- When a computer succeeds / fails : to succeed, Software is designed and built.

- The dual role of S/W : A product and at the same time, the vehicle for delivering a product:

i) As a product it delivers the computing potential embodied by computer hardware - information transformer - producing, managing, acquiring, modifying, displaying or transmitting information that can be simple, as a single bit or as complex as a multimedia presentation.

ii) A vehicle for delivering a product : software acts as the basis for control of the computer, the communication of information, and the creation and control of other programs.

- Ubiquitous Computing - More information appliances that have broadband connectivity to the Web to provide a blanket of connectedness over all homes, offices and motorways.

Software Definition

Software is:

i) Instructions (computer programs) that when executed provide desired features, functions, and performance

ii) Data structures that enable the programs to adequately manipulate information.

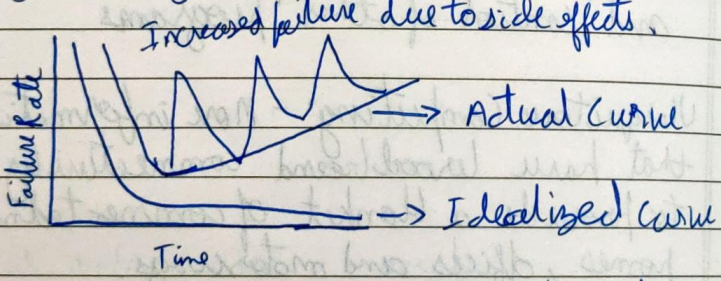
iii) Documentation that describes operation and use of the programs.

Nature of Software

1) Software is developed or engineered it is not manufactured in the classical sense

2) Software doesn't wear out but it does deteriorate- Software not susceptible to the environment maladies

Software Failure Curve - During its life, software will undergo change - No spare part in software.



3) Although the industry is moving toward component-based construction, most software continues to be custom built.

Software Application Domains

1) Information content: meaning and form of incoming and outgoing information - structured, experts.

2) Determinacy - predictability of the order and timing of information.
   i) Determinate systems without interruption
   ii) Indeterminate systems - output that varies as a function of environment and time - multi user, OS

3) System Software - Serve other programs determinate and indeterminate.
   Determinate - compilers, editors and file management utilities
   Indeterminate - Operating systems components, drivers telecommunication

4) Real-time - Monitors / analyzes / controls - real world events. such as user traffic control.

5) Business Software : Discrete "systems" evolved into MIS - applications restructure existing data in a way that facilitates business operations.

6) Embedded software : Intelligent products - embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets. Perform very limited and esoteric functions or provide significant function and control capability.

7) Product-line software: Designed to provide a specific capability for use by many different customers. Product line software can focus on a limited and esoteric marketplace or address mass consumer markets

8) Web Based Software - web pages.

9) AI Software - makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Expert systems, artificial neural networks etc.

Legacy Software - Older or much older - longevity and bussiness criticality, but poor quality.

- Reasons Software needs to change -
  i) Adaptation to meet the needs of new computing environments or technology
  ii) Enhanced to meet new business requirements
  iii) Extended to make it interoperable with other more modern systems or databases
  iv) Rearchitected to make it viable within a network environment.

SE Layers

Tools
↓
Methods
↓
Process
↓
A Quality focus

Process Layer - Defines a framework that must be established for effective delivery of software engineering ~~tools~~ technology - basis for project management and control, establishes the context in which technical methods are applied, work products produced, milestones are established, quality is assured and change is properly managed.

Methods - Provide technical how-to's for building software, encompass a broad array of tasks that include communication, requirement analysis, design modelling, program construction, testing and support; methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Tools - Provide automated or semi automated support for the process and the methods. When tools are integrated, a system for the support of software development called computer aided software engineering.

Types of software myths -

i) Management myths - Managers use myths to reduce Pressure
ii) Customer myths - False expectations of developers
iii) Practitioner myths - Old ways and attitudes die hard
    ↳ Eg: Once we write the program and get it to work, our job is done.

Process Framework

A generic framework includes following activities - Communication, Planning, Modeling, Construction, Deployment.

- Communication with the stakeholders to understand their objective for the project and to gather requirements that help define software features and functions

- Planning - helps guide team as it makes the journey. The software project plan - defines the SE work by describing the technical tasks to be conducted, risks likely, required resources, work products to be produced and a work schedule.

