Software life cycle models

- The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes:

1. Requirement phase
2. Design Phase
3. Implementation Phase
4. Test Phase
5. Installation and check out Phase
6. Operation and Maintenance Phase

Build and Fix Model

- Product is built without any specification or any attempt at design

- Simple two phase model

- Suitable for small programming exercises of 100-200 lines
- Code soon becomes unfixable & unexchangeable.
- Maintenance is practically not possible

Waterfall Method

- This model is easy to understand and reinforces the notion of "define before design" and "design before code".

- The model expects complete & accurate requirements early in the process.

Problems of waterfall model

(i) It is difficult to define all the requirements at the beginning of a project.

(ii) This model is not suitable for accomodating any change.

(iii) A working version of the system is not seen until late in the project's life.

(iv) It does not scale up well to large projects.

(v) Real projects are rarely sequential

Iterative Enhancement Model

This model has the same phases as the waterfall model, but with fewer restrictions. Generally the phases occur in the same order as waterfall model but they may be conducted in several cycles.

Useable product is released at the end of each cycle, with each release providing additional functionality.

- Customers and developers specify as many requirements as possible & prepare a SRS document.
- Developers and customers then prioritize these requirements.
- Developers implement the specified requirements in one or more cycles of design, implementation and test based on the defined priorities.

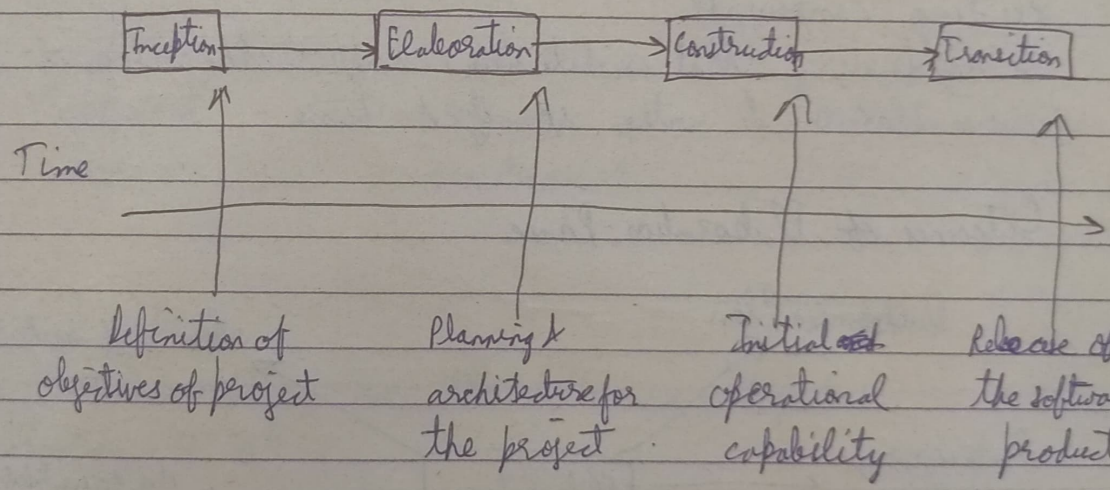
Rational Unified Process

- Software engineering process with the goal of producing good quality maintainable software within specified time and budget.

- Developed through a series of fixed length mini projects called iterations.

- Maintained & enhanced by Rational Software Corporation and thus referred as RUP.

Phases of the Unified Process



Inception : defines the scope of the project.

Elaboration : - How do we plan & design the project?

- What resources are required?

- What type of architecture may be suitable?

Construction : the objectives are translated in design & documents

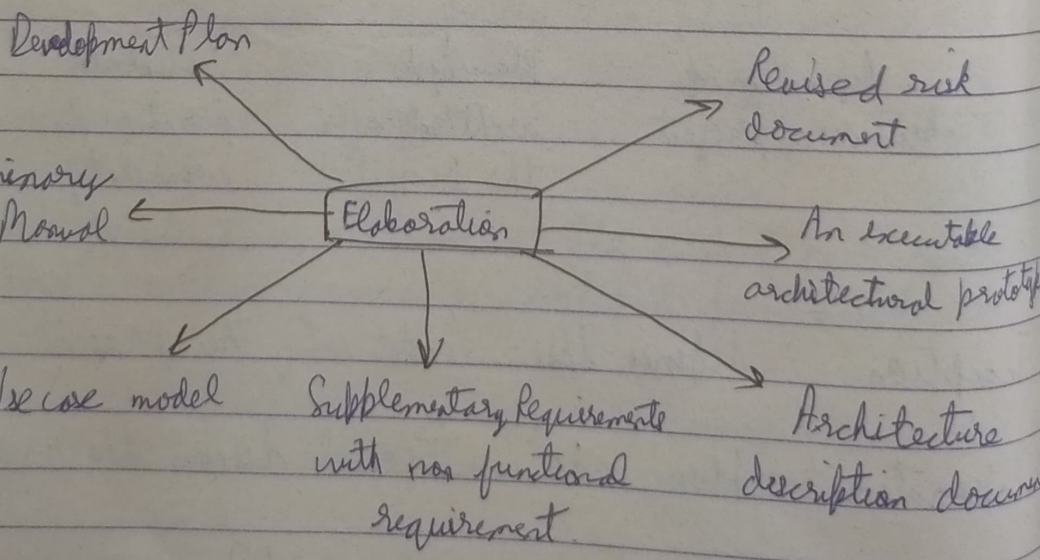
Transition : involves many activities like delivering, training supporting, and maintaining the product.

Elaboration Phase

The elaboration phase has the following objectives :

- Establishing architectural foundations
- Design of use case model
- Elaborating the process, infrastructure & development environment
- Selecting components
- Demonstrating that architecture support the vision at reasonable cost & within specified time.

