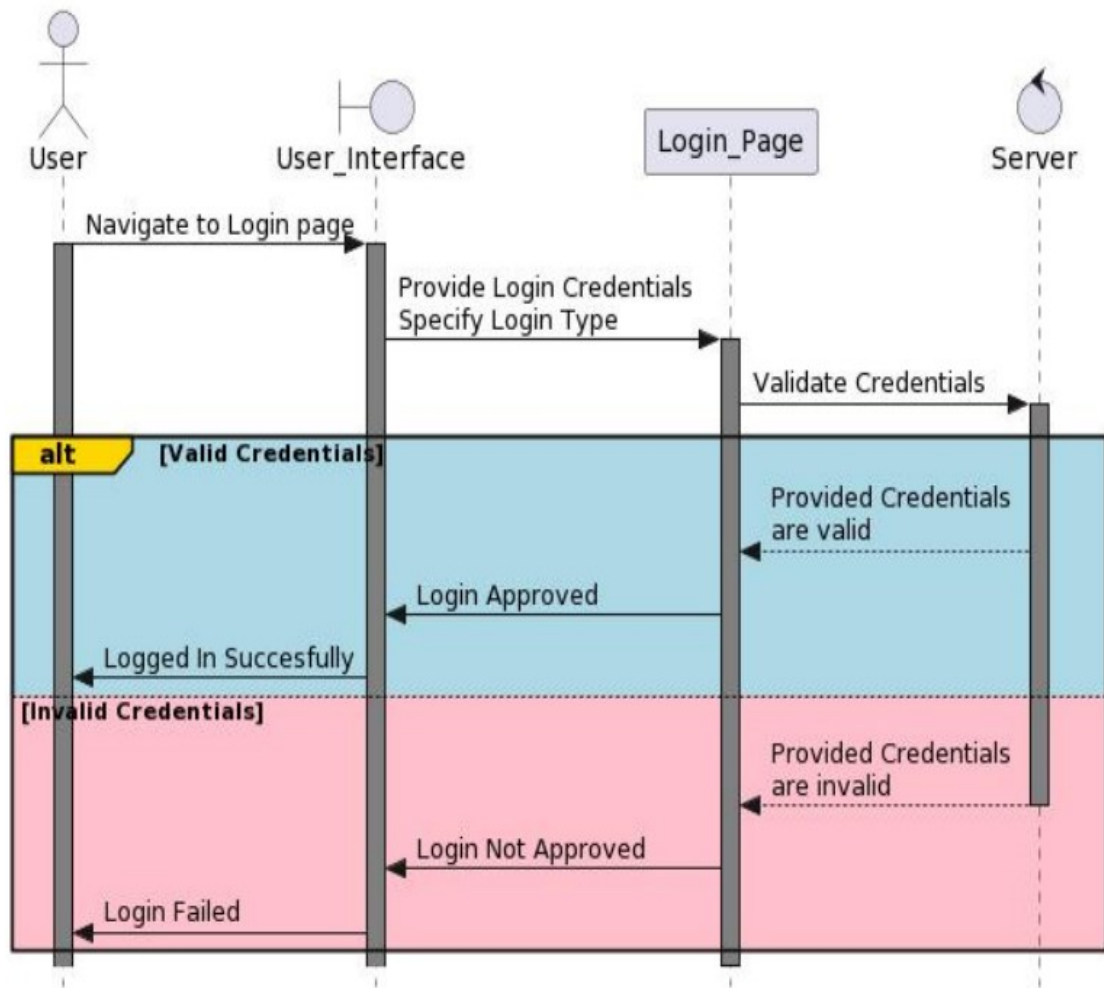
TestModule.java

```
import java.util.Vector;
public class TestModule
public Integer TestId;
publicStringCourseName;
publicIntegerTestDuration;
publicIntegerResult;
publicVectorTakeTest;
publicVectorGiveTest;
publicAdminOverseasTest;
publicVectormyCourse;
publicVectorPartOf;
publicVectorOverseesTest;
publicvoidaddTest()
publicvoiddeleteTest()
publicvoidmanageTest()
publicvoidgiveResult()
publicvoidgetResult()
publicvoidraiseQueries()
```