- Modeling - creating models to better analyze requirements and the design that will achieve those requirements.

- Construction - This activity combines code generation and that is required to uncover errors in the code.

Deployment - The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

Umbrella Activities included

1) Software project tracking and control - allows team to assess ~~Progress~~ progress against project plans and take any necessary action to maintain the schedule.

2) Risk Management - assesses ~~the~~ risks that may affect the outcome of the project or the quality of the product.

3) Quality Assurance - Defines and conducts the activities required ~~by~~ to ensure software quality

4) Technical reviews - Assesses software engineering work ~~products~~ products in an effort to uncover and remove errors before they are propogated to the next activity.

5) Reusability Management - Defines criteria for work product reuse and establishes mechanisms to achieve reusable components.

6) Work product preparation and production - encompasses the activities required to create work products such as models, documents, logs, forms and lists.

Prescriptive process models - Rigid

- They stress detailed definition, identification, and application of process activities and tasks.
- Good, but often not achieved objectives: quality, make projects manageable, timely delivery, reduce costs, guide teams.
- Prescriptive models if ~~not~~ not applied with adaptation, level of bureaucracy increases → difficulty for all stakeholders.

Agile process models - informal

- Agile process models emphasize project agility and are informal.
- Agile because they emphasize maneuverability and adaptability
- Appropriate for many types of projects, particularly useful for web applications
- Process Models: Prescriptive with adaptability and agility

Process Models

- Phases in Software Development:
  Requirements → Design → Development → Testing → Maintainance

- Prescriptive Process Models:

  - Prescribes set of process elements - Framework activities, SE actions, tasks, work products, quality assurance and change control mechanisms

  - Each process model also prescribes a workflow.

The Waterfall Model

- Also called classic life cycle.
- The oldest paradigm

Works best when-

i) Requirements of a problem are reasonably well understood

ii) Well-defined adaptations or enhancements to an existing system must be made

iii) Requirements are well defined and reasonably stable.

- Real projects rarely follow sequential flow.

- It is often difficult for the customer to state all requirements explicitly.

- The customer must have patience - A working version of the program will not be available until late in the project time-span.
- Leads to blocking state for team members.

Incremental Process Models

- Combines elements of waterfall model applied in an iterative fashion.
- Each linear sequence produces deliverable increments of the software
- The first increment is often a core product.
- The core product is used
- Based on evaluations the a plan is developed for the next increment.
- Unlike prototyping a incremental model focuses on the delivery of an operational product with each increment.

★ Particularly useful when staffing is unavailable

- Increments can be planned to manage technical risks.

The RAD Model

- Rapid Application Development is Incremental Process Model.
- Emphasizes on short development cycle.
- A "high speed" adaptation of the waterfall model.
- Uses a component-based construction approach.
- May deliver the software within a very short time period if requirements are well understood and project scope is constrained
- The applications should be modularized by and addressed by separate RAD teams.
- Integration is required.

Drawbacks:

- It requires sufficient human resources for large projects.
- RAD projects will fail if developers and customers are not committed to the ~~so~~ rapid-fire activities.
- If a system cannot be properly modularized, building the components necessary for RAD will be problematic.

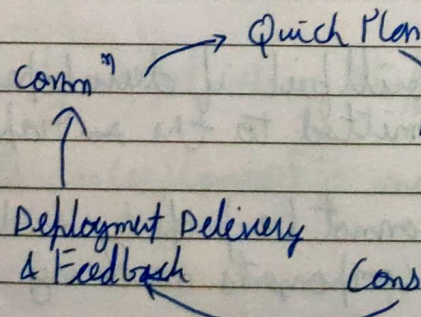
- RAD may not be appropriate when technical risks are high.

Evolutionary Process Models

- Software like all complex systems evolves over a period of time
- Business and product requirements often change as development proceeds, making a straight line path to an end product is unrealistic
- Evolutionary models are iterative.

Prototyping

- Customers defines general objectives but not sure with detailed input, processing and output.
- Stand-alone or a complimentary model-assisting the software engineer and the customers to better understand what to be built when requirements are fuzzy.



Prototyping problems:

1) Customers may press for immediate delivery of working but inefficient products.
2) The developer often makes implementation compromises to get prototype working properly.

Defects

- Can be injected at any of the major phases
- Cost of latency: Cost of defect removal increases exponentially with latency time.
- Cheapest way to detect and remove defects close to where it was injected
- Hence must check for defects after every phase.