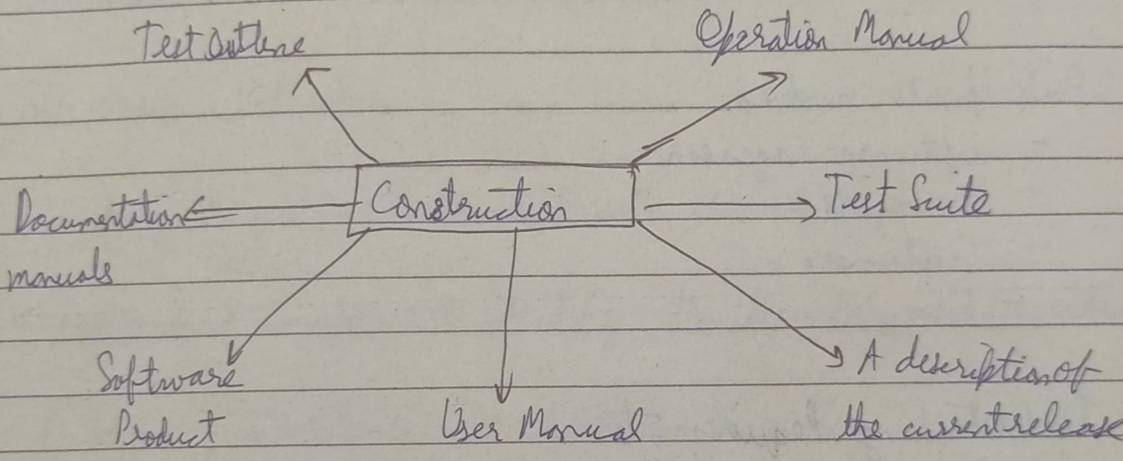
Outcomes of Elaboration Phase



Construction Phase

The construction phase has the following objectives:

- Implementing the project
- Minimizing development cost
- Management and optimising resources
- Testing the product
- Assessing the product releases against acceptance criteria

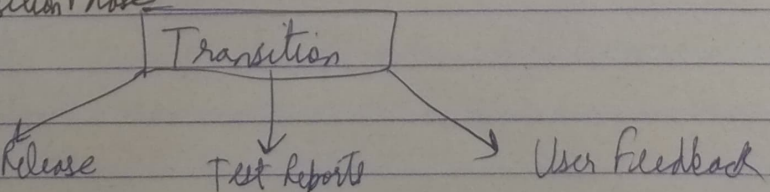


Transition Phase

The transition phase has the following objectives:

- Starting of testing
- Analysing of user's views
- Training of ~~users~~ users
- Tuning activities including bug fixing & enhancements for performance & reusability.
- Assessing the customer satisfaction.

Outcomes of Transition Phase:



Selection of a model is based on:
a) Requirements
b) Development team
c) Users
d) Project type and associated risk

For this check PPT of SDLC

Requirement Engineering Process
- Helps software engineer to better understand the problem

- Participants involved:
  - software engineers
  - Managers
  - Customers
  - Users

Understanding Requirements:
- Collecting needs from the customer
- Managing the process.
- Tasks involved:
  - Inception
  - Elicitation
  - Elaboration
  - Negotiation
  - Specification
  - Validation
  - Requirements Management

Inception (Beginning)

- During inception, the requirements asks a set of questions to establish :
  - Basic understanding of the problem
  - Nature of the solution that is desired

- Requirements engineers need to identify the stakeholders, recognize multiple viewpoints, work toward collaboration and initiate the communication.

Elicitation (Extraction)

Eliciting requirements is difficult because of

- Problems of scope → identify the boundaries of the system
- Problems of understanding → domain, computing environment
- Problems of Volatility → requirements may change overtime

Elicitation may be accomplished through two activities:
- → Collaborative Requirements Gathering
- → Quality Function Deployment

Elaboration (explanation)

- Takes the information obtained during inception & elicitation.
- Focuses on developing a refined model of software functions, features & constraints.
- This is analyzing phase
- It defines the functional, information and behavioral constraints of the problem domain.

Negotiation (Cooperation)

- Software engineer reconciles the conflicts b/w what the customer wants and what can be achieved.
- Requirements are ranked by the customer, users and other stakeholders.
- Risks associated with each requirement are identified

Specifications

- Final work product produced by the requirements engineer
- Form of SRS
- Serves as a foundation.
- It formalizes the functional and behavioral requirements of the proposal software in both ~~case~~ the graphical and textual format

Validation

- Specification is examined to ensure ~~A~~ that all the software requirements have been stated unambiguously.
- Errors have been detected and corrected.
- Members involved:
  - → Software Engineers
  - → Customers
  - → Users
  - → Other stakeholders

Requirements Management

- Project team performs a set of activities to identify, control and track requirements and changes to the requirements at any time as the project proceeds.

- Each requirements into one or more traceability tables.
- Tables may be stored in a database that relate features, sources, dependencies subsystems and interfaces to the requirements

Types of Requirements

- Customer Requirements
  → Define the expectations in terms of Mission, objectives, etc

- Functional Requirements
  → Explain what has to be done.
  → Identify the necessary action or activity and task.

- Non functional Requirements
  → Specify criteria that can be used to judge the operation of a system rather than behaviour.

- Performance Requirements
  → Examine the accuracy & speed with which a mission or function must be executed.
  → Measured in terms of quality, quantity, timeliness or readiness.

- Design Requirements:
  → Build to, code to, buy to
  → Use technical data packages and technical manuals.

- Derived Requirements:
  → Implied or transformed from higher level requirements.

Those who are involving in requirement analysis :
→ Requirement Engineer
→ System Analyst
→ System Engineer
→ Project leader

Requirement Engineering
- Feasibility Study
  → Find out the current user needs
  → Budget

- Requirement Analysis
  → What the stakeholders require from the system.

- ~~Requirem~~ Requirements Definition
  → Defines the requirements in a form of understandable to the customer.

- Requirement Specification
  → Defines the requirements in detail.

Requirement Document:
- Official Statement
- Include both a definition & specification
- Specify external system behavior
- Specify implementation constraints
- Easy to change.

Problems of Requirements Analysis:
- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Requirements change during the analysis process

Ground Work for Establishment:

- Ground work for requirement analysis consist of:

→ Identifying stakeholders
→ Recognizing viewpoints
→ Establishing collaboration among the stakeholders through conducting conversion and questionnaire among the stakeholders.

Stakeholders Identification:
- Stakeholder may be a project team member, employee of the user organization or a senior manager.

- Stakeholders analysis is a technique to identify and analysis the stakeholders project.

- Provides information on stakeholders and their relationships, interests and their expectations.

Requirement Elicitation:
- Discovering the requirements for the system.
- Identify the requirements by communicating with the customers, system users and other.

Requirements sources:
- Domain ~~sources~~ Knowledge
- Stakeholders
- Operational environment
- Organizational environment

Elicitation Techniques
- Interviews
- Scenarios
- Facilitated Meeting
- Prototypes
- Observation

CRC Cards
- CRC Card = Class Responsibility Collaborator Card

- a collection of ~~index~~ standard index cards that have been divided into three sections

  - a class represents a collection of similar objects
  - a responsibility is something that a class knows or does
  - a collaborator is another class that a class interacts with to fulfill its responsibilities.