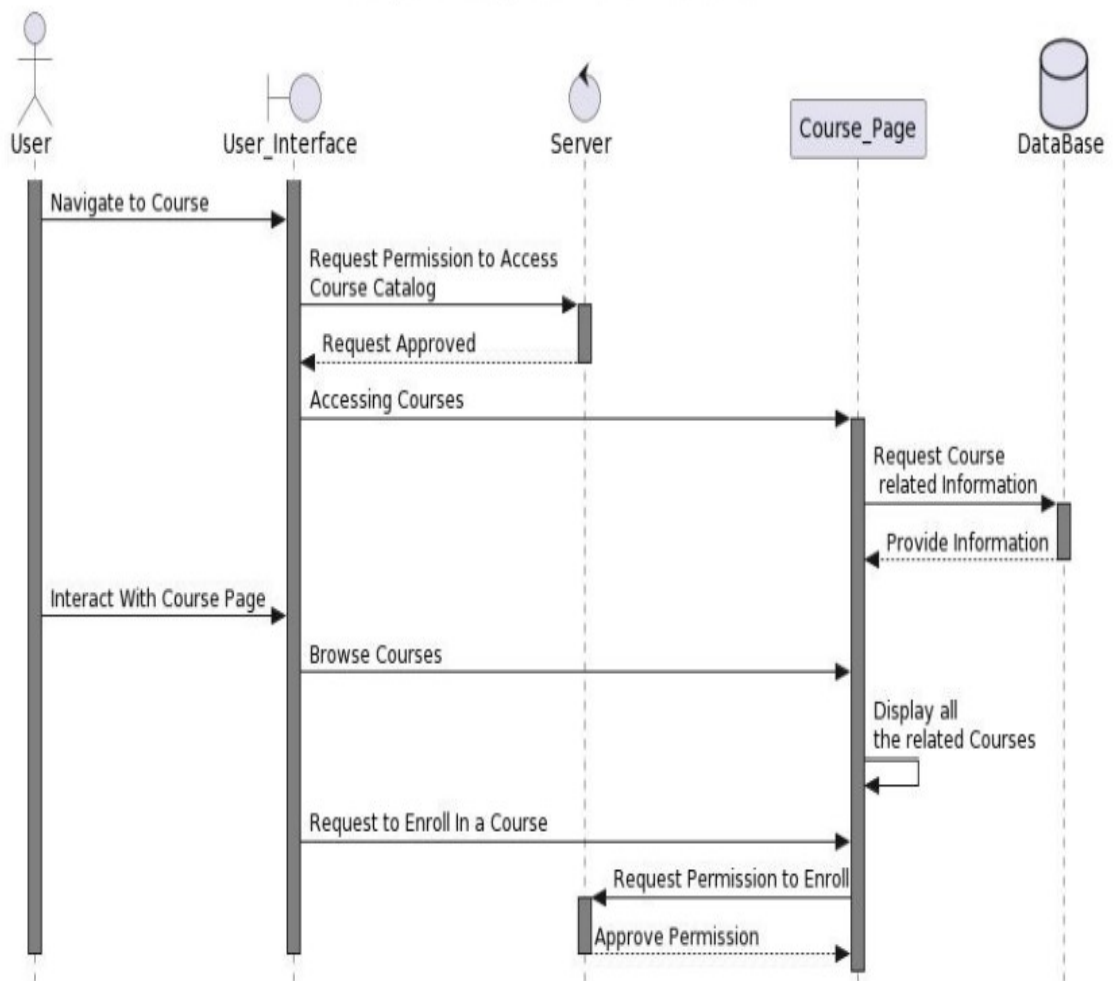
Sequence Diagrams



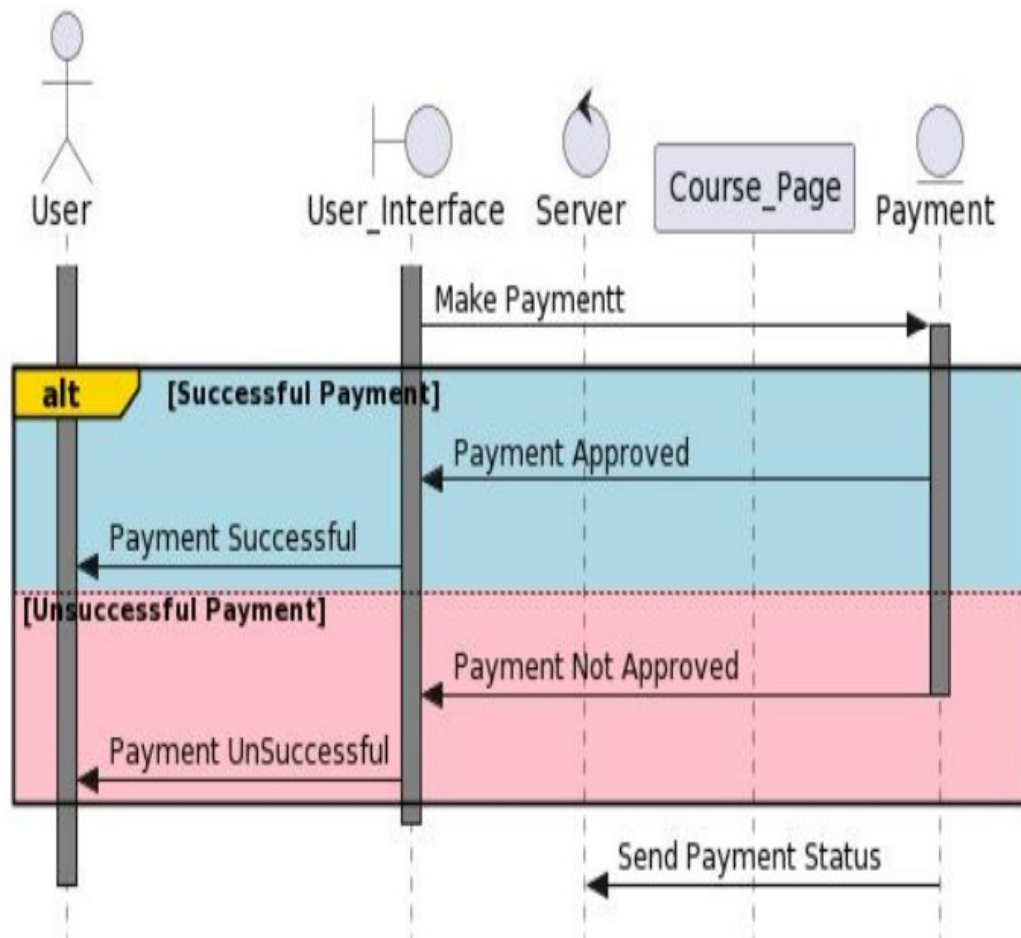
Sequence Diagram for Login Interface



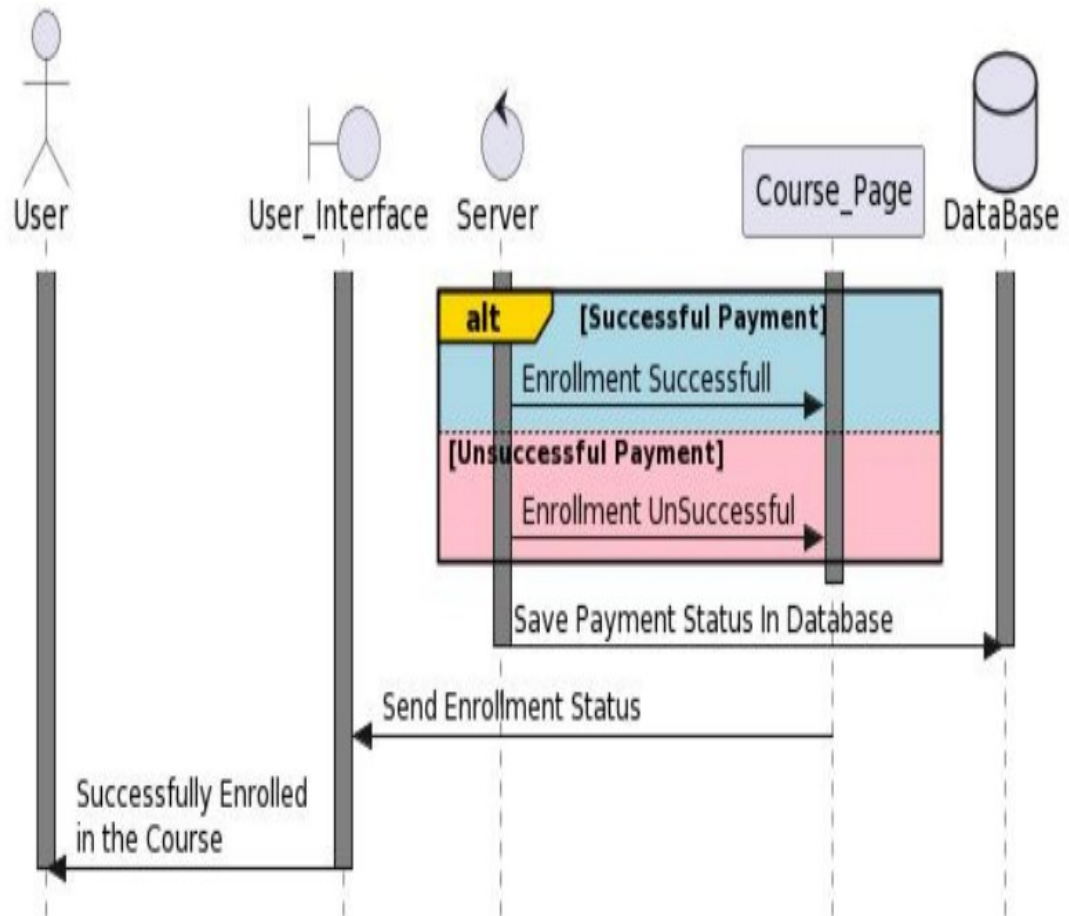
Sequence Diagram for Course Acceses



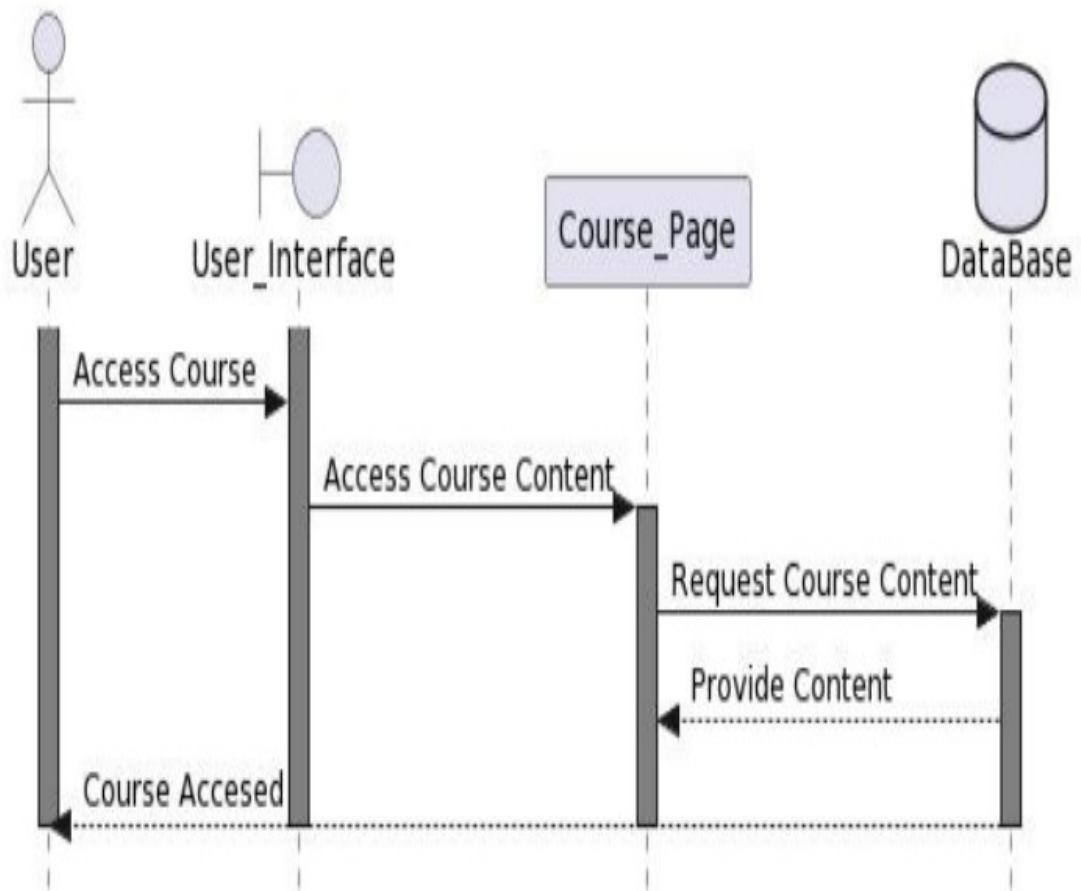
Sequence Diagram for Course Payment



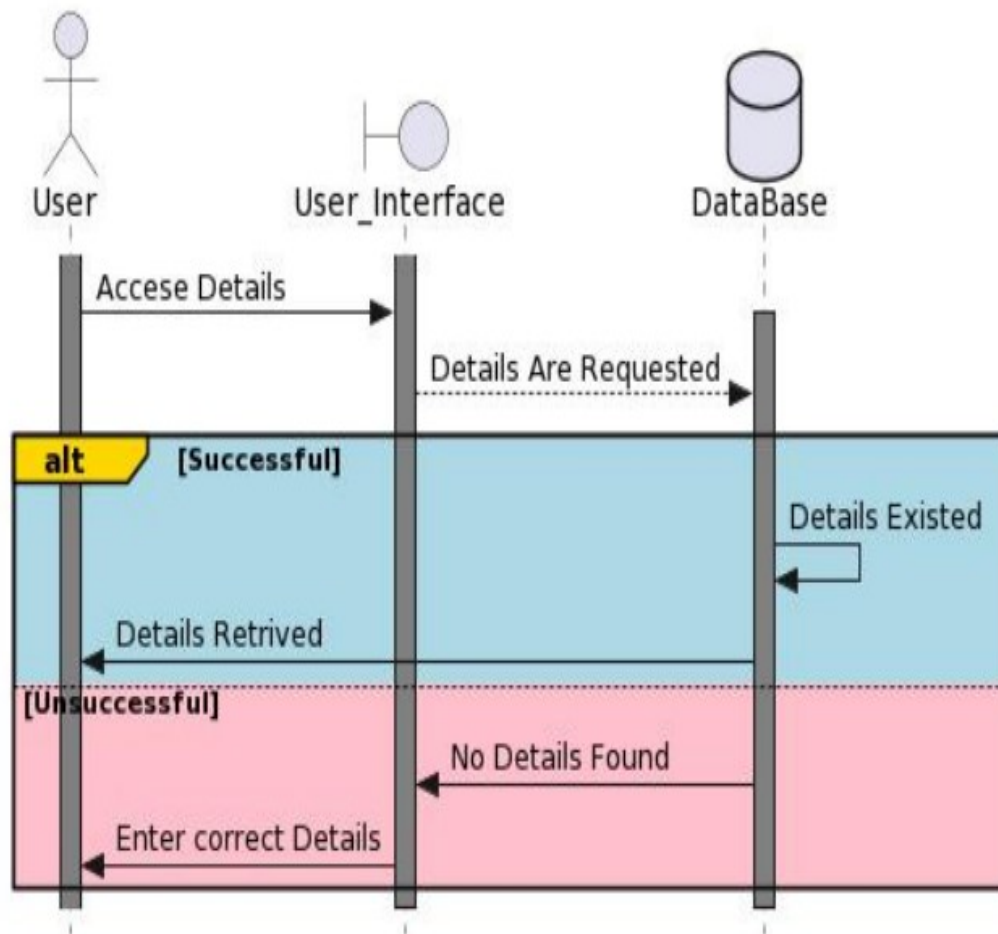
Sequence Diagram for Course Enrollment



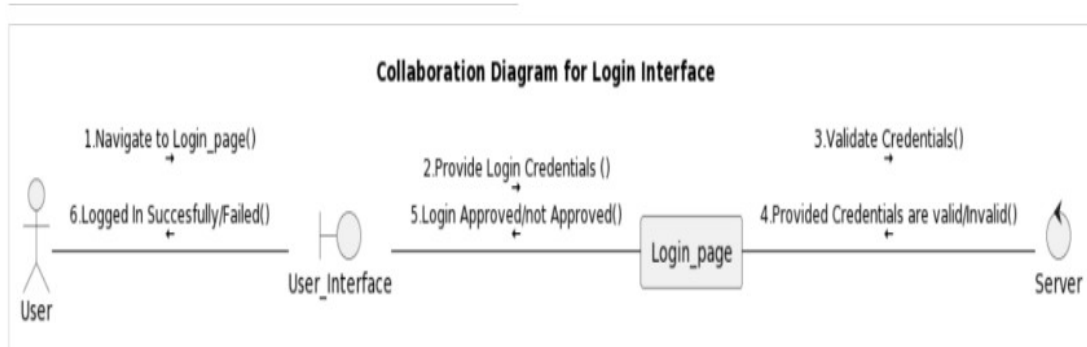
Sequence Diagram for Course Accesses

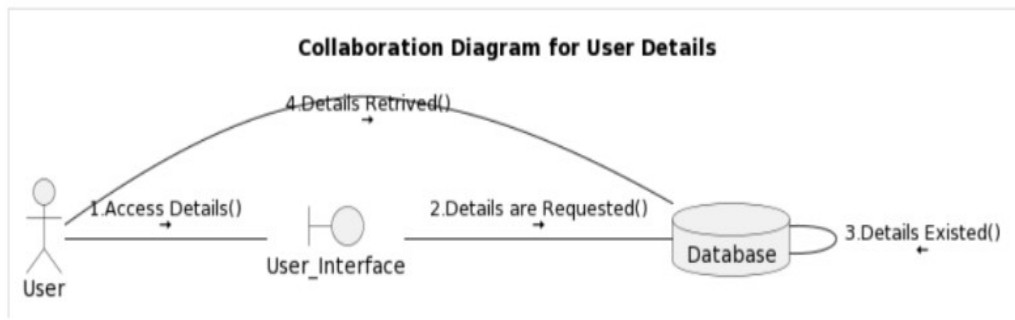


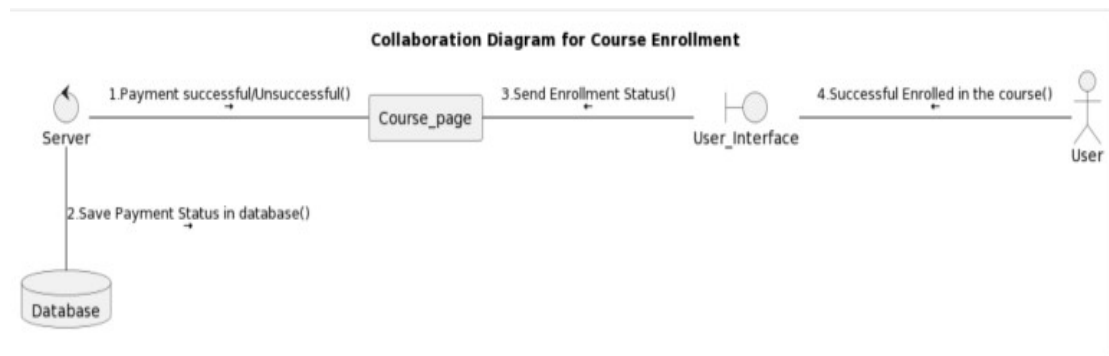
Sequence Diagram for User Details

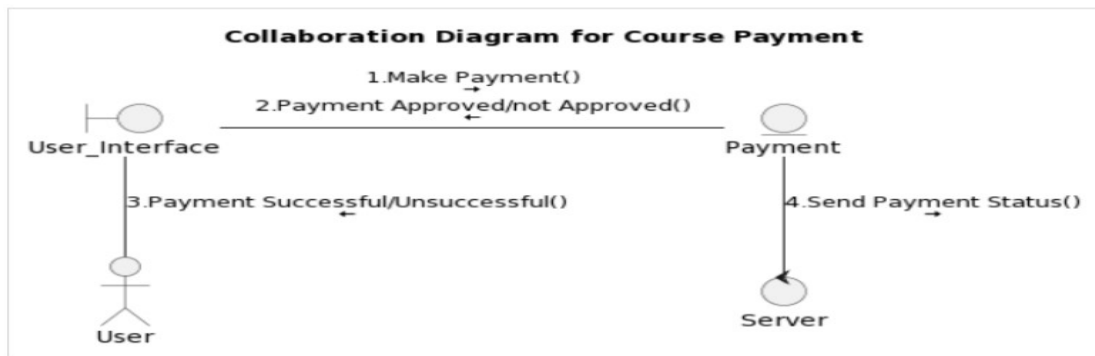


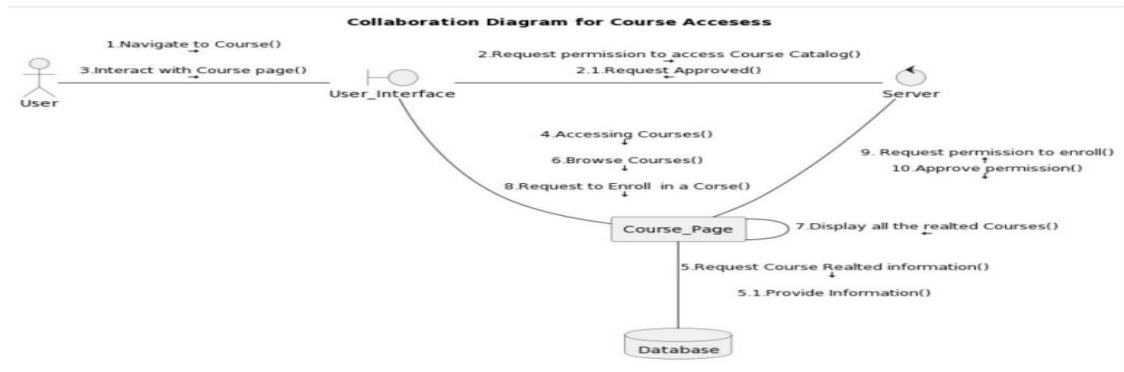
Collaboration Diagrams

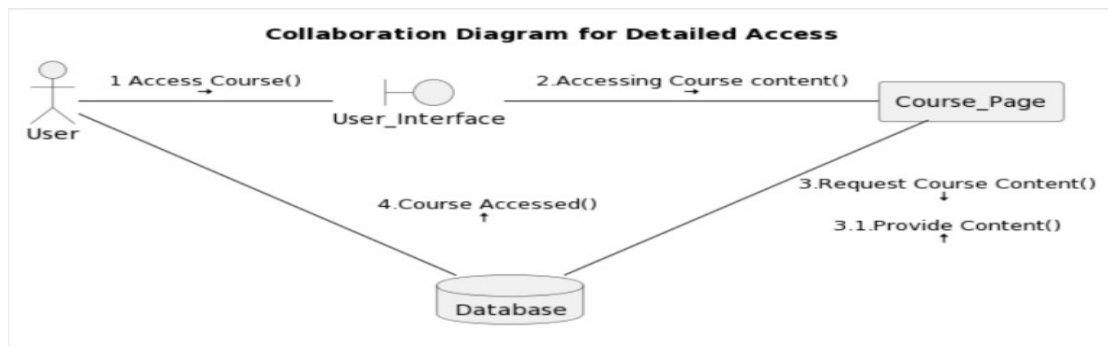












COCOMO Model

Aim:

To estimate the cost of the iLearn The Digital Learning Environment by using COCOMO Model.

Description:

COCOMO(Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models. The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort Schedule.

Effort:

Amount of labor that will be required to complete a task. It is measured in person months units.

Schedule:

Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, months. Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

Boehm's definition of organic, semidetached, and embedded systems: Organic – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience xxix regarding the problem. Semi-detached – A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, knowledge of the various programming environment.lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity.

Eg:

Compilers or different Embedded Systems can be considered of Semi-Detached type.

Embedded –

A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models. All the above system types utilize different values of the constants used in Effort Calculations.

Types of Models:

COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are three types of COCOMO model: Basic COCOMO Model Intermediate COCOMO Model Detailed COCOMO Model The first level, Basic COCOMO can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations. Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases, i.e in case of Detailed it accounts for both these cost drivers and also calculations are performed phase-wise henceforth producing a more accurate result. These two models are further discussed below

Estimation of Effort: Calculations – Basic Model

– $E = a(KLOC)^b$ time = $c(Effort)^d$ Personrequired = Effort/time The above formula is used for the cost estimation of the basic COCOMO model, and also is used in the subsequent

The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. The development time is measured in months. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.