Factors to be considered for Software Dev:

1) User Satisfaction
2) Time
3) Quality Factors
4) Adaptable for changes
5) Risk Management
6) Evolution.

The Spiral Model

- Couples the iterative nature of prototyping with the controlled and systematic aspects of waterfall model.
- It provides for a rapid development of increasingly more complete versions of the software.
- It is a risk-driven process model generator.
- Two main distinguishing features -
  i) Cyclic Approach - Incrementally increasing a system's degree of definition and implementation while decreasing its degree of risk.
  ii) A set of anchor point milestones - For ensuring stakeholders commitment to feasible and mutually satisfactory system solution.
- Unlike models that end when software is delivered, Spiral model can be adapted to apply throughout the life of computer s/w.
- It demands direct consideration of technical risks at all stages of the project

Drawbacks

- It may be difficult to convince customers that the evolutionary approach is controllable
- It demands considerable risk assessment expertise and relies on this expertise for success.
- If a major risk is not uncovered and managed, problems will undoubtedly occur.

Weakness of EPMs

- Uncertainty in number of total cycles required
- Does not establish maximum speed of evolution
- Software processes should be focused on flexibility and extensibility rather than on high quality which sounds scary.
  - Prioritize speed as if time-to-market missed project may be meaningless.

The Concurrent Development Model

- Also called concurrent engineering
- Defines a series of events that will trigger transitions from state to state for each of the s/w engineering activities, actions or tasks

- Applicable to all types of s/w development
- Defines a network of activities
- Events generated at one point in process network trigger transitions among states.

Specialized Process Models

- Take on many of the characteristics of one or more of the conventional models.
- Tend to be applied when narrowly defined software engineering approach is chosen.

eg:
- Component-Based development
- The formal methods model
- Aspect oriented software development

Component-Based Development

- Commercial off the shelf software components can be used.
- Components should have well defined interfaces.
- Incorporates many of the characteristics of spiral model.
- Components can be designed as either conventional software modules or OOP classes or packages of classes

The Formal Methods Model

- Encompasses a set of activities that leads to formal mathematical specification of computer software.
- Have provision to apply a rigorous, mathematical notation.
- Ambiguity, incompleteness and inconsistency can be discovered and corrected more easily.
- Offers promise of defect-free software.
- Cleanroom Software engineering is a variation of the Formal Methods model.

Critical Issues:

i) Time consuming and expensive development.

ii) Expensive training required

iii) Difficult to use models as communication mechanism for technically unsophisticated customers.

Aspect Oriented Software Development (AOSD)

'Concerns' - Customer required properties or areas of technical interest - span the entire S/W architecture.

ex: Concerns:
i) Security
ii) Fault Tolerance
iii) Task Synchronization
iv) Memory Management

- Crosscutting concerns - When concerns cut across multiple system functions, features and information, they are often referred to as crosscutting concerns.

- Aspectual requirements - Define those crosscutting concerns that have impact across the S/W architecture.

- AOSD or AOP (programming) provides a process and methodological approach for defining, specifying, designing and constructing aspects

- If a distinct AOSD has not yet matured it is likely AOSD will adopt characteristics of both spiral and concurrent process models

The Unified Process (UP)

- It is a use-case driven, architecture-centric, & iterative and incremental software process.
- UP is an attempt to draw on the best features and characteristics of conventional s/w process models
- Implements many of principles of agile software development.
- UP framework is used for object-oriented software engineering using UML (Unified Modeling Language)

Phases of inception:

- Encompasses both customer communication and planning activities.
- Fundamental business requirements described through a set of preliminary use-cases.

Elaboration:

- Encompasses planning and modeling.
- Refines and expands preliminary use cases.
- or.

- Expands architectural representation to include -
  i) Use-case model
  ii) Analysis Model
  iii) Design Model
  iv) Implementation Model
  v) Deployment model.

Construction:

- Makes each use case operational for end users.
- As components are being implemented, unit tests are designed and executed for each.
- Integration activities are conducted
- Use-~~case~~ cases are used to derive a suite of acceptance tests.

~~Tran~~ Transition

- Software given to end users for beta testing
- Software team creates necessary support info guides.
- At conclusion of transition phase, software increment becomes a usable software release.

Production :

- coincides with deployment activity of the generic process.
- The on going use of software is monitored.
- Support for operating environment is provided.
- Defect reports and requests for changes are submitted and evaluated

Work Products.

