2. Project overview:

In our Plasma management system, a citizen can request or donate plasma. First the citizen must register into the system. Then he can log in to the system anytime with the password. To donate plasma a citizen must make a consultation with a doctor first. To take consultation, a citizen first has to get serial to book an appointment. A receptionist handles the appointments. If the receptionist provides him/her serial number only then he/she can consult with a doctor. A doctor runs checkup on the citizen who wants to donate plasma. The doctor gives the citizen clearance to donate plasma. A lab assistant takes plasma from the citizen. He also analyses the plasma and makes a report on it. Then he stores the plasma. A citizen can request for donation. They can ask for blood or plasma donation. Their request for blood is forwarded to a blood bank. Their request for plasma is forwarded to the system. The system checks for plasma availability and provides them plasma if available. The system also stores report, manages donor and also manages other applications. The receptionist provides the report to the citizen.

Module : Registration, System, Accounts, Lab

Use case: Use case diagrams are used to visualize, specify, construct, and document the (intended) behavior of the system, during requirements capture and analysis.

- Generalization : The child use case inherits the behavior and meaning of the parent use case.
- Actor : Actor specify something and system present the result to him. Exm: Faculty system e log in kore tarenvironment dekhe activity korte pare
- Include : ekta task er sathe arekta task jokhonfully dependent thake . ekta korle arekta kora lagbe. Log in - insert pass
- Extend : ekta task er jokhon onekgulo way te kora jay. Pay tution – online /cash
- Cross reference : onno kono use case er part hoye thakle er reference dite hobe
- Pre- condition : jemon boi sell korte hole age boi thaka lagbe .
- Post condition: Boi kinle ekta boi je common ekta update korte hobe

Class Diagram

- **The Class diagram represents** classes, their component parts, and the way in which classes of objects are related to one another.

The **Class diagram includes** attributes, operations, stereotypes, properties, associations, and inheritance.

Class vs. Object Diagram

- The class defines the *rules; the objects express the facts.*
- The class defines what *can be; the object describes what is.*
- If the Class diagram says, "This is the way things should be," but the Object diagram graphically demonstrates that "it just ain't so," then you have a very specific problem to track down. The reverse is true, too.
- The Object diagram can confirm that everything is working as it should.

- **Attributes** describe the appearance and knowledge of a class of objects.
- **Operations** define the behavior that a class of objects can manifest.
- **Stereotypes** help you understand this type of object in the context of other classes of objects with similar roles within the system's design.
- **Properties** provide a way to track the maintenance and status of the class definition.
- **Association** is just a formal term for a type of relationship that this type of object may participate in. Associations may come in many variations, including simple, aggregate and composite, qualified, and reflexive.
- **Inheritance** allows you to organize the class definitions to simplify and facilitate their implementation.
- **Aggregation** is a special type of association used to indicate that the participating objects are not just independent objects that know about each other. Instead, they are assembled or configured together to create a new, more complex object.
- For example, a number of different parts are assembled to create a car, a boat, or a plane.
- **Composition** is used for aggregations where the life span of the part depends on the life span of the aggregate.

Activity Diagram

We use **Activity Diagrams** to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. An activity diagram is a **behavioral diagram** i.e. it depicts the behavior of a system. An activity diagram portrays the **control flow from a start point to a finish point** showing the various decision paths that exist while the activity is being executed. An activity diagram is used to model the workflow depicting **conditions, constraints, sequential and concurrent activities**.

Activity Diagram Notations –

Initial State – The starting state before an activity takes place is depicted using the initial state.(only one)

Action or Activity State – An activity represents execution of an action on objects or by objects.(one application)

Action Flow or Control flows – Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.

Decision node and Branching – When we need to make a decision before deciding the flow of control, we use the decision node.

Guards – A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

Fork – Fork nodes are used to support concurrent activities

Join – Join nodes are used to support concurrent activities converging into one.

Merge or Merge Event – Scenarios arise when activities which are not being executed concurrently have to be merged.

Swimlanes – We use swimlanes for grouping related activities in one column. They usually give more clarity to the activity diagram

Final State or End State – The state which the system reaches when a particular process or activity ends is known as a Final State or End State

Sequence Diagram

A **sequence diagram** is the most commonly used **interaction** diagram.
Interaction diagram –
An interaction diagram is used to show the **interactive behavior** of a system. Communicate by passing massage

Sequence Diagrams –
A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

Sequence Diagram Notations

Actors – An actor in a UML diagram represents a type of role where it interacts with the system and its objects.

Lifelines – A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline.1) Persistent, 2) Temporary

Messages – Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows.

Synchronous messages – A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed the processing of the message.

Asynchronous Messages – An asynchronous message does not wait for a reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not.

Create message – We use a Create message to instantiate a new object in the sequence diagram.

Delete Message – We use a Delete Message to delete an object.

Self Message – Certain scenarios might arise where the object needs to send a message to itself.

Guards – To model conditions we use guards in UML. They are used when we need to restrict the flow of messages on the pretext of a condition being met.

An operation is **executing** when some process is running its code. An operation is **suspended** when it sends a synchronous message and is waiting for it to return. An operation is **active** when it is executing or suspended. The period when an object is active can be shown using an *execution occurrence*.

State chart Diagram

A **state diagram** is used to represent the condition of the system or part of the system at finite instances of time. It's a **behavioral** diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as **State machines** and **State-chart Diagrams**.

Uses of statechart diagram –

- We use it to state the events responsible for change in state (we do not show what processes cause those events).
- We use it to model the dynamic behavior of the system .
- To understand the reaction of objects/classes to internal or external stimuli.

Components:

Initial state – We use a black filled circle represent the initial state of a System or a class.

Event: The reason of Transition

Transition – The arrow is labelled with the event which causes the change in state.

State – We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.

Fork – We use the fork notation to represent a state splitting into two or more concurrent states.

Join – We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.

Self transition – There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self transitions to represent such cases.

Action: An action is associated with an event. An action is the behavior that is triggered by the event and it is the behavior that actually changes the attributes that define the state of the object

Component & Deployment Diagram

Module: Components are the parts of the system which has some specific task to complete. Component is independent and replaceable.

Module : Accounts , Library, Register, Marketing

Component: Part of a software that does the specific similar task of the software function. Fuction er object niye kaj kore. Exam: Registration,

Artifact: Physical element gulo different file e store kori. Jemon database er info file e save kori.

Interface: We declare the function (not details) and these are implemented by others. Direct interaction between two model is not possible. It is not designed due to security issue.

- **A provided interface**

Characterize services that the component offers to its environment. Is modeled using a ball, labelled with the name, attached by a solid line to the component

- **A required interface**

Characterize services that the component expects from its environment. Is modeled using a socket, labelled with the name, attached by a solid line to the component

Usage Dependencies: Ekta element er kono kaj implement korte gele arekta element er dorkar hoy.

Register-> tution fee

Connector : Duita component er moddhe connect kore. Onno ekta component. Jemon registration component er sathe account component ke connect kore.

Port: Specifies a distinct interaction point.Between that component and its environment. Between that component and its internal parts

External View : An external view (or black box view) shows publicly visible properties and operations

Internal View : An internal, or white box view of a component is where the realizing classes/components are nested within the component shape

Assembly Connector : A connector between 2 components defines that one component provides the services that another component requires. An assembly connector is notated by a "ball-and-socket" connection.

SEMANTICS: Are that signals travel along an instance of a connector originating in a required port and delivered to a provided port

DELEGATION: **Links the external contract of a component to the internal realization**

Deployment Diagram:

- **Contains nodes and connections**
- **A node usually represent a piece of hardware in the system**
- **A connection depicts the communication path used by the hardware to communicate**
- **Usually indicates the method such as TCP/IP**

- **An artifact :** Is the specification of a phisycal piece of information

  Ex: source files, binary executable files, table in a database system

Software Measuring :

Its is the evaluation criteria to test the quality of work regrading designing requirement analysis, implementation of the software coding or to know about the standard. Three entities : 1) **Process** 2) **Resource**, 3) **Product**

A **process** is any software-related activity such as change analysis, specification, design, coding and testing.

A **resource** is input to a process, for example personnel, hardware and software.

A **product** is any intermediate or final output resulting from a software process such as system documentation (for example, specification and design), program listings, test data, source code and object code.

An **internal attribute** is one which can be measured in terms of the process, product or resource itself. For example, complexity, modularity and reusability are internal attributes of the source code of a program.

An **external attribute** is one which can only be measured with respect to the relation of a process, product or resource to its environment, for example the maintainability of program source code or productivity of software personnel.

WMC : Sum of all methods complexity calculate individuals and sum up all.

CBC : koytar sathe interact kore

NOC : Child koyta. The number of children (NOC) metric value of a class *C* is the number of immediate subclasses of C

DIT : Parent er distance. The DIT of a class *C* in an inheritance hierarchy is the depth from the root class in the inheritance tree .It is the length of the shortest path from the root of the tree to the node representing *C* or the number of ancestors *C* has

LCOM : The Lack of Cohesion in Methods metric is **a measure for the number of not connected method pairs in a class representing independent parts having no cohesion**.

Cufflink between classes: duita class er sathe connected hoake

Assembly Connector:

A node: network environment এ important. memory থাকে, Processor থাকে. Physical element.

Behavioral things: Dynamics part. How our software will work. These are the verb.

Interaction: এক function/class আরেকটাকে call করে interaction করে message pass করবে। Display করায়

State-machine: প্রতিটা object এর বিভিন্ন state থাকে। Accounts থেকে টাকা তুললে, জমা করলে state change হয়।

Activity: একটা action. function/method কে activity বলে। এর মাধ্যমে বিভিন্ন action নেয়া হয়

Activity এর প্রতিটি step → action

A use case: is a requirement analysis. কি কাজ করবে system টা?

component: বড় software কে ছোট ছোট ভাগে ভাগ করি। Hospital এ Accounts, Lab, Pathology. component independent হয়। interface থাকে। এর: দুইটা component একে অপরের সাথে direct contact করতে পারে না। interface দিয়ে করে।

Artifact: physical element যেমন Different File এ store করি। যেমন Database এর info file এ save করি।

→ Annotation things : বিভিন্ন কমেন্ট দিয়ে যে clear করতে চাই।

Structural things:

* class: is a description of set of objects, attributes, relationship, semantics. object হবে যখন specific value দিয়ে দিবো।

* interface: is a collection of operation that specify a service of a class or component.

* collaboration: responsibilities of classes and interaction between them to complet the task

Grouping things: বিভিন্ন ~~component~~ ক্লাসের যে বিভিন্ন same ডিসিশন নিয়ে যে কাজগুলো component বানাই। এখান থেকেই গ্রুপিং হয়। বিজনেস এর সব ক্লাসগুলো একটা group এ রাখতে পারি।

Package: বিভিন্ন element কেউ চাইলে একসাথে প্যাকেজ করে রাখতে পারি। যেমন Business rules.