Intermediate Model – The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software systems. However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their attributes: (i) Product attributes –

Required software reliability extent xxxi Size of the application database The complexity of the product

(ii) Hardware attributes –

Run-time performance constraints Memory constraints The volatility of the virtual machine environment Required turnabout time

(iii) Personnel attributes

Analyst capability Software engineering capability Applications experience Virtual machine experience Programming language experience

(iv) Project attributes –

Use of software tools Application of software engineering methods Required development schedule

Detailed Model –

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort. The Six phases of detailed COCOMO are: Planning and requirements xxxii System design Detailed design Module code and test Integration and test Cost Constructive model The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.

YOUR BASIC COCOMO RESULTS!!								
MODE	"A" variable	"B" variable	"C" variable	"D" variable	KLOC	EFFORT, (in person/months)	DURATION, (in months)	STAFFING, (recommended)
organic	2.4	1.05	2.5	0.38	14	38.33943241241644	9.993678334925779	3.8363684648952816

Explanation: The coefficients are set according to the project mode selected on the previous page, (as per Boehm,81). The final estimates are determined in the following manner:

effort = $a * KLOC^b$, in person/months, with KLOC = lines of code, (in the thousands), and:


duration = $c * effort^d$, finally:

staffing = effort/duration

For further reading, see Boehm, "Software Engineering Economics", (81)

WARNING: If you see "NaN" in any field above, you have entered an **INVALID** value for KLOC!! Hit the "BACK" button on your browser, hit the "RESET" button, and enter a **DECIMAL NUMBER** in the KLOC input text box!

Thank you, and happy software engineering!



Function Point Analysis

AIM:

To calculate the Function point for the i learn Digital learning Environment

DESCRIPTION:

Function Point Analysis was initially developed by Allan J. Albercht in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG). The initial Definition is given by Allan J. Albrecht: FPA gives a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer.

FPA provides a standardized method to functionally size the software work product. This work product is the output of software new development and improvement projects for subsequent releases. It is the software that is relocated to the production application at project implementation. It measures functionality from the user's point of view i.e. on the basis of what the user requests and receives in return. Function Point Analysis (FPA) is a method or set of rules of Functional Size Measurement. It assesses the functionality delivered to its users, based on the user's external view of the functional requirements. It measures the logical view of an application, not the physically implemented view or the internal technical view.

The Function Point Analysis technique is used to analyze the functionality delivered by software and Unadjusted Function Point (UFP) is the unit of measurement.

Objectives of FPA:

The objective of FPA is to measure the functionality that the user requests and receives. The objective of FPA is to measure software development and maintenance independently of the technology used for implementation. It should be simple enough to minimize the xxxv overhead of the measurement process. It should be a consistent measure among various projects and organizations.

Types of FPA:

Transactional Functional Type –

External Input (EI): EI processes data or control information that comes from outside the application's boundary. The EI is an elementary process. External Output (EO): EO is an elementary process that generates data or control information sent outside the application's boundary. External Inquiries (EQ): EQ is an elementary process made up of an

input-output combination that results in data retrieval.

Data Functional Type –

Internal Logical File (ILF): A user identifiable group of logically related data or control information maintained within the boundary of the application.

External Interface File (EIF):

A group of users recognizable logically related data allusion to the software but maintained within the boundary of another software.

Counting Function Point (FP):

Step-1:

$F = 14 * \text{scale}$ Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF).

Below table shows scale:

0 - No Influence

1 - Incidental

2 - Moderate

3 - Average

4 - Significant

5 - Essential

Step-2:

Calculate Complexity Adjustment Factor (CAF).

$$\text{CAF} = 0.65 + (0.01 * F)$$

Step-3:

Calculate Unadjusted Function Point (UFP).

Step-4:

Calculate Function Point.

$$\text{FP} = \text{UFP} * \text{CA}$$

Function Point Counter

For help, please refer to the right side of the window.

Measurement Parameter	Count		Weighting factor		
number of user inputs	<input type="text" value="10"/>	X	<input type="text" value="average"/>	=	<input type="text" value="40"/>
number of user outputs	<input type="text" value="13"/>	X	<input type="text" value="average"/>	=	<input type="text" value="65"/>
number of user inquiries	<input type="text" value="3"/>	X	<input type="text" value="average"/>	=	<input type="text" value="12"/>
number of files	<input type="text" value="7"/>	X	<input type="text" value="average"/>	=	<input type="text" value="70"/>
number of external interfaces	<input type="text" value="3"/>	X	<input type="text" value="average"/>	=	<input type="text" value="21"/>
Total Count					<input type="text" value="208"/>

Complexity Adjustment Values		
1	Does the system require reliable backup and recovery?	<input type="text" value="Essential"/>
2	Are data communications required?	<input type="text" value="Moderate"/>
3	Are there distributed processing functions?	<input type="text" value="Incidental"/>
4	Is Performance critical?	<input type="text" value="Significant"/>
5	Will the system run in an existing heavily utilized operational environment?	<input type="text" value="Significant"/>
6	Does the system require online data entry?	<input type="text" value="Essential"/>
7	Does the online data entry require the input transaction to be built over multiple screens or operations?	<input type="text" value="Average"/>
8	Are the master files updated online?	<input type="text" value="Average"/>
9	Are the inputs, outputs, files, or inquiries complex?	<input type="text" value="Moderate"/>
10	Is the internal processing complex?	<input type="text" value="Average"/>
11	Is the code designed to be reusable?	<input type="text" value="Essential"/>
12	Are conversion and installation included in the design?	<input type="text" value="Incidental"/>
13	Is the System designed for multiple installations in different organizations?	<input type="text" value="Moderate"/>
14	Is the application designed to facilitate change and ease of use by the user?	<input type="text" value="Essential"/>

Total Function Point Count:

Gantt Chart

AIM:

Illustrate the use of the Gantt project. for i learn digital Environment.

DESCRIPTION:

A gantt chart is a horizontal bar chart used in project management to visually represent a project plan over time. Modern gantt charts typically show you the timeline and status—as well as who's responsible—for each task in the project.

Here's a quick look at the details a gantt chart enables you to capture at a glance:

How a project breaks down into tasks
When each task will begin and end
How long each task will take
Who's assigned to each task
How tasks relate to and depend on each other
When important meetings, approvals, or deadlines need to happen
How work is progressing in a project
The full project schedule from start to finish
In other words, a gantt chart is a super-simple way to communicate what it will take to deliver a project on time and budget. That means it's a whole lot easier to keep your project team and stakeholders on the same page from the get-go.

Gantt charts may seem complicated at first. But once you learn the basics, you'll be able to read and create a gantt chart easily and tell exactly where your projects are and what needs to happen to guide them to success. Elements of a gantt chart
Reading a gantt chart really comes down to understanding how the different elements come together to make a gantt chart work.

Let's review some basic terminology so you understand the key parts of a gantt chart and how they function in a project plan:

Task list: Runs vertically down the left of the gantt chart to describe project work and may be organized into groups and subgroups.

Timeline: Runs horizontally across the top of the gantt chart and shows months, weeks, days, and years.

Dateline: A vertical line that highlights the current date on the gantt chart.

Bars: Horizontal markers on the right side of the gantt chart that represent tasks and show progress, duration, and start and end dates.

Milestones: Yellow diamonds that call out major events, dates, decisions, and deliverables

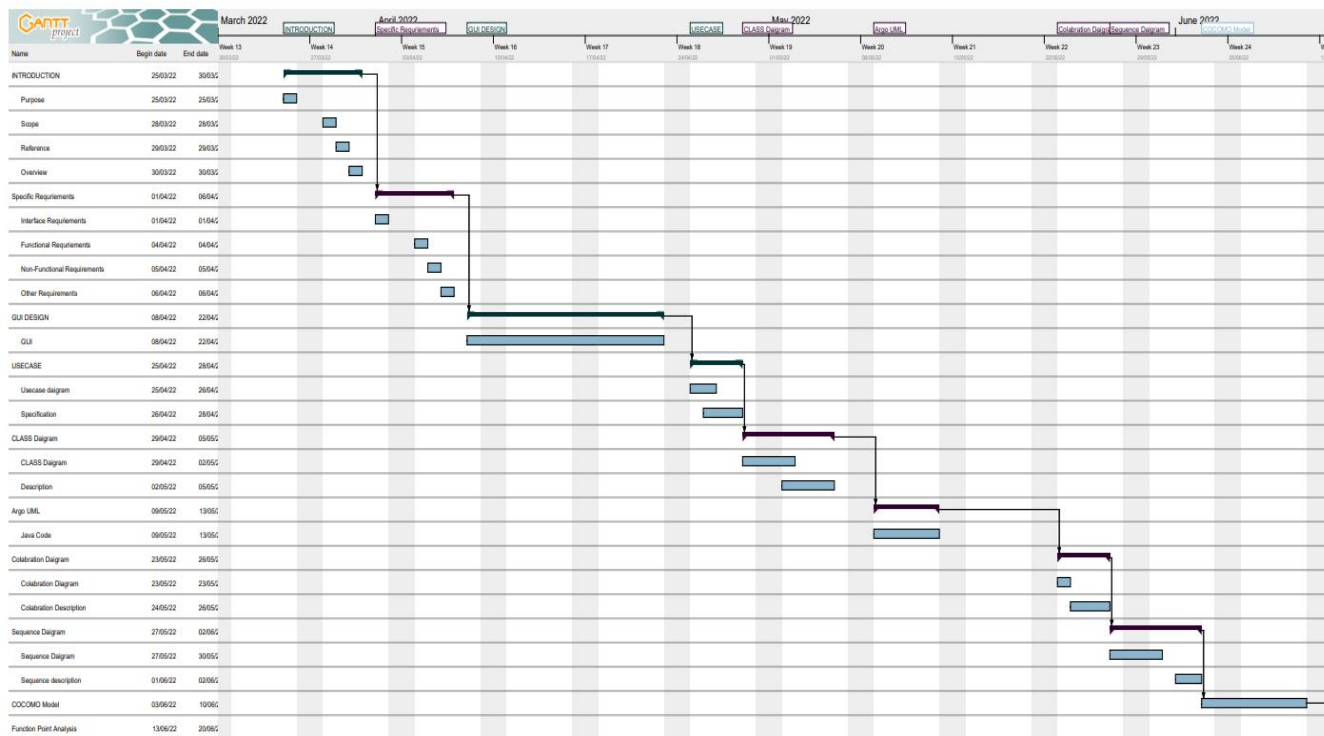
Dependencies: Light gray lines that connect tasks that need to happen in a certain order.