- Inception.
- Elaboration
- Construction
- Transition

Agile Process Model

Manifesto -

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan.

Agility -

- Effective response to change
- Effective communication in structure and attitude among all team members, technological and business people
- Drawing the customer into the team. Eliminating 'us and them attitude'. Plan must be flexible
- Eliminate all but essential work products and keep them lean.
- Emphasize an incremental delivery strategy as opposed to immediate products that gets working software to customers as rapidly as feasible

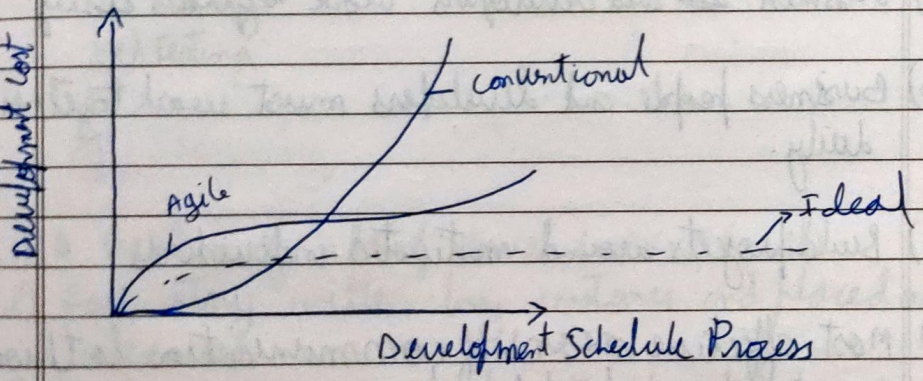
Steps:

- Why - modern business env^n fast-paced and ever-changing. It has demonstrated to deliver successful systems quickly.
- What - Basic activities ie - comm^n, planning, modeling, construction and deployment remain but they push the team toward construction and delivery sooner.

Cost of Change:

Conventional Wisdom - It is the cost of change that increases non linearly as a project progresses It is relatively easy to ~~accomodate~~ accommodate a change when a team is gathering requirements early in a project costs are minimal. But if the middle validation testing, a stakeholder is requesting a major functional change.

A well designed agile process may 'flatten' the cost of change curve by coupling incremental delivery with agile ~~process~~ practices such as continuous unit testing and pair programming. Thus the team can accommodate changes late in software without dramatic cost and time impact.



Agile Process

Driven by customer descriptions of what is required.
- Recognizes plans are short lived
- Develops software iteratively with heavy emphasis on construction activities

- They has to adapt as changes occurs due to unpredictability

- Delivers multiple 'software increments'. Delivers an operational prototype of portion of an operation system to collect customer feedback for adaptation.

Principles

1) Highest priority to satisfy the customer through early and continuous delivery of valuable software

2) Welcoming changing requirements, even late in development

3) Deliver working software frequently with preference shorter time scale

4) Business and developers work together daily

5) Business people and developers must work together daily.

6) Build projects around motivated individuals

7) Most effective means of communication is through face-to-face development

8) It promotes sustainable development.

9) Continuous attention to technical excellence and good design

Simplicity - The art of maximizing amount of work not done.

The best architectures, requirements and designs emerge from self-organizing teams.

Extreme Programming (XP)

Most widely used agile process, originally proposed by Kent Beck.

XP uses and OOP approach

4 framework activities
i) Planning
ii) Design
iii) Coding
iv) Testing

XP - Planning

1) Creation of a set of stories (User Stories)
2) Each story written by customer and placed on an index card.
3) Customer assigns a value to the story
4) Agile team assesses each story and assigns a cost.
5) Stories are grouped to for a deliverable increment.
6) A commitment is made on delivery date.
7) After the first increment "project velocity" is used to help define subsequent delivery dates for other increments

XP - Design

1) KIS - Keep It Simple principle.
2) Encourage use of CRC (class-responsibility-collaborator) cards.
3) For difficult design problems suggest creation of spike solutions
4) Encourages refactoring - an iterative refinement of the internal program design.

XP - Coding

1) Recommends the construction of a series of unit tests for each of the stories before coding commences.

2) Encourages "pair-programming"
   - Mechanism for real-time problem solving and QA
   - Keeps developers focused on problem at hand.

3) Needs continuous integration of the s/w which provides a 'smoke testing' environment.

