**3. SYSTEM DESIGN**

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer’s goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement have been specified and analyzed, system design is the first of the three technical activities -design, code and test that is required to build and verify software.

The importance can be stated with a single word “Quality”. Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. Design is the only way that it can accurately translate a customer’s view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design I risk building an unstable system one that will be difficult to test, one whose quality cannot be assessed until the last stage.

During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities like architectural design, data structure design, interface design and procedural design.

**3.1 Database Design (E-R Diagram)**

Database Designing is a part of the development process. In the linear development cycle, it is used during the system requirements phase to construct the data components of the analysis model. This model represents the major data objects and the relationship between them. It should not be confused with data analysis, which takes place in the system design phase.

As in a DFD, a model of data consists of a number of symbols joined up according to certain conventions. System designers describe these conceptual modeling using symbols from a modeling method known as entity relationship analysis.

The data pertaining to proposed system is voluminous that careful design of the database must proceed storing the data in the database. A database management system provides flexibility in storing and retrieving of data and production of information. The RDBMS is a bridge between the application programs which determine what data is needed and how they are processed and the operating system of the computer, which is responsible for placing database that a specific program will use.

**Entity Relationship Diagram**

Entity relationship analysis uses three major abstractions to describe data. The Entities (which are distinct things in the enterprise), Relationships (which are meaningful interactions between the objects) and Attributes (which are the properties of the entities and relationship). The entity Relationship Diagram (ERD) depicts the relationship between the data objects. The ERD is the notation that is used to conduct the date modeling activity the attributes of each data object noted is the ERD can be described resign a data object description.

The primary purpose of the ERD is to represent data objects and their relationships. The relative simplicity and pictorial clarity of this diagramming technique may well account in large part for the widespread use of ER model. The relation upon the system is structure through a conceptual ER-Diagram, which not only specifics the existential entities but also the standard relations through which the system exists and the cardinalities that are necessary for the system state to continue.

The entity Relationship Diagram (ERD) depicts the relationship between the data objects. The ERD is the notation that is used to conduct the date modeling activity the attributes of each data object noted is the ERD can be described resign a data object description. The primary purpose of the ERD is to represent data objects and their relationships. The set of primary components that are identified by the ERD are

* Data object
* Relationship s
* Attributes
* Various types of indicators.

**ER Diagram Components**

Rectangle, it represents the entity set.

Ellipse, it represents attributes.

Diamond, it represents relationship sets.

Lines, which link attributes to entity sets and

entity sets to relationships.

Double Ellipse, it represents multivalve

attributes.

**Mapping Cardinalities**

It expresses the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, the mapping cardinality must be one of the following:

**One-to-One:** An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

**One-to-many:** An entity in A is associated with any number in B. An entity in B is associated with any number in A.

**Many-to-many:** Entities in A and B are associated with any number from each other.

**Cardinality:** It indicates that which type relationship the business rule follows is called cardinality.

**Connectivity:** It specifies that which type of notation the entities are connected in both sides that one side or many side.



**Fig 3.1.1: ER Diagram for Overall System**

3.2 DATA DICTIONARY

Data dictionaries are integral component of structured analysis, since data flow diagrams by themselves do not fully describe the subject of the investigation. The data dictionary provides additional information about the system. A data dictionary is a catalog – repository provides additional information about the system.

A data dictionary is a catalog – repository of the elements in a system. In a data dictionary we will find a list of all elements composing the data flowing through the system. The major elements are data flows, data stores and process. The data dictionary stores details and descriptions of those elements.

Why is a data dictionary important?