Progress: Shows how far along work is and may be indicated by percent complete and/or bar shading.

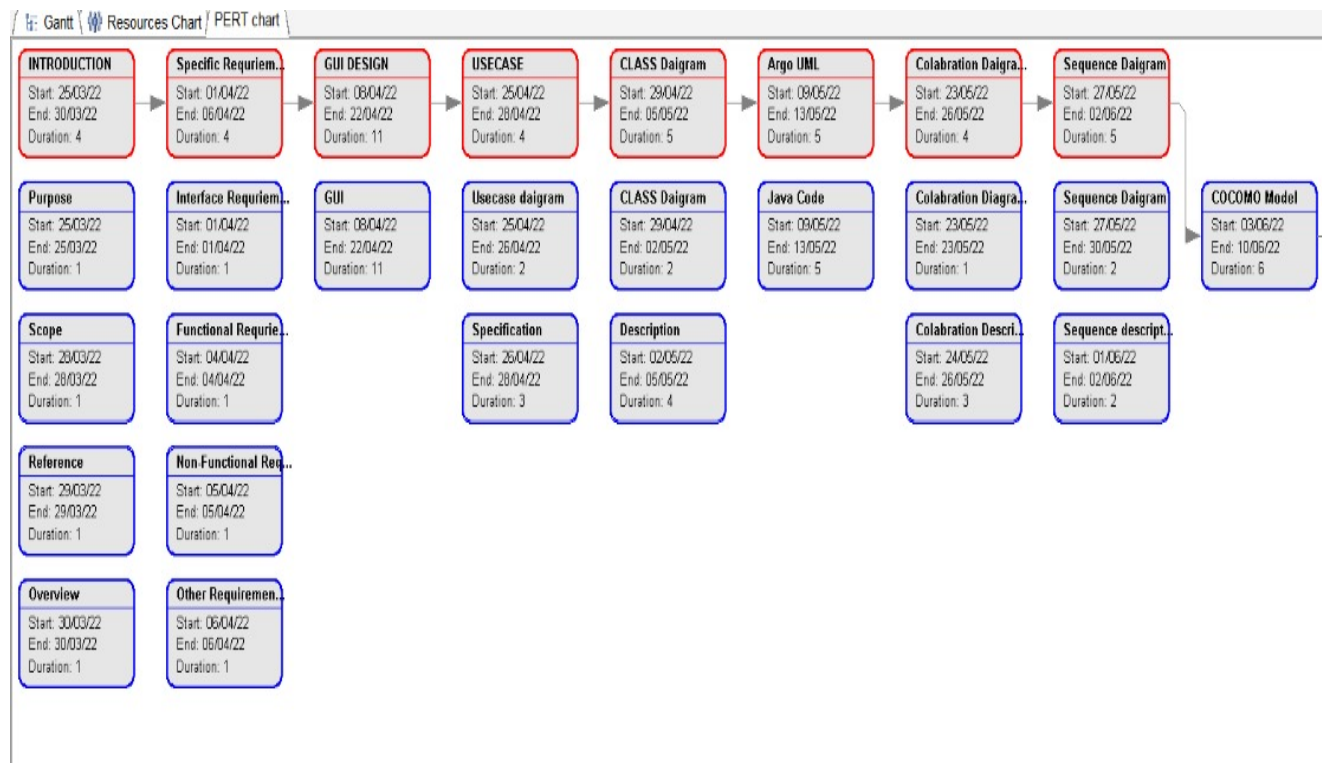
Resource assigned: Indicates the person or team responsible for completing a task Here's a gantt chart with these components highlighted.

This gantt chart highlights 8 features every gantt chart should include:

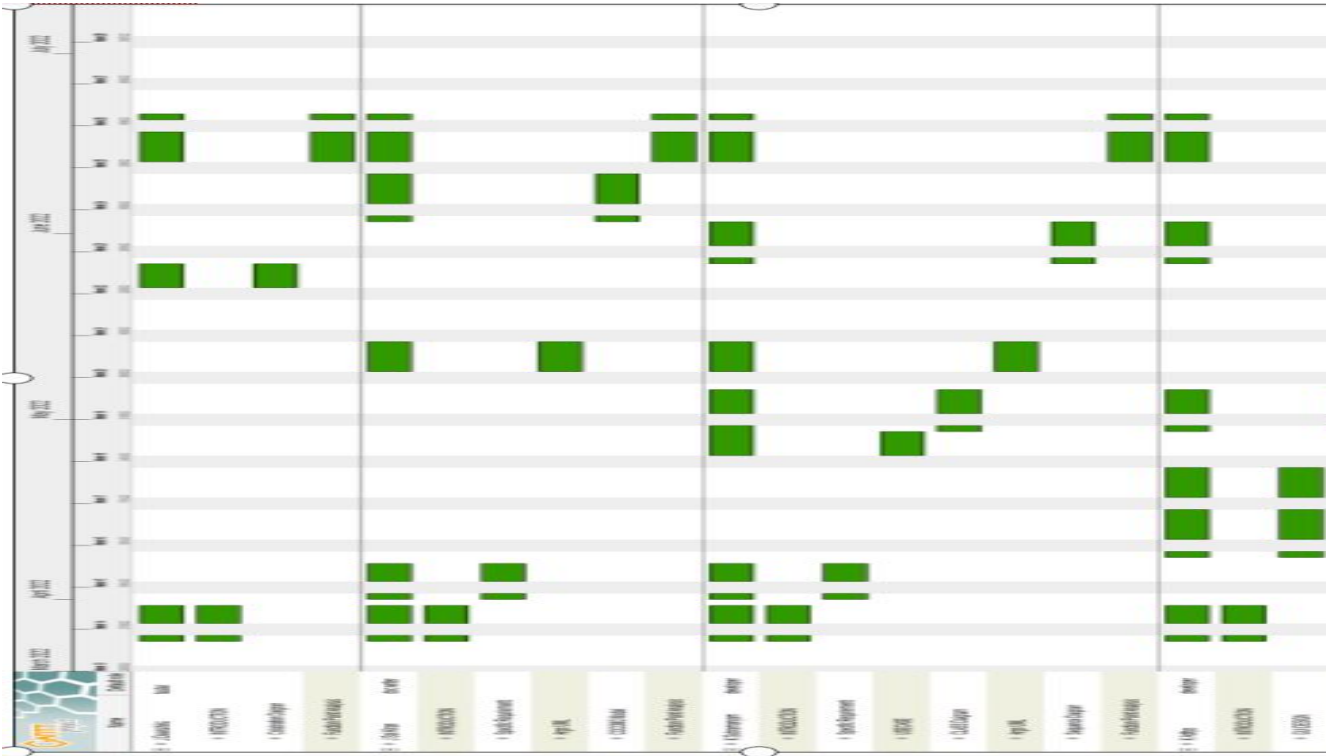
Gantt Chart



Pert chart



Resource chart



ER Diagram

Aim:

To Draw E-R diagrams, DFD for “I Learn Digital Learning Environment”.

Description:

An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation that depicts relationships among people, objects, places, concepts or events within an information technology (IT) system. An ERD uses data modeling techniques that can help define business processes and serve as the foundation for a relational database.

Purpose of ERD

oThe database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD. oThe ERD serves as a documentation tool. oFinally, 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.

Importance of ERDs and their uses

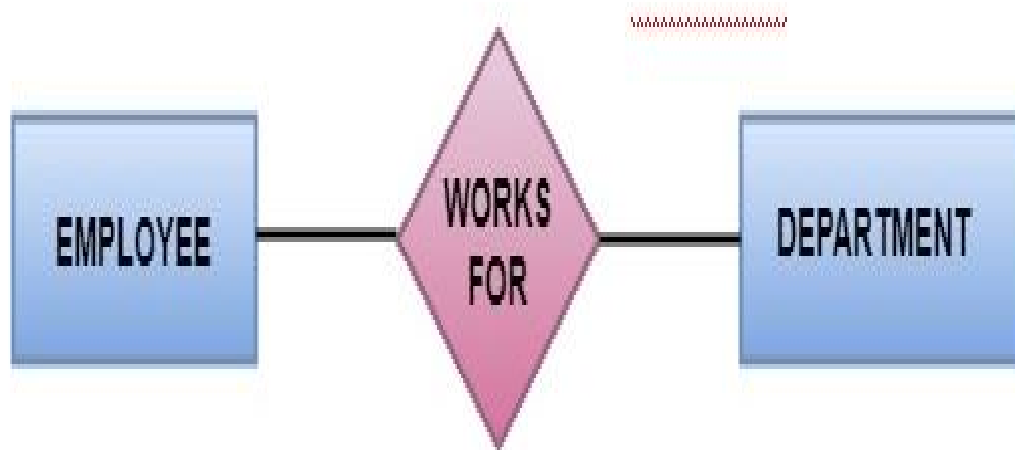
Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a reference point, should any debugging or business process re-engineering be needed later Components of an ER Diagrams

1. 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.