XP - Testing

1) Unit tests should be implemented using a framework to make testing automated. This encourages a regression testing strategy

2) Integration and validation testing can occur on a daily basis.

3) Acceptance tests, also called customer tests, are specified by the customer and executed to assess customer visible functionality

4) Acceptance tests are derived from user stories

Scrum

- Iteration of plan, build, test, review.
- Product owner - Manager
- Scrum Master - Team Leader.
- Team - Developers and testers

Development Process:

1) Product Backlog: Tasks iterated over many loops
The loops are the entire process till launch phase.

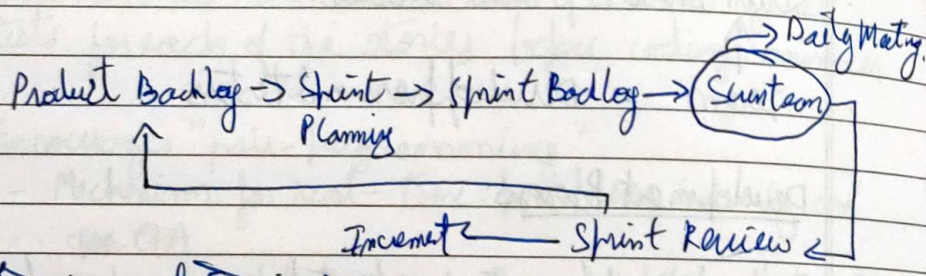
- Each of these tasks / backlogs are termed as user stories.
- The product owner and scrum master prioritize the user stories.
- Using these user stories and priority the a sprint backlog is designed.

Sprint Backlog

1) Sprint Planning - Discussions b/w product owner describes end goal for product

2) Daily Scrum - Developers, testers daily meetings to plan

3) Sprint Review - Each and every user story is usually designed with 2 weeks in mind. At end of 2 weeks Sprint review is performed.



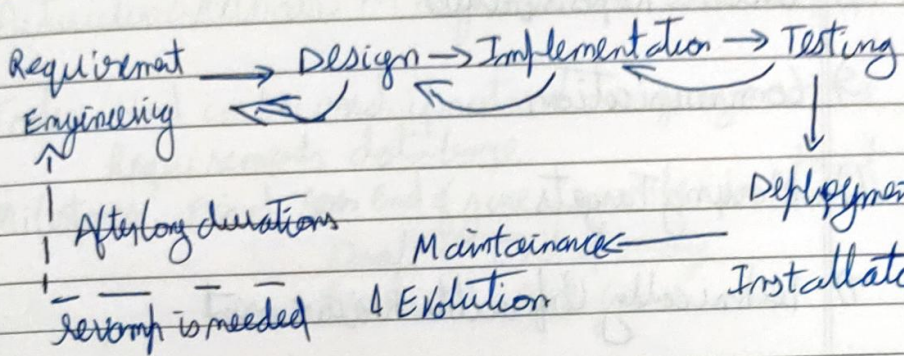
Requirement Engineering

C5- Requirements Engineering

Requirements of a software system

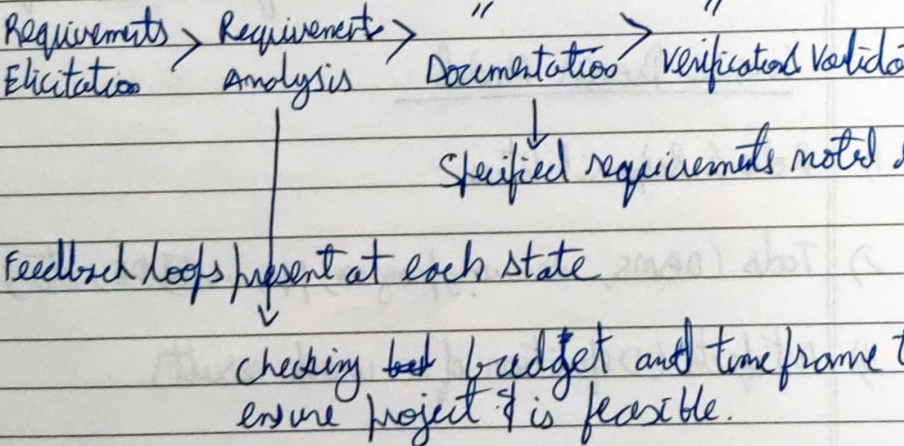
- Requirements engineer is communicator between the stakeholders and technical team.

Process



Feedback loops present at each step in order to backtrack.