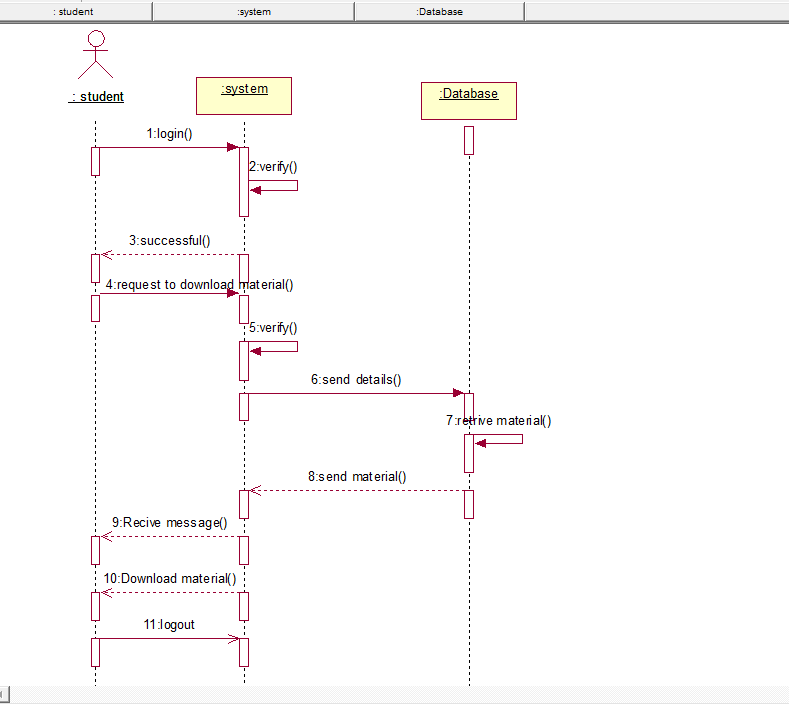
Analysis use data dictionaries for five important reasons:

* To manage the details of the large system
* To communicate a common meaning for all system elements
* To document the features of the system

# To facilitate analysis of the details in order to evaluate characteristics and determine where system changes should be made.

**Table Name: Abc**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | USERID | VARCHAR(20) | PRIMARY KEY | | PASSWORD | VARCHAR(20) | NOT NULL | | TYPEOFUSER | VARCHAR(10) | NOT NULL |   **Table No: 3.2.1**  **TABLE: COURSE**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | COURSEID | VARCHAR(20) | PRIMARY KEY | | COURSE | VARCHAR(30) |  | | DURATION | NUMBER(5) |  | | FEES | NUMBER(5) |  | | STATUS | VARCHAR(1) |  |   **TABLE: COURSEDETAILS**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | COURSEID | VARCHAR(20) |  | | DOL | DATE |  | | CHAPTER | VARCHAR(30) |  | | FILENAME | VARCHAR(100) |  |   **TABLE: COURSEHTML**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | COURSEID | VARCHAR(20) |  | | DOL | DATE |  | | CHAPTER | VARCHAR(30) |  | | TOPIC | VARCHAR(30) |  | | NOTES | VARCHAR(4000) |  |   **TABLE: ADMIN\_INBOX**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | MID | NUMBER | PRIMARY KEY | | USER\_NAME | VARCHAR2(30) |  | | MSG\_FROM | VARCHAR2(50) |  | | MSG\_SUBJECT | VARCHAR2(50) |  | | MSG\_DATA | VARCHAR2(1000) |  | | MSG\_DATE | DATE |  |   **TABLE: USER\_INBOX**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | MID | NUMBER | PRIMARY KEY | | USER\_NAME | VARCHAR2(30) |  | | MSG\_FROM | VARCHAR2(50) |  | | MSG\_SUBJECT | VARCHAR2(50) |  | | MSG\_DATA | VARCHAR2(500) |  | | MSG\_DATE | DATE |  |   **TABLE: MATCHQUES**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | QNO | NUMBER(10) | PRIMARY KEY | | COURSE | VARCHAR(20) |  | | QUESTION | VARCHAR(150) |  | | OPTION1 | VARCHAR(100) |  | | OPTION2 | VARCHAR(100) |  | | OPTION3 | VARCHAR(100) |  | | OPTION4 | VARCHAR(100) |  | | ANSWER | VARCHAR(100) |  | | IMAGE | VARCHAR(50) |  |   **TABLE: FILLQUES**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | QNO | NUMBER(10) | PRIMARY KEY | | COURSE | VARCHAR(20) |  | | QUESTION | VARCHAR(150) |  | | ANSWER | VARCHAR(100) |  | | IMAGE | VARCHAR(50) |  |   **TABLE: REGISTRATION**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | USERID | VARCHAR(20) | PRIMARY KEY | | PASSWORD | VARCHAR(20) |  | | FNAME | VARCHAR(20) |  | | LNAME | VARCHAR(20) |  | | DOB | DATE |  | | DOJ | DATE |  | | SEX | CHAR(1) |  | | EDUCATION | VARCHAR(20) |  | | OCCUPATION | VARCHAR(40) |  | | ADDR | VARCHAR(20) |  | | STATE | VARCHAR(20) |  | | CITY | VARCHAR(20) |  | | COUNTRY | VARCHAR(20) |  | | ZIP | VARCHAR(10) |  | | PHONE | NUMBER(8) |  | | TELEPHONE | NUMBER(10) |  | | EMAIL | VARCHAR(30) |  | | JOINUSER | NUMBER(20) |  | | CNO | NUMBER(10) |  | | STATUS | CHAR(1) |  |   **TABLE: COURSEJOIN**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | CNO | NUMBER(10) |  | | USERID | VARCHAR(20) |  | | COURSE | VARCHAR(20) |  | | FEES | NUMBER(10) |  | | STATUS | CHAR(1) |  |   **TABLE: DDPAYMENT**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | CNO | NUMBER(10) |  | | USERID | VARCHAR(20) |  | | DOD | DATE |  | | BANK | VARCHAR(20) |  | | DDNO | VARCHAR(20) |  | | AMT | NUMBER(10) |  | | STATUS | VARCHAR(1) |  |   **TABLE: EXAMPER**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | CNO | NUMBER(10) |  | | DOE | DATE |  | | COURSE | VARCHAR(20) |  | | STATUS | VARCHAR(1) |  | | RESULT | VARCHAR(10) |  | | MARKS | NUMBER(10) |  |   **TABLE: SETEXAM**   |  |  |  | | --- | --- | --- | | **COLUMN NAME** | **DATA TYPE & SIZE** | **CONSTRAINT** | | EXTYPE | NUMBER(1) |  |   **3.3 UML Design**  To understand the UML, you need to form a conceptual model of the language, and this requires learning three major elements the UML’s basic building blocks, the rules that dictate how these building blocks may be put together, and some common mechanisms that apply throughout the UML. Once you have grasped these ideas, you will be able to read UML models and create some basic ones. As you gain more experience in applying the UML, you can build on this conceptual model, using more advanced features of the language.  **Basic Building Blocks of the UML**  The vocabulary of the UML encompasses three kinds of building blocks  1. Things  2. Relationships  3. Diagrams  **Things in the UML**  Things are the abstractions that are first class citizens in a model, relationships tie these things together, and diagrams group interesting collections of things. These things are the basic object-oriented building blocks of the UML. You use them to write well-formed models. There are four kinds of things in the UML  1. Structural Things  2. Behavioral Things  3. Grouping Things  4. Annotational Things  **Structural Things**  Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.  **Classes**  First, a class is a description of a set of objects that share the same attributes, operations, relationships and semantics. A class implements one or more interfaces graphically, a class is rendered as a rectangle, usually including its, name, attributes, and operation.   |  | | --- | | Windows | | Origin  Size | | Open()  Close()  Move()  Display() |     **Interface**  Second, an interface is a collection of operations that specify a service of a class or component. An interface therefore describes the externally visible behavior of that element. An interface might represent the complete behavior of a class or component or only a part of that behavior. An interface defines a set of operation specifications. Graphically, an interface is rendered as a circle together with its name. An interface is rendered rarely stands alone. Rather, it is typically attached to the class or component that realizes the interface.  Description: C:\DOCUME~1\GAFFAR~1\LOCALS~1\Temp\ksohtml\wps_clip_image-27934.png  **Collaborations**  Third, collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that’s bigger than the sum of all the elements. Therefore, collaborations have structural, as well as behavioral, dimensions. It participate in several collaborations these collaborations therefore represent the implementation of patterns that make up a system. Graphically, collaboration is rendered as an ellipse with dashed lines, usually including only its name.  Description: C:\DOCUME~1\GAFFAR~1\LOCALS~1\Temp\ksohtml\wps_clip_image-21169.png  **Use Cases**  Fourth, a use case is a description of set of sequence of actions that a system performs that yields an observable result of value to a particular actor. A use case is used to structure the behavioral things in a model. A use case is realized by collaboration. Graphically use case is rendered as an ellipse with solid lines, usually including only its name.  Description: C:\DOCUME~1\GAFFAR~1\LOCALS~1\Temp\ksohtml\wps_clip_image-16096.png  The remaining three things active classes, components, and nodes are all class like meaning they also describe a set of objects that share the same attributes, operations, relationships, and semantics. However, these three are different enough and are necessary for modelling certain aspects of an object-oriented system, and so they warrant special treatment.  **Active Classes**  Fifth, an active class is a class whose objects own one or more processes or threads and therefore can initiate control activity. Its objects represent elements whose behavior is concurrent with other elements. Graphically, an active class is rendered just like a class, but with heavy lines, usually including its name, attributes, and operations. The remaining two elements component and nodes are also different. They represent physical things, whereas the previous five things represent conceptual or things.   |  | | --- | | Event Manager | |  | | Suspend()  Flush() |   **Components**  Sixth, a component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. In a system, you’ll encounter different kinds of deployment components, such as COM+ component or Java Beans, as well as components that are artifacts of the development process, such as source code files. A component typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations. Graphically a component is rendered as a rectangle with tabs, usually including only its name.  Description: C:\DOCUME~1\GAFFAR~1\LOCALS~1\Temp\ksohtml\wps_clip_image-31426.png  **Nodes**  Seventh, a node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability. A set of components may reside on a node and may reside on a node and may also migrate from node to node. Graphically, a node is rendered as a cube, usually including only its name.      These seven elements- classes, interfaces, collaborations, use cases; active classes, components, and nodes are the basic structural things that you may include in a UML model. There are also variations on these seven, such as actors, signals, and utilities (kinds of classes), processes and threads (kinds of active classes), and applications, documents, files, libraries, pages, and tables (kinds of components).  **Behavioral Things**  Behavioral things are the dynamic parts of UML models. These are the verbs of a model representing behavior over time and space. In all, there are two primary kinds of behavioral things.  First, an interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. The behavior of a society of objects or if an individual operation may be specified with an interaction. An interaction involves a number of other elements, including messages, action sequences (the behavior invoked by a message), and links (the connection between objects). Graphically message is rendered as a directed line, almost always including the name of its operation.  Description: C:\DOCUME~1\GAFFAR~1\LOCALS~1\Temp\ksohtml\wps_clip_image-32012.png    Second, a state machine is a behavior that specifies the sequence of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events the behavior of an individual class or a collaboration of classes may be specified with a state machine. A state machine involves a number of other elements, including states, transitions (the flow from state to state), events (things that trigger a transition), and activities (the response to a transition). Graphically, a state is rendered as a rounded rectangle, usually including its name and its sub states.  Description: C:\DOCUME~1\GAFFAR~1\LOCALS~1\Temp\ksohtml\wps_clip_image-20169.png  These two elements interactions and state machines are the basic behaviour things that you may include in a UML model. Semantically, these elements are usually connected to various structural elements, primarily classes, collaborations and objects.  **Grouping Things**  Grouping things are the original parts of UML models. These are the boxes into which a model can be decomposed. In all, there is one primary kind of grouping thing, namely, packages.  **Packages**  A package is a general-purpose mechanism for organizing elements into groups. Structural things, behavioral things, and even other grouping things may be placed in a package. Unlike components (which exist at run time), a package is purely conceptual (meaning that exist only development time). Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents.    Packages are the basic grouping things with which you may organize a UML model there are also variations, such as frameworks, models, and subsystems (kinds of packages).  **Annotational Things**  Annotational things are the explanatory1 parts of UML model. These are the comments you may apply to describe, illuminate, and remark about any element in a model.  Description: C:\DOCUME~1\GAFFAR~1\LOCALS~1\Temp\ksohtml\wps_clip_image-10985.png  There is one primary kind of annotational thing, called a note. A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements. Graphically note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment.  **Notes**  This element is the one basic annotational thing you may include in a UML model. You’ll typically use notes to adorn your diagrams with constraints or comments that are best expressed in informal or formal text. There are also variations on this element, such as requirements (which specify some desired behavior from the perspective of outside the model).  **Relationships in the UML**  These relationships are the basic relational building blocks of the UML. You use them to write well-formed models. There are four kinds of relationships in the UML.  1. Dependency  2. Association  3. Generalization  4. Realization  First, a dependency is a semantic relationship between two things in which a change to one thing the independent thing) may affect the semantics of the other thing (the dependent thing). Graphically dependency is rendered as a dashed line, possibly directed, and occasionally including a label.      Second, an association is a structural relationship that describes a set of links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts. Graphically, an association is rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and role names.  employer  employee  Third, a generalization is a specialization/generalization relationship in which objects of the specialized element are substitutable for objects of the generalized element (the parent). In this way, the child shares the structure and the behavior of the parent. Graphically a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent.    Fourth, a realization is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. You’ll encounter realization relationships in two places between interfaces and the classes or components that realize them and between use cases and the collaborations that realize them. Graphically, a realization relationship is rendered as a cross between a generalization and a dependency relationship.  These four elements are the basic relational things you may include in a UML model. There are also variations on these four, such as refinement, trace, include, and extended (for dependencies). UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram which is as follows.  **User Model View**  This view represents the system from the user’s perspective. The analysis representation describes a usage scenario from the end-users perspective.  S**tructural Model View**  In this model the data and functionality are arrived from inside the system. This model view models the static structures.  **Behavioral Model View**  It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.  **Implementation Model View**  In this structural and behavioral as parts of the system are represented as they are to be built.  **Environmental Model View**  In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.  **Diagrams in the UML**  A diagram is the graphical presentation of a set of elements most often rendered as a connected graph of vertices (things) and arcs (relationships). You draw diagrams to visualize a system from different perspectives, so a diagram represents an elided view of the elements that make up a system. The same element may appear in all diagrams, only a few diagrams, or in no diagrams at all. In theory, a diagram may contain any combination of things and relationships. In practice, however, a small number of common combinations arise, which are consistent with the five most useful views that comprise the architecture of a software intensive system. For this reason, the UML includes nine such diagrams. **Use Case Diagram** Use Case diagrams are one of the five diagrams in the UML for modelling the dynamic aspects of systems activity diagrams, sequence diagrams, state chart diagrams and collaboration diagrams are the four other kinds of diagrams in the UML for modelling the dynamic aspects of systems. Use Case diagrams are central to modelling the behavior of the system, a sub-system, or a class. Each one shows a set of use cases and actors and relationships. **Class Diagram** Class diagrams are the most common diagrams found in modelling object-oriented systems. A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Graphically, a class diagram is a collection of vertices and arcs. **Interaction Diagrams** An Interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. Interaction diagrams are used for modeling the dynamic aspects of the system.  **Activity Diagram**  An Activity Diagram is essentially a flow chart showing flow of control from activity to activity. They are used to model the dynamic aspects of as system. They can also be used to model the flow of an object as it moves from state to state at different points in the flow of control. State Chart Diagram A state chart diagram shows a state machine. State chart diagrams are used to model the dynamic aspects of the system. For the most part this involves modeling the behavior of the reactive objects. A reactive object is one whose behavior is best characterized by its response to events dispatched from outside its context. Component Diagram The component diagrams for the selected component package, along with new this list always contains an entry for the main component diagram for the component package.  **Deployment Diagram**  A deployment diagram shows processors, devices, and connections. Each model contains a single deployment diagram which shows the connections between its processors and devices, and the allocation of its processes to processors.  **Use Case Diagram**  Use-case diagram graphically depict system behavior. These diagrams present a high level view of how the system is used as viewed from an outsider’s (actor’s) perspective. A use-case diagram may depict all or some of the use cases of a system.  A use case diagram can contain:   * Actors * Use cases   Interaction or relationship between actor and use cases in the system including the associations, dependencies, and generalizations. Use-case diagram can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the systems implemented.    **Fig 3.3.1: Use case Diagram for Overall System**  **Class Diagram**  The UML class diagram also referred to as object modeling, is the main static analysis diagram. Object modeling is the process by which the logical objects in the problem space are represented by the actual objects in the program. These diagrams show a set of classes, interfaces, and collaborations and their relationships. These diagrams address the static design view of a system. This view primarily supports the functional requirements of a system – the services the system should provide to its end users. A class diagram is a collection of static modeling elements, such as classes and their relationships, connected as a graph to each other and to their contents.  Class diagrams commonly contain the following things:   * Classes. * Interfaces. * Collaborations. * Dependency, Generalization and association relationships.     **Fig 3.3.2: Class Diagram for Overall System**  **Sequence Diagram**  It is an interaction diagram that emphasizes the time ordering of messages. A sequence diagram shows objects participating in the interaction by their lifetime and the messages that they exchange/arranged in the time sequence.  **Description**  It contains the following elements:   * **Object:** It is represented as horizontal rectangle. * **Object Lifeline:** It represents the existence of an object at a particular instance of time and is represented as * **Focus of control:** It is a tall, thin rectangle that shows the period of time during which an object is performing an action. * **Messages:** It is communication between objects, shown as horizontal solid arrow from one object to another object.   In the below diagram  It is a type interaction diagram that shows the interaction between a set of objects. The way they are linked to each other and also the message.    **Fig 3.3.3: Sequence Diagram for select course** |  |  |



**Fig 3.3.4: Sequence Diagram for Download material**



**Fig 3.3.5: Sequence Diagram for set exam paper**



**Fig 3.3.6: Sequence Diagram for conducting exam**

**Activity Diagram**

An Activity diagrams illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special kind of state chart diagram, it uses some of the same modeling conventions.

**Basic Activity Diagram Symbols and Notations**

**Action states**

Action states represent the non-interruptible actions of objects. You can draw an action state in Smart Draw using a rectangle with rounded corners.

**Action Flow**

Action flow arrows illustrate the relationships among action states.

**Object Flow**

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object.

**Initial State**

A filled circle followed by an arrow represents the initial action state.

**Final State**

An arrow pointing to a filled circle nested inside another circle represents the final action state.

**Branching**

A diamond represents a decision with alternate paths. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else'.

**Synchronization**

A synchronization bar helps illustrates parallel transitions. Synchronization is also called forking and joining.

****

**Activity diagram for set exam paper**

****

**Fig 3.3.7: Activity Diagram for select course**

****

**Fig 3.3.8: Activity Diagram for Download material**

****

**Fig 3.3.10: Activity Diagram for conducting exam**

**Deployment Diagram**

Deployment diagrams describe the configuration of run-time processing resource elements and the mapping of software implementation components on to them. These Diagrams contain components and nodes, which represent processing or computational resources, including computers, printers, etc.

****