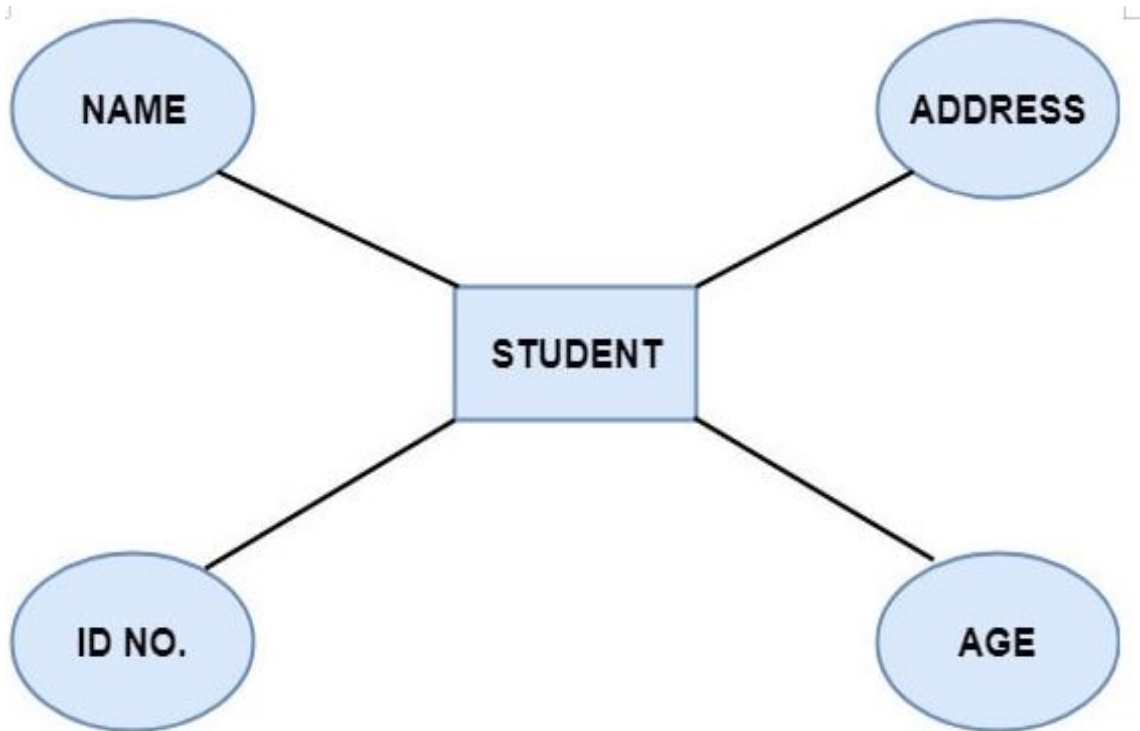
Entity Set

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.



2. 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.

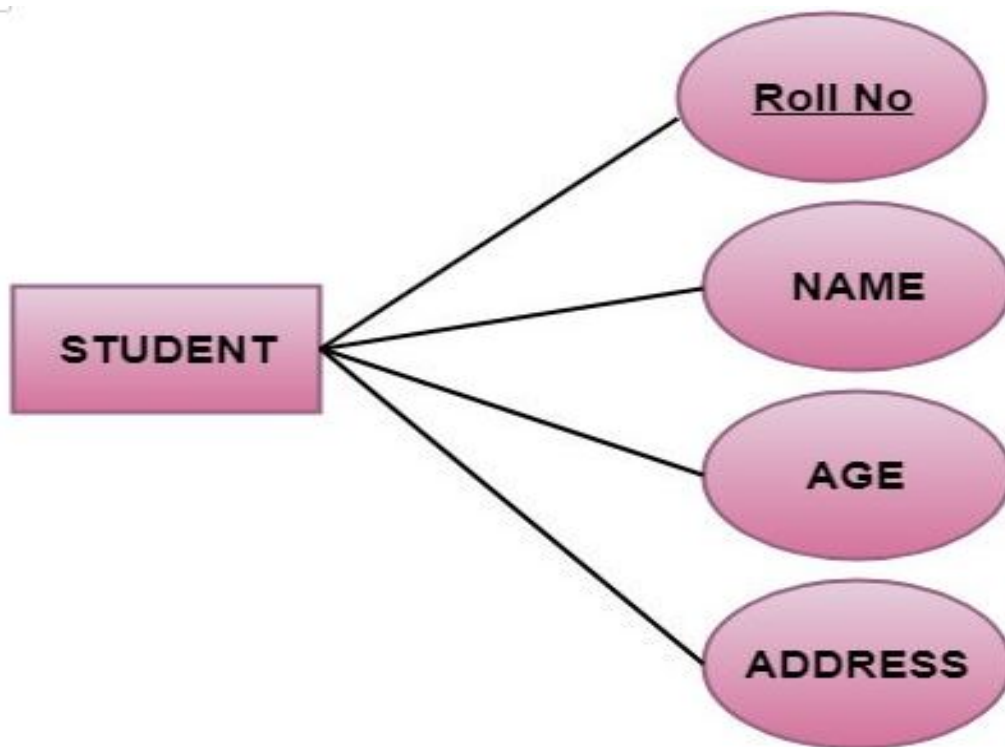


There are four types of Attributes:

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

1. Key attribute:

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



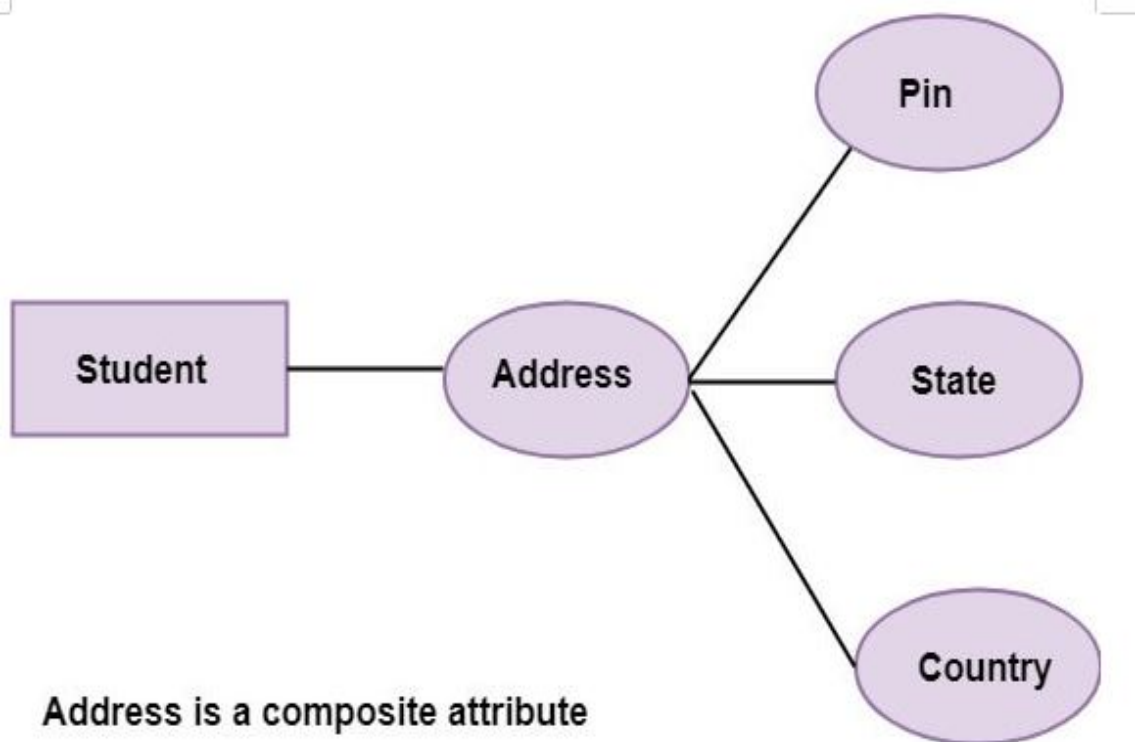
There are mainly three types of keys:

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.

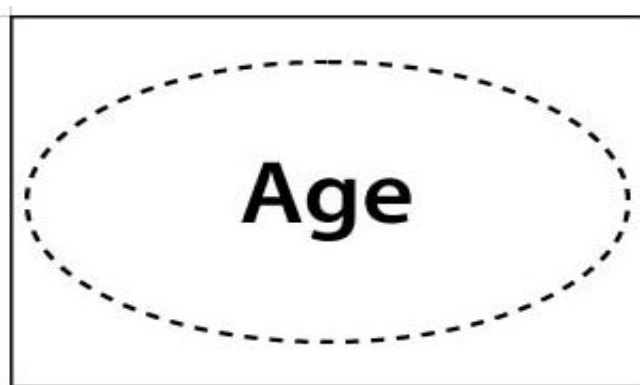
2. Composite attribute:An attribute that is a combination of other attributes is called a composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other characteristics such as pin code, state, country.