Requirement process



Challenges

1) Incomplete or hidden requirements
2) Inconsistent requirements
3) Terminology
4) Unclear Responsibilities
5) Communication
6) Moving Targets
7) Technically Unfeasible Requirement
8) Stakeholders don't know requirements from previous solutions
9) Underspecified requirements
10) Unclear, unmeasurable non functional requirement

Generic Process Model

1) Roles & Responsibilities
2) Tools (DBMS, drawing program, ppt, word & IDE, etc)
3) Artifacts (Definition of a work results)

4) Activities & Methods
5) Milestones
Output: Process Model

Eg: Artifacts: Quality Requirements

Roles: (QA role, lets q/A req. reviewed and signed off)

Activities & Methods: NFR framework to develop QR req.

Tools: Word can be used if natural language or a Requirements database.

Milestones: Final ~~xxx~~ End of june (2 weeks) for final ~~stories~~
Draft: End of may
Reviewed: Mid of June

Class Based Modeling

Approaches:

Noun-Phrase approach → Identify, eliminate, classify → change all found to any into categories
Relevant, Fuzzy, Irrelevant

Common class patterns approach

The class responsibilities collaboration (CRC) approach.

The use case-driven approach

Guidelines for Identifying Classes

Look for nouns and noun phrases in the problem statement
All classes must make sense in the application domain.

Ex:

External attributes (other systems, devices, people)
Things (reports, displays, letters, signals)
Occurrences or events (Property transfer, fund transfer, robot movements)
Roles (Manager, engineer, salesperson)
Organizational Units (division, group, team)
Places (Manufacturing floor, loading dock etc)
Structures (Sensors, vehicles, computers)
Not objects/attributes (Number, type)

Selection characteristic

- Retained Information: Information about class must be remembered so that the system should function
- ~~Needed~~ Needed Services: Must have set of identified operations that can change value of attributes in some way.
- Multiple Attributes: Should be represented in the class
- Common Attributes: Set of CA can be applied or applied to all instances of that class.
- Essential Requirements: Information producing or consuming by external entities should be considered as classes.

Guidelines for Refining Classes

1) Redundant Classes:

- Do not keep two classes that express same information

2) Adjective Classes:

- Does object represented by noun act different when adjective is applied to it.

3) Attribute class -

- Attributes of a feature should not be different class.

Eg: Demographics of Membership are not classes but attributes of membership class.

4) Irrelevant Classes:

- Each class must have a purpose and every class should be clearly defined.

- If no SOP for a class, eliminate it

Eg: Slide 115

Nouns:

1) Store ✗ ✓
2) Inventory ✗ ✓
3) POS terminals ✗ ✓
4) Items ✓
5) Quantities ✗
6) Returns Desk ✗ ✗
7) Loading Dock ✓
8) Arriving shipment ✓
9) Suppliers
10) Meat Dept ✓
11) Produce Dept ✓
12) Loss/Discount ✓
13) Spoilage ✗
14) Handling Returns ✓

Common Class patterns approach

Types of classes:

- Concept
- Event
- Organization
- People
- Place
- Tangible things and device class
- Review the class purpose

1) Concept Class

- Idea or understanding of the problem
- Principles that are not tangible and used to organize of keep track of business activities

Eg: Performance, reservation

2) Event Class:

- Things that happen at a given date and time
- Points in time that must be recorded.

Eg. Request, order.

3) Organization class

- A collection of people, resources, facilities, or groups to which the users belong.

Ex. Accounting department.

4) People Class

- Different roles users play in interacting with application.

Guidelines:

- Classes that represent users of the system.

ex. operator, clerk.

- Classes representing people who do not use the system but about whom information is kept by the system.

ex. Employee, client, teacher etc

5) Places Class

- Areas set aside for people or things
- Physical locations that the system must keep information about.

ex. Building, store, site, travel office.

6) Tangible things

- Physical objects or groups of objects that are tangible and devices with which application interacts.

ex. CDs, ATM.

Q.1) ATM case study
Slide (127)

1) Concept - ~~Deposit, Withdraw, No Access~~ Availability

2) Event - Recorded transaction, Account, checking account, Savings Account

3) Organization - Bank class

4) People - Client

5) Places - Building

6) Tangible - ATM kiosk

Guidelines for naming Classes:

- Should be singular
- Class name should begin with an uppercase letter
- Compound words - First letter of each word