- An index card consists of heavy paper cut to a standard size, used for recording and storing small amounts of discrete data.

Format of CRC cards:

Class Name:	
Responsibilities (what class does or knows)	Collaborators: (which classes help it perform each responsibility)

What is CRC card

- An effective way to analyze scenarios
- As a development tool that facilitates brainstorming & enhances communication among developers
- The cards are arranged to show the flow of messages among instances of each class.
- As team members walk through the scenario,
  - → they may assign new responsibilities to an existing class,
  - → group certain responsibilities to form a new class.
  - → divide responsibilities of one class into more fine grained ones.
- Responsibilities of a class (What information you wish to maintain about it)
  - Knowing responsibilities
    - instance of a class must be capable of knowing (the values of its attributes and its relationships)
    - Example: Student have names, addresses & phone no.

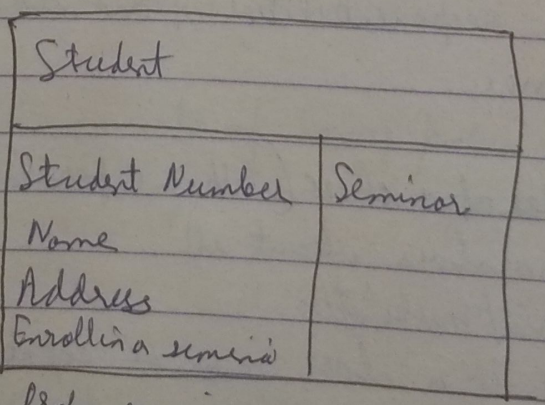
- Doing responsibilities
  - things that an instance of a class must be capable of doing
  - Example: students enroll in seminars, drop seminars

- A class is able to change the values of the things it knows, but unable to change

- Sets of of class forms a collaboration
  - Class does not have sufficient information to fulfill its responsibilities
  - A class must collaborate with other classes to get the job done

- Allows analyst to think in terms of an instance of a clients, servers and contracts.

→ Client objects - sends a request to an instance of another class for an operation to be executed.
→ Server objects - ~~receives~~ recieves the request from the client objects
→ Contracts objects - formalizes the interaction b/w the client and server objects.
eg → student CRC Card



Object Oriented Analysis & Design

Modeling - It is the describing a system at a high level of abstraction

UML stands for Unified Modeling language

Objectives of UML

- UML is a general purpose notation that is used to

→ visilize
→ ~~only~~ specify
→ construct
→ document
the artificate of a software system

- Use Case Diagrams — for functional models
- Class Diagrams } for structural models
- Object Diagrams
- Sequence Diagrams } for dynamic models
- Activity Diagrams
- State Diagrams

Development Process

- Requirements elicitation - High level capture of user/system requirements.
  - Use case Diagram
- Identify major objects and relationships
  - Object and class diagrams
- Create scenarios of usage
  - Class, sequence and collaboration diagrams
- Generalize scenarios to describe behaviour
  - Class, state and activity diagram
- Refine and add implementation details
  - Component and deployment diagrams

Structural Diagrams

- Class Diagram - set of classes and their relationships. Describes interface to the class (set of operations describing services).
- Object Diagram - set of objects (class instances) and their relationships
- Component Diagrams - logical grouping of elements and their relationships.
- Deployment Diagram - set of computational resources that host each component.

Behavioral Diagram

- Use case diagram - high level behaviours of the system, user goals, external entities ≈ actors
- Sequence diagram - focus on time ordering of messages
- Collaboration diagram - focus on structural organization of objects and messages
- State Diagram - event driven state changes of system
- Activity Diagram - flow of control between activities

Use Case Diagram

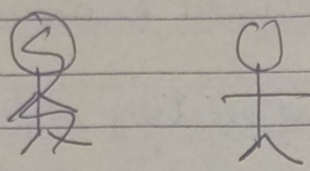
They are the descriptions of the functionality of a system from a user perspective

- Depicts behaviors of the system as it appears to an outside user
- Depicts the functionality and ~~was~~ users of the system
- Shows the relationship b/w the actors of the system ~~that~~, the use cases they use, and the ~~not~~ relation b/w different use cases

Components of use case diagram:-

- Actor
- Use case
- System Boundary
- Relationship

Actor is someone or something that interacts with the system
Actors are not part of the system



- Actors carry out use cases and a single actor may perform more than one use cases.
- Actors are determined by observing the direct use of the system.

Primary actors :-
- Acts on the system
- Initiates an interaction with the system
- Uses the system to fulfil his/her goal
- Events something we don't have control over.

Secondary actors :-
- Is acted on/ invoked / used by the system
- Helps the system to fulfills its goal
- Something the system uses to get its job done.

Use Case
- A use case is a pattern of behaviours, the system exhibits
- The purpose of use case is to define a piece of coherent behaviour without revealing the internal structure.
- The use cases are sequence of actions that the user takes on a system to get particular target.

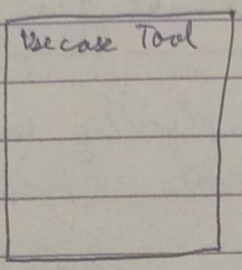
eg (Add course)

- Most of the use cases are generated in initial phase, but may add some more after proceeding.

System Boundary

It helps to identify what is external versus internal and what the responsibilities of the system are.

eg →

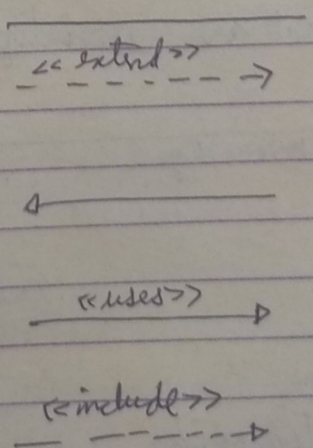


Relationship

- Relationship is an association b/w use case & actor.

Types of use case relationships :

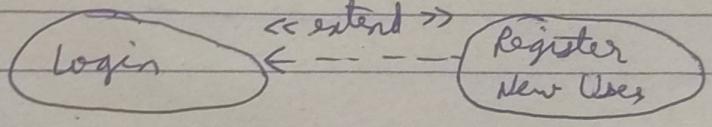
- Association
- Extend
- Generalization
- Uses
- Include



Extend Relationship

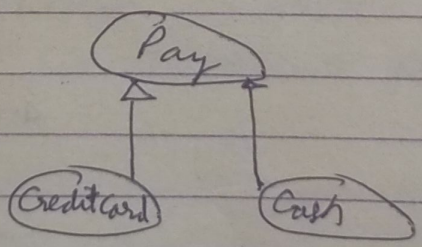
- The extended relationship is used to indicate that use case completely consists of the behavior of another use case at one or specific point.
- The insertion of additional behaviours into a base use case that does not know about it.
- The base use case implicitly incorporates the behaviours of another use case certain points called extension points.

eg →



Generalization

- Generalization is a relationship b/w a general use case and a more specific use case that inherits and extends feature to it.
- Use cases that are specialized versions of other uses cases.
- ~~It is shown~~ eg

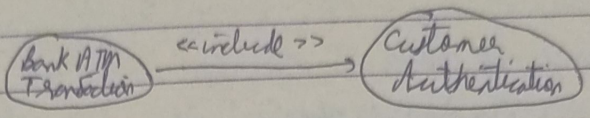


Include Relationship

The insertion of additional behaviour into a base use case that explicitly describes the insertion.

- The base use case explicitly incorporates the behaviour of another use case at a location specified in the base.

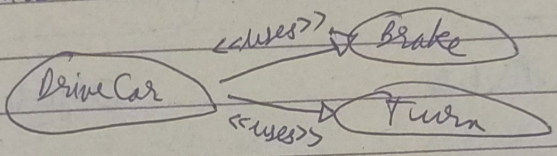
eg



Uses Relationship

- When a use case uses another process, the relationship can be shown with the uses relationship.
- This is shown as a solid line with hollow arrow point

eg →

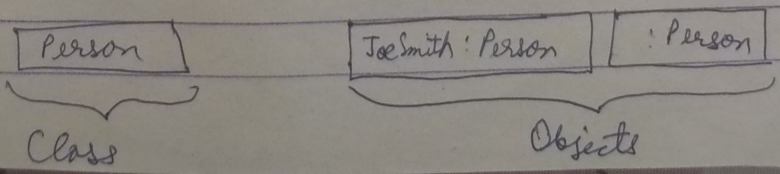


Class Diagram

- The purpose of a class modelling is to describe objects.
- An object is a concept, abstraction or thing that has meaning for a domain / application.
- Some objects have real world counterparts while others are conceptual ~~de~~ entities
- All objects have identity and are distinguishable.

- An object is an instance of a class
- A class describes a group of objects with some attributes, behavior, kinds of relationships and semantics.

Class / object representation

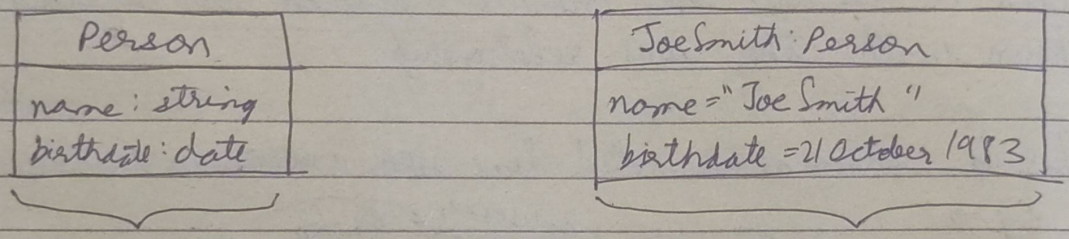


Values and attributes

- Value: piece of data
- Attribute: a named property of a class that describes a value held by each object of the class

~~- Attributes~~

eg →



Class with attributes

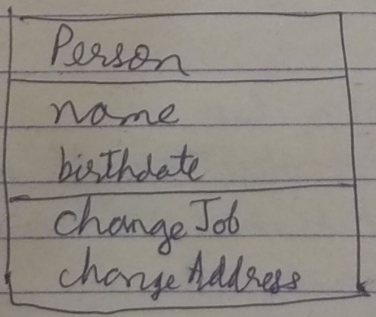
Objects with values

Object Identifiers

- Object Identifiers are implicit
- Objects belonging to the same and having the same attribute values may be different individual objects

- An operation is a function or procedure that may be applied to or by objects in a class.
- All objects in a class share the same operations.

eg →



Object Oriented Analysis

It is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises of interacting objects.

Object - It is a real world element in an object oriented environment that may have physical or a conceptual existence.

Class - A class represents a collection of objects having similar characteristic properties that exhibit common behaviour.

Encapsulation - It is the process of binding both attributes & methods together within a class. The internal details of a class can be hidden from outside.

Data Hiding - A class is designed such a way that its data can be accessed only by its class methods and isolated from direct outside access. This process is called data hiding.

Inheritance - When a new class is being created from an already existing class, then it is known as inheritance. The existing class is called base class. and the new class is called derived class.

Types of inheritance :-

1) Single Inheritance - When a new class is derived from a single base class.

2) Multiple Inheritance - When a has been derived from a multiple base class.

3) Multilevel – A subclass derives from a superclass which in turn is derived from another class.

4) Hierarchial – A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.

5) Hybrid – A combination of multiple and multilevel inheritance

~~Polymt~~

Polymorphism

Polymorphism implies using operations in different ways, depending upon the instance they are operating upon.

Abstraction

It denotes the essential characteristics of an object that distinguish it from all other kinds of object and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer

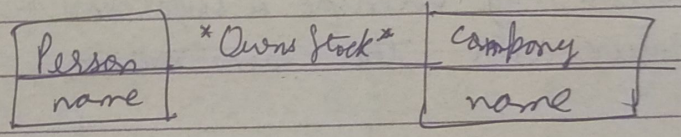
Modularity

It is the process of decomposing a problem into a set of modules so as to reduce the overall complexity of the problem.

Class Diagram (Continued)

- A link is a physical or conceptual connection among objects.
- Most links relate two objects, but some links relate three or more object.
- A link is an instance of an association.
- An association is a description of a group of links with common structure & semantics.
- Association is denoted by a line.

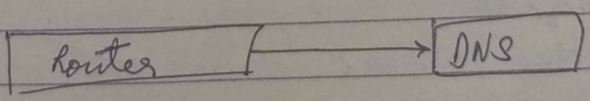
eg → Class Diagram



Unidirectional vs. Bi-directional Associations

- Associations are inherently bi-directional
- The association name is usually read in a particular direction but the binary may be traversed in either direction.

eg → Unidirectional

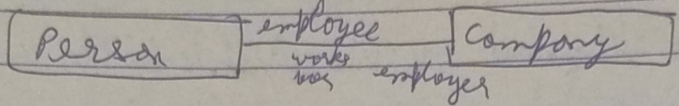


Multiplicity

It specifies the number of instances of one class that may relate to a single instance of the associated class.

- Associations have ends. They may have names

eg →



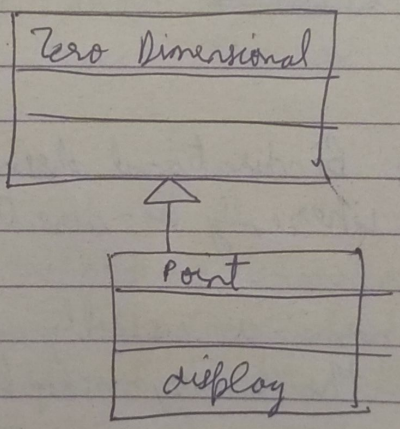
Generalization / Inheritance

- Generalization is the relationship b/w a class (superclass) and one or more variations of the class (subclasses)

- Generalization organizes classes by their similarity and their differences, structuring the ~~object~~ descriptions of objects.

- A subclass may override a superclass feature by redefining a feature with the same name.

eg →



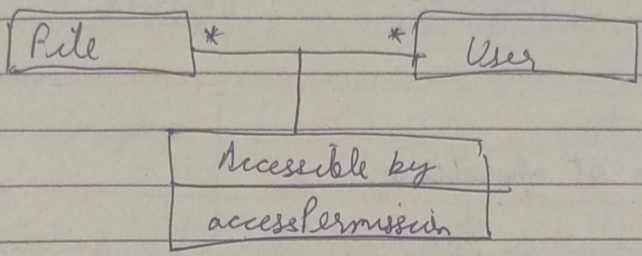
Use of generalization

- Used for three purposes:
  - Support of polymorphism
  - Structuring the description of objects
  - Enabling code reuse.

Association Class

- UML offers the ability to describe links of association with attributes like any class.

eg →



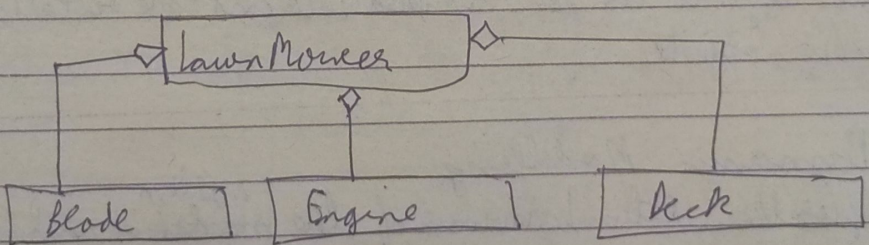
Aggregation

- Aggregation is a strong form of association in which an aggregate object is made of constituent parts.

- Aggregation is a transitive relation:
  - if A is a part of B and B is a part of C then A is also a part of C.

- Aggregation is antisymmetric
  if A is a part of B, then B is not a part of A

eg →



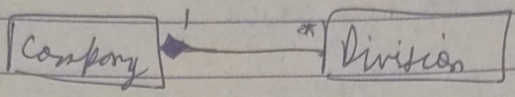
Composition

It is a form of aggregation with additional constraints:

- A constituent part can belong to atmost one assembly
- it has coincident lifetime with the assembly.
- Deletion of assembly object triggers automatically a deletion of all constituents object via composition.

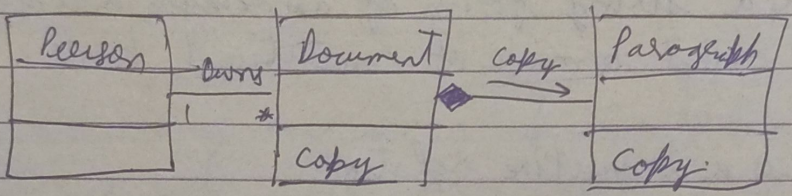
- ~~Composition implies~~

eg →



Propagation of operations

- Propagation is the automatic application of an operation to a network of objects when the operation is applied to some starting object.



Object Modeling

It develops the static structure of a system in term of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships b/w objects.

Dynamic Modelling

After the static behaviour of the ~~model~~ system is analysed, its behaviour with respect to time and external changes needs to be examined.

Functional Modelling

Functional model shows the processes that are performed with an object and how the data changes as it moves b/w methods.

Advantages & Disadvantages of Object Oriented Analysis

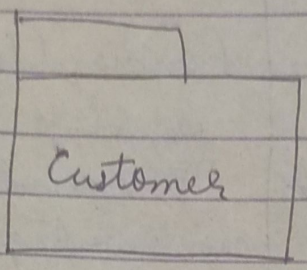
Advantages	Disadvantages
Focuses on data rather than ~~proc~~ procedure	Functionality is restricted with objects
Encapsulation & data hiding helps in data privacy	It cannot identify which objects would generate an optimal system design
It allows effective management of software complexity by the virtue of modularity	The object oriented models do not easily show the comm. b/w the objects in the system

Advantages & Disadvantages of Structured Analysis

Advantages	Disadvantages
It follows top down approach in contrast to bottom up approach of object oriented analysis	In traditional structured analysis models, one phase should be completed before the next phase. This pose a problem in design.
It is based upon functionality This gives a better understanding of the system as well as generates more complete systems.	The initial cost of constructing the system is high.
	It does not support reusability of code

Package:
A package is an organized group of elements. A package may contain structural things like classes, components and other packages in it.

eg →



Object Diagram

Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams. Object diagrams represent an instance of a class diagram

Where to use object diagrams?
Object diagrams can be imagined as the snapshot of running system at a particular moment.

It is used for:
- Making the prototype of a system
- Reverse Engineering
- Modeling complex data structures
- Understanding the system from practical perspective

Activity Diagram

Activity
An activity is a method of a class in a specification perspective diagram.

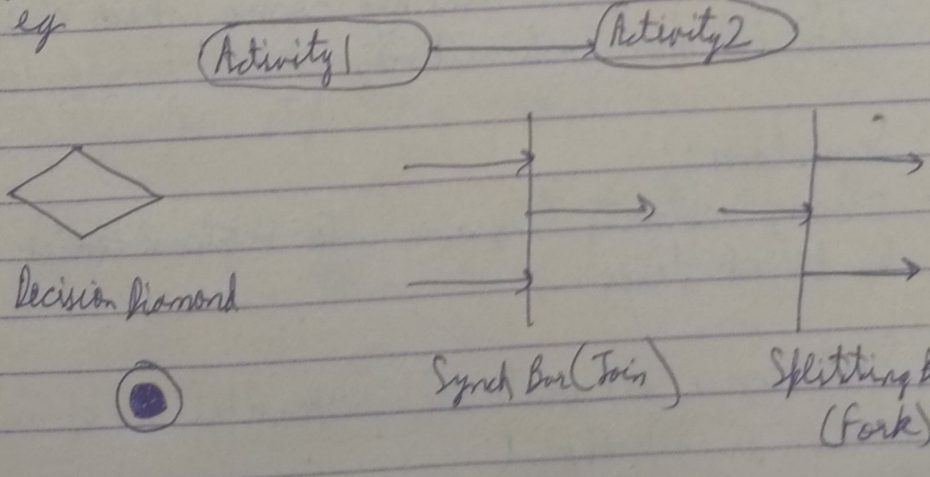
: Activity arrangement
- Sequential - one activity is followed by another.
- Parallel - two or more sets of activities are performed concurrently, and order is irrelevant.

- Activity diagram records the dependencies b/w activities such as which things can happen in parallel and what must be be finished before something else can start.

Activity Diagram

- Activity diagrams are useful for describing complicated methods
- Activity diagrams are useful for describing use cases, since after all, a use case is an interaction, which is a form of activity
- Activity Diagrams are like Flow Charts, but Flow Charts are usually limited to sequential activities while Activity Diagrams can show parallel activities as well.

eg



Subactivity

- A subactivity starts another activity graph without using an operation.
- Used for functional decomposition, non-polymorphic applications, like many workflow systems.

~~eg →~~

Synchronization bar can break a trigger into multiple triggers operating in parallel or can join multiple triggers into one ~~wait~~ when all are complete.

Decision Diamond - used to describe nested decisions

eg → Check lab 5 assignment in laptop

Drawbacks

- Activity Diagrams tell you what is happening, but not who does what

- In domain modeling, this diagram type does not convey which people or departments are responsible for each activity.

- In programming, it does not convey which class is responsible for each activity.

Swimlanes

- Arrange activity diagrams into vertical zones separated by dashed lines.

- Each zones represents the responsibilities of a particular class or department.

When to use activity diagrams

- Do use them for
  - ~~Anything use~~
  - Analysing use cases
  - Understanding workflow across many use cases.
  - Dealing with multi thread applications.

- Dont use them
  - to see how objects collaborate
  - to see how an object behaves over its lifetime

  eg - check assignment 2 of theory class

Interaction ~~Diagrams~~ Model

- The class model that describes the objects in a system and their relationships

- The state model describes the lifecycles of the objects

- At high level, use cases describes how a system interacts with outside actors.

- Activity diagrams provide details and & show the flow of control among the steps of a computation.

- Sequence diagrams provide more details and show the messages exchanged among ~~the~~ a set of objects overtime.

Scenario - An sequence of events that occurs during one particular execution of a system.

- Scope of scenario can vary
  - May include all events in the system.
  - May include only those events generated by certain objects.

- These express interaction at high level
- A scenario contains messages b/w objects as well as activities performed by objects.

- Each message transmits information from one object to another.

Sequence Diagram

- Scenario is convenient for writing but does not clearly show the sender and the receiver of each messages

- Sequence Diagram
  - Shows the participants in an interaction and the sequence of messages among them.

  - Shows the interaction of a system with its actors to perform all or part of a use case.

Key Parts of a Sequence Diagram

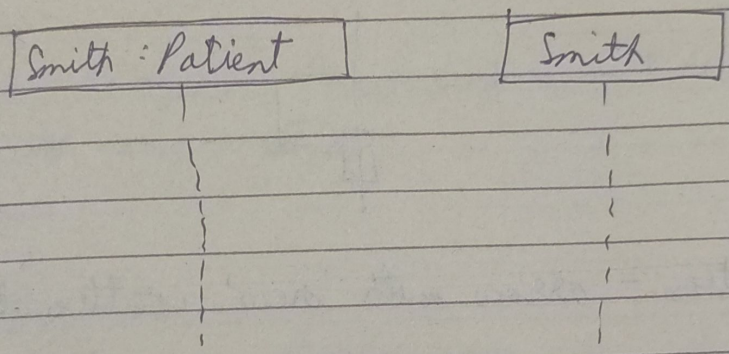
- participant: an object or entity that acts in the sequence diagram.
- message: Communication b/w participant objects

The axes in a sequence ~~object~~ diagram:
- horizontal: which object / participant is acting.
- vertical: lifeline

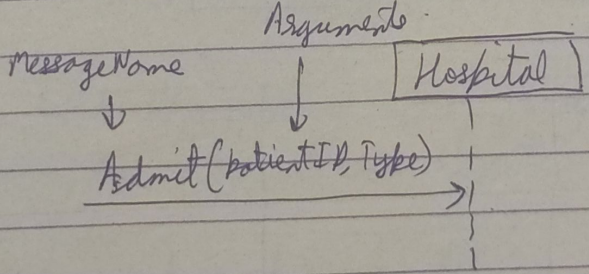
Representing objects -:
- Squares with object type, optionally proceeded by object name and colon

Name syntax: <objectname>: <classnam>

eg →

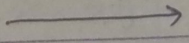


Message is indicated by horizontal arrow to other objects.

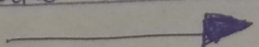


Dashed arrow back indicates return

Asynchronous Message - Asynchronous messages dont need a reply for interaction to continue



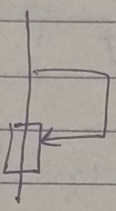
Synchronous Message - A synchronous message requires a response before the interaction can continue.



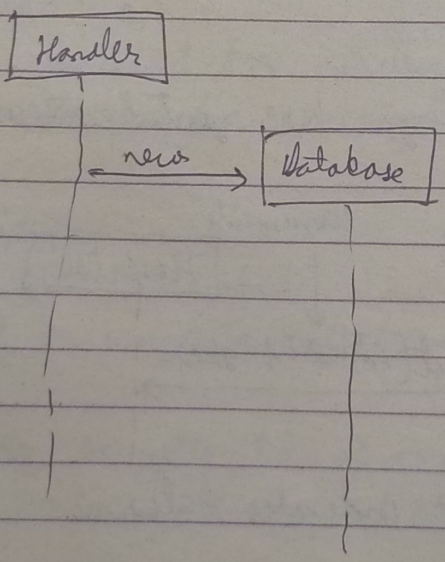
Reply or Return Message - A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

<---------- Reply or Return message.

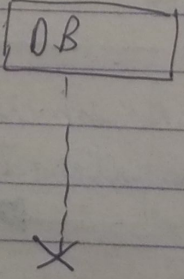
Self Message - A message an object sends to itself, usually shown as U shaped arrow pointing back to itself.



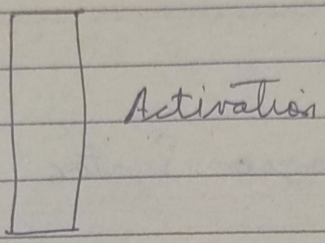
Creation - arrow with 'new' written above it



Deletion - an X ~~obje~~ at bottom of object lifeline



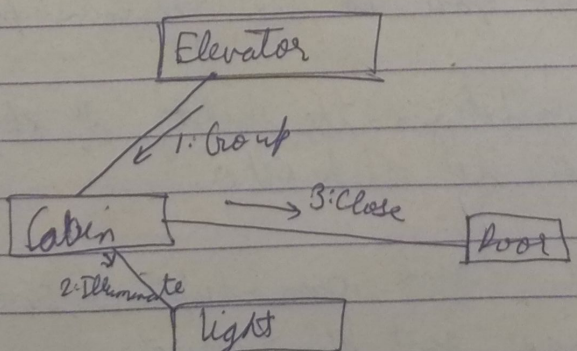
Activation Box - Activation box represents the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a grey ~~rectangular~~ rectangle placed vertically on its lifeline



Eg of Sequence Diagram - Check experiment No 6 lab ~~ass~~ assignment.

Collaboration Diagram

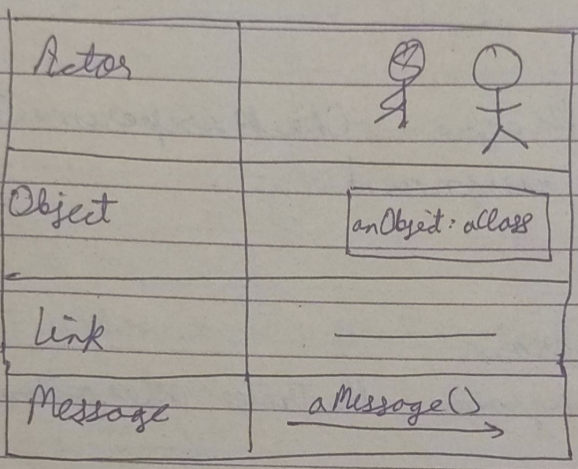
- Collaboration diagrams illustrates interaction b/w objects.
- It illustrates messages being sent b/w classes & objects.
- It express both the context of a group of objects and the interaction b/w these objects.
- These are very useful for visualising the relationship b/w objects collaborating to perform a specific task.



- There are three elements of a collaboration diagram:

→ objects
→ links
→ messages.

Collaboration Diagram Syntax



Object - ~~object~~ object name : object class

Links

- The connecting lines drawn b/w objects are links
- They enable you to see relationships b/w objects.
- This symbolizes the ability of objects to send messages to each other
- A single link can support one or more messages sent b/w objects.

- The visual representation of a link is straight line.

If the objects sends a message to itself, the link carrying these messages is represented as a loop icon.

Messages
- An interaction is initiated by a group of objects that collaborate by exchanging messages.
- Messages in collaboration diagrams are shown as arrows.
- Message icons have one or more messages associated with them.
- Messages are composed of messages text prefixed by a sequence number.
- The numbers indicate the sending order.

Message Flow Notation

→ Synchronous

→ Flow of control

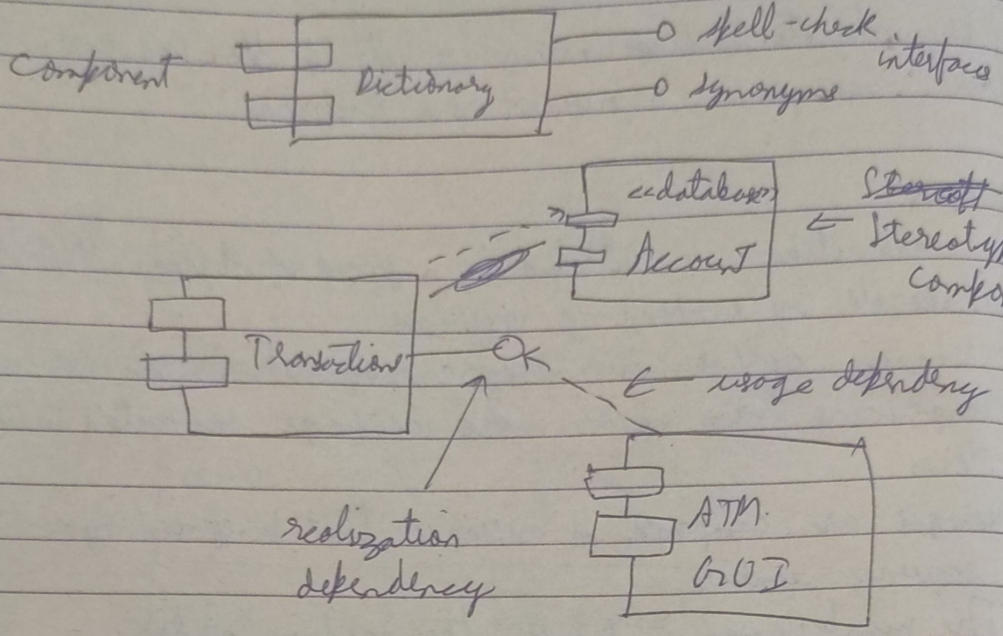
→ Asynchronous

- - - → Return

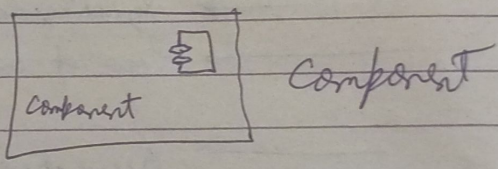
- Sequence diagram is better at time ordering.
- Collaboration diagram is better at showing the relationship b/w objects.
- Links in a collaboration diagram directly correlate to associations b/w classes in a class diagram.

Eg of collaboration diagram → exp 7 in lab assignments

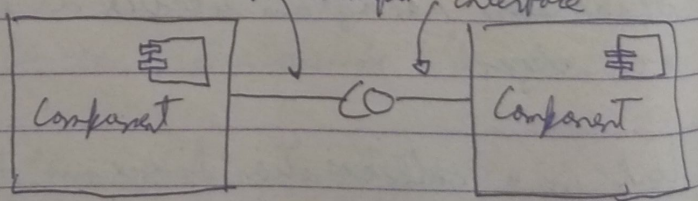
Eg of Component Diagram



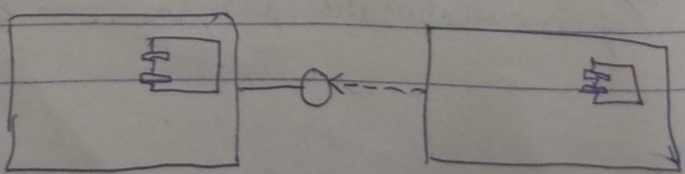
Component - A component is a logical unit block of the system, a slightly higher abstraction than classes



Interface - An interface describes a group of operations used or created by components. A full circle represents an interface created or provided by the circle
A semi circle represents a required interface, like a person's input



Dependencies - Draw dependencies among components using dashed arrows

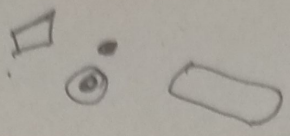


Physical views

- a system model is inteded to show the logical and design aspects of the system independent of its final packaging in an implementation medium.
- The implementation aspects are also important for both reusability and performance purposes.
- UML includes two kinds of views for representing implementation units
  - the implementation view
  - the deployment view

Implementation view

- shows the physical packaging of the reusable pieces of the system into substitutable units called components
- A component is a physical unit of implementation with well defined interfaces that is inteded to be used as a replacable part of a system.
- Each component embodies the implementation of certain classes from the system design.
- a component in a system can be replaced by another component that supports the proper interfaces.
- The component view shows the network of dependencies among components.



Deployment view

- shows the physical arrangement of runtime computational resources, such as computers and their interconnection called nodes.

- A node is a runtime physical object that represents a computational resource, generally having at least a memory and often processing capability.

- A node ~~represents~~ is a physical entity that executes one or more components, subsystems or executables.

- Artificats are concrete elements that are caused by a development process.

~~State Chart Diagram~~

State Modelling Diagram

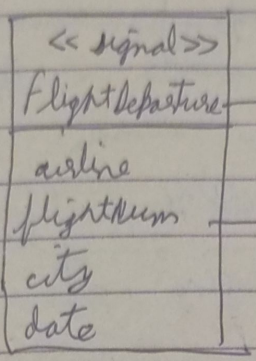
- The state model examine changes to the objects and their relationships over time.

- The state models describes the sequence of operations that occur in response to events.

Cocurrent event: usually unrelated events; have no effect on one another

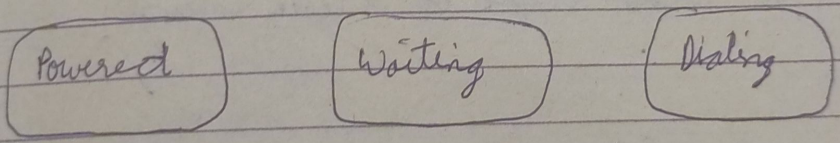
Signal Event - the event of sending or receiving of a signal

- sending of a signal by one object is a distinct event from its reception by another.



States

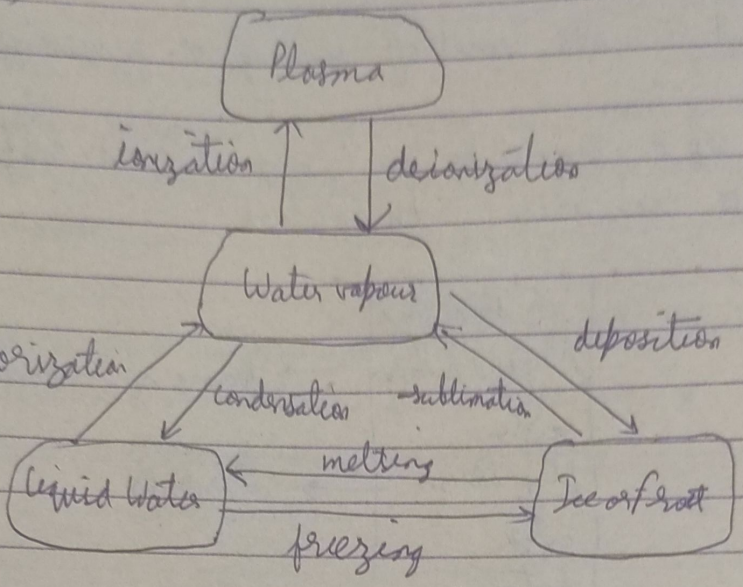
- an abstraction of values & links of an object
- UML Notation : a rounded box containing an optional state name.



- Objects in a class have a finite number of possible states
  - Each object can only be in one state at a time.

- A state specifies the response of ~~that~~ an object to input events.

eg →



Change event: event caused by satisfaction of boolean expression

Time event: event caused by occurence of an absolute time or the ~~elapse~~ elapse of a time interval

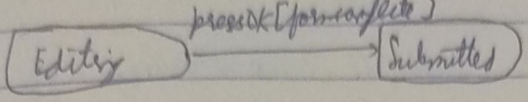
Transition - an instantaneous change in state.

Guard Condition - boolean expression that must be true for transition to occur.

- checked only once, at the time event occurs; transition fires if true.

Transition is:
- enabled when source state is active and guard condition is satisfied
- fires when enabled and triggering event occurs.

eg →



State Diagrams
- a graph whose nodes are states and whose directed arcs are transition b/w states.

- Specifies state sequences caused by event sequences.

- all objects in a class execute the state diagram for that class.

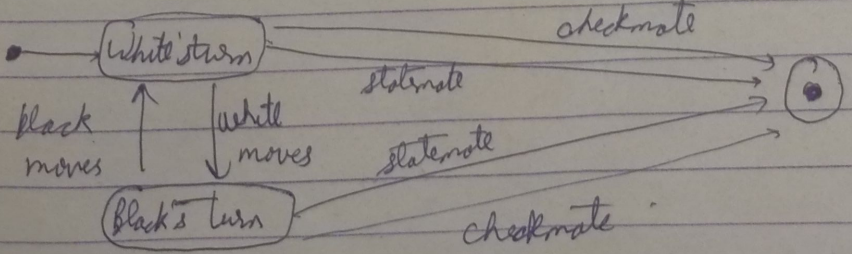
State Model
- multiple state diagrams, one for each class with important temporal behaviour.

- State diagrams can represent

  - Continuous loops
  - One-shot life cycle

One shot state diagram
- represents objects with finite lives
  - have initial & final states

- initial state - entered on object creation
- final state - entry implies destruction of object



Concurrency -:

- State model supports concurrency among objects
- Object can change & act state independent of one another.

eg of state diagram -: check experiment 8 in lab folder