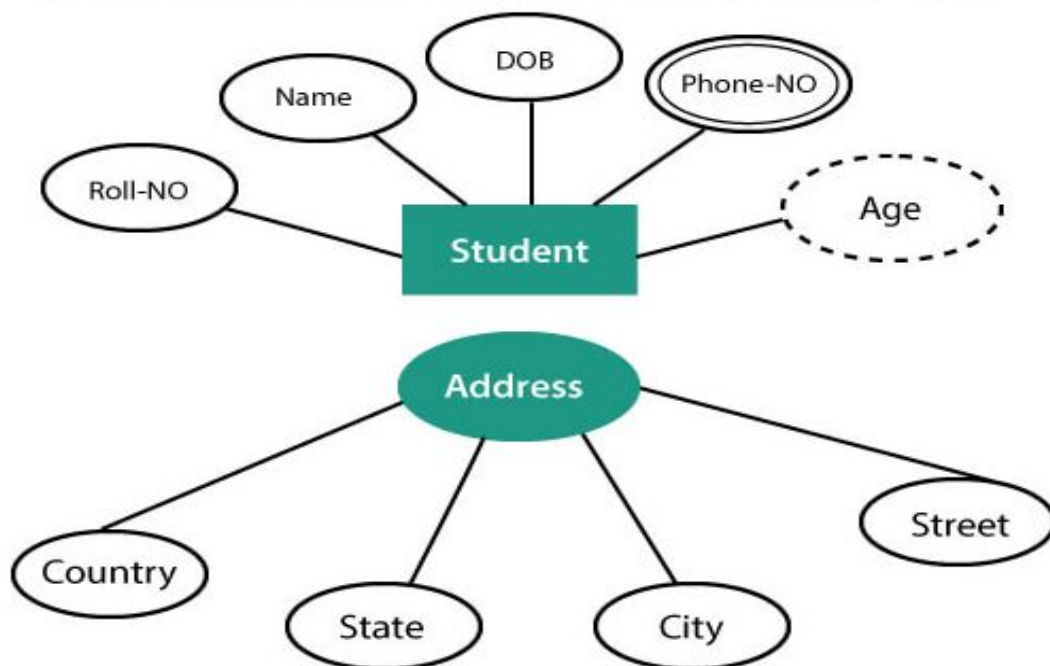
3. Single-valued attribute:Single-valued attribute contain a single value. For example, SocialSecurityNumber.

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 dateofbirth. In the ER diagram, Derived attributes are depicted by the dashed ellipse.



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



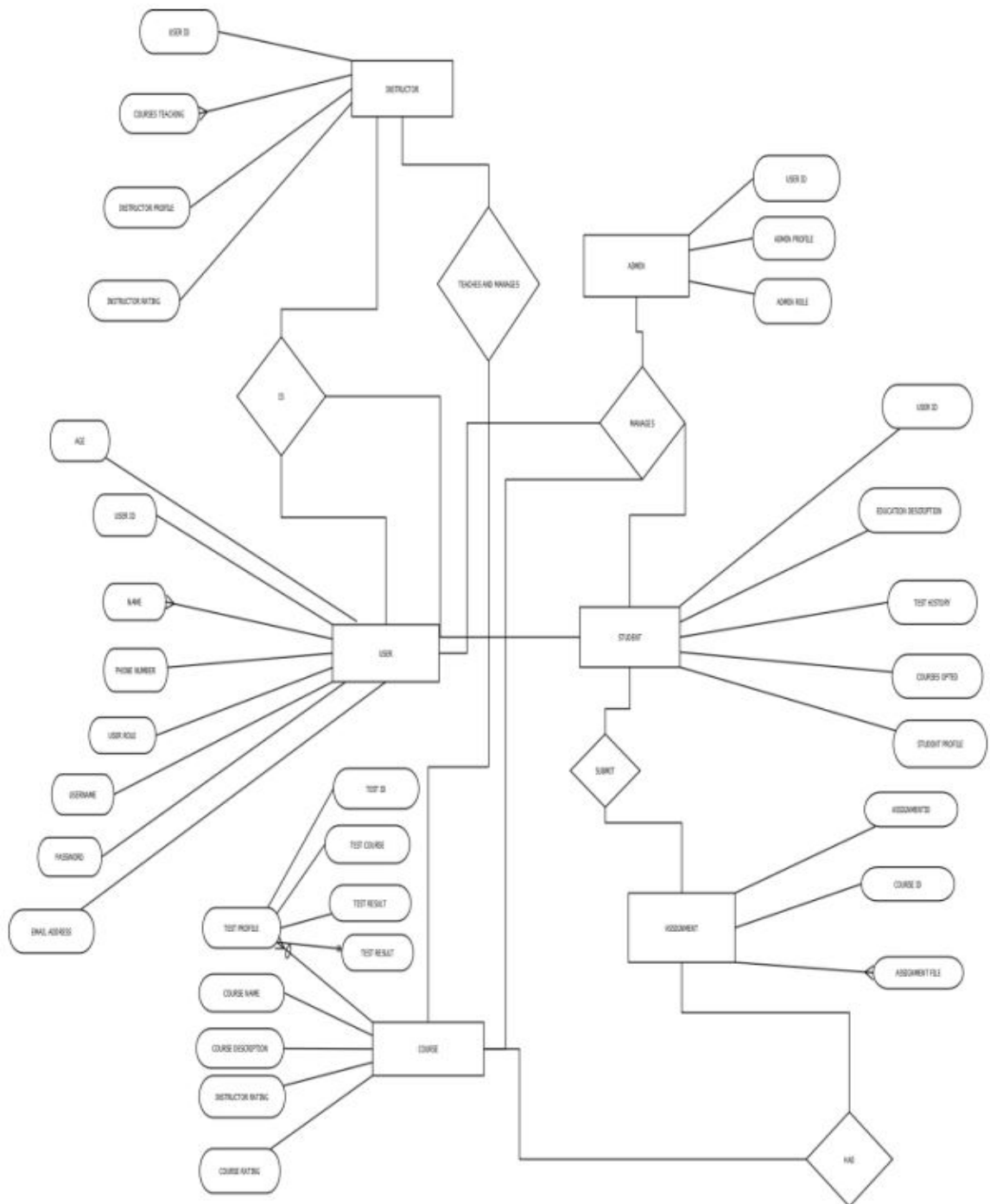
3. Relationships

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

ER Diagram:



DFD Diagram

Description:

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.

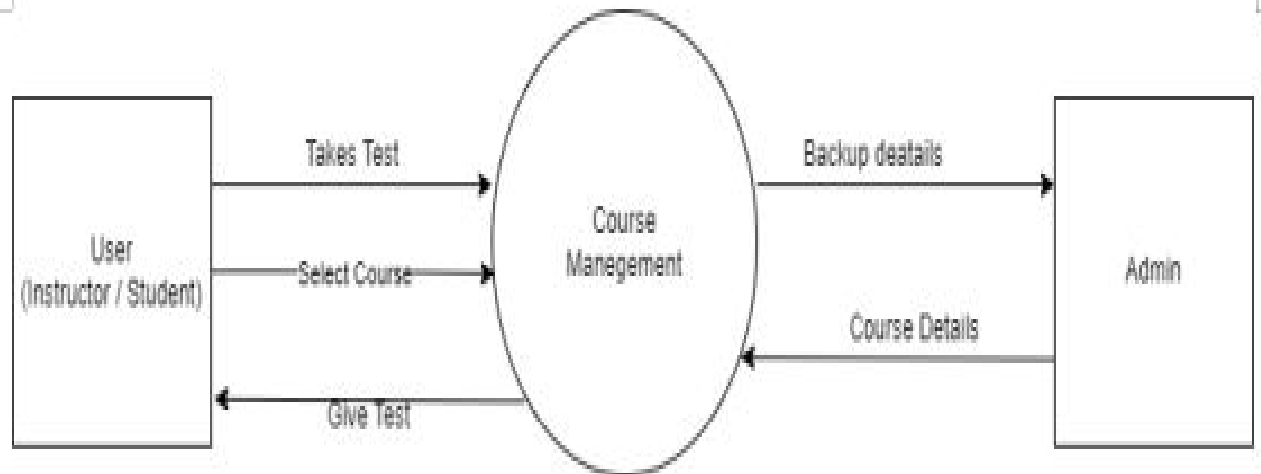
The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble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

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

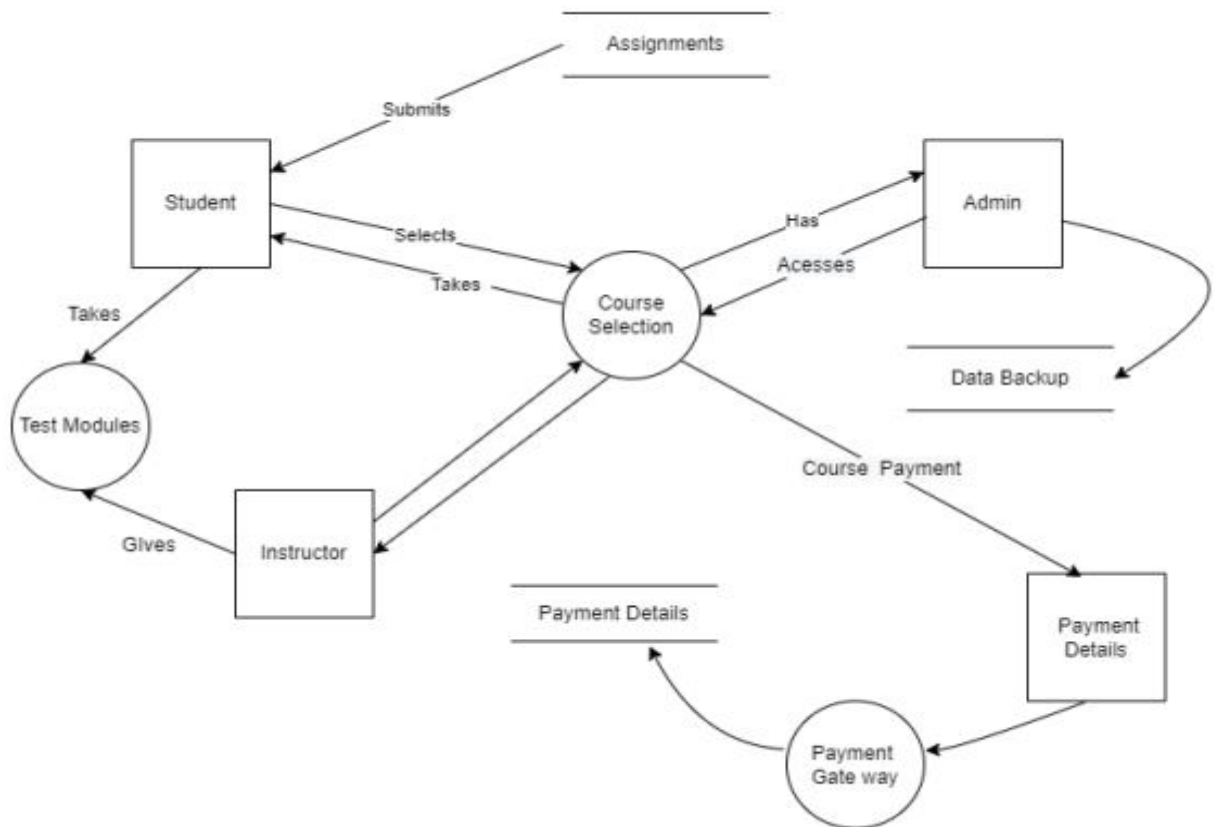
It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented 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:



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.

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 subprocesses.



Unit Testing

Aim:

To Design of Test cases based on requirements and design for “ I Learn Digital Learning Environment” using Unit Testing.

Description:

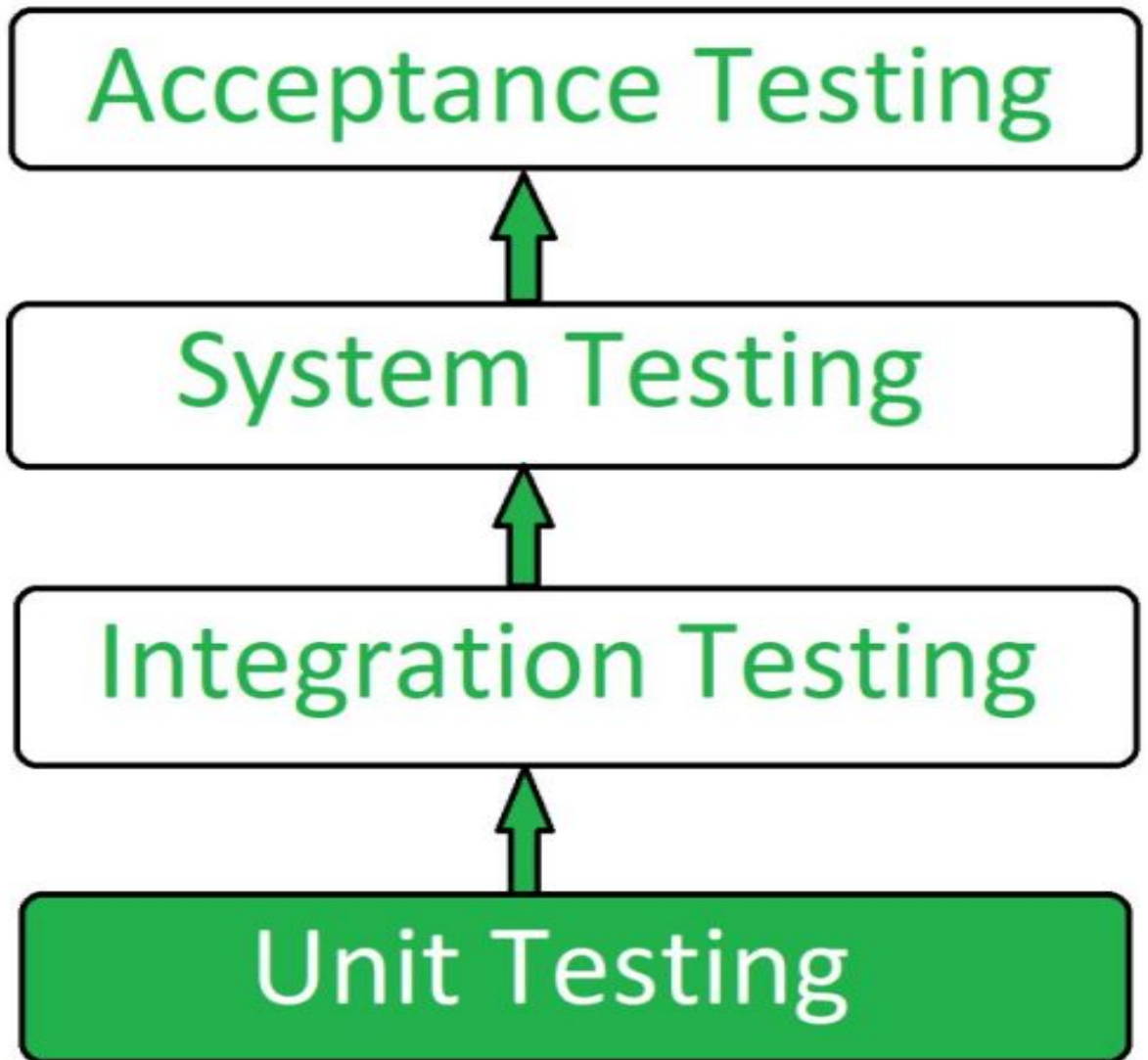
A unit test is a way of testing a unit - the smallest piece of code that can be logically isolated in a system. In most programming languages, that is a function, a subroutine, a method or property. The isolated part of the definition is important. In his book ”Working Effectively with Legacy Code”, author Michael Feathers states that such tests are not unit tests when they rely on external systems: “If it talks to the database, it talks across the network, it touches the file system, it requires system configuration, or it can’t be run at the same time as any other test.”

Software developers have been testing software for as long as they have been writing code, but the ability to automate software testing appeared around the 1980s. (Testing references provides a handy timeline of software testing.) Suddenly, instead of having to run a program manually against lists of test values, or setting breakpoints in the code and tracing the program’s logic step by step, developers could do what they do best — turn testing into another program.

Objective of Unit Testing:

The objective of Unit Testing is:

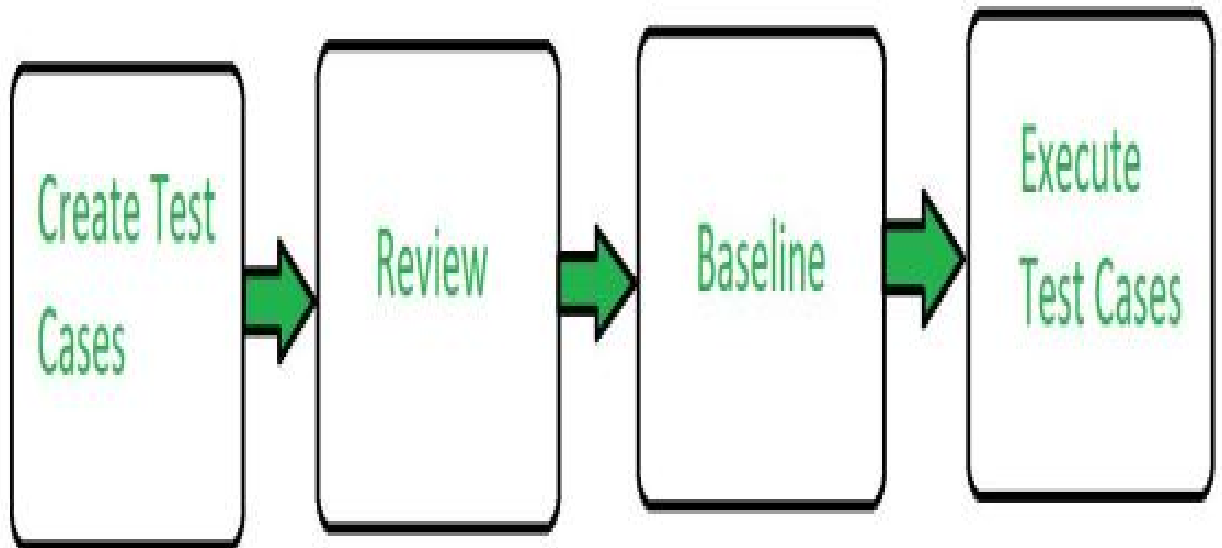
- 1.To isolate a section of code.
- 2.To verify the correctness of the code.
- 3.To test every function and procedure.
- 4.To fix bugs early in the development cycle and to save costs.
- 5.To help the developers to understand the code base and enable them to make changes quickly.
- 6.To help with code reuse.



Types of Unit Testing:

There are 2 types of Unit Testing:

1. Manual
2. Automated.

Workflow of Unit Testing:

Unit Testing Techniques:

There are 3 types of Unit Testing Techniques. They are

1.Black Box Testing:This testing technique is used in covering the unit tests for input, user interface, and output parts.

2.White Box Testing:This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.

3.Gray Box Testing:This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

Unit Testing Tools:

Here are some commonly used Unit Testing tools:

- 1.Jtest
- 2.Junit
- 3.NUnit
- 4.EMMA
- 5.PHPUnit

Advantages of Unit Testing:

- 1.Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
- 2.Unit testing allows the programmer to refine code and make sure the module works properly.
- 3.Unit testing enables testing parts of the project without waiting for others to be completed.

Disadvantages of Unit Testing: 1.The process is time-consuming for writing the unit test cases.

- 2.Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.
- 3.Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.
- 4.It requires more time for maintenance when the source code is changed frequently.
- 5.It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.

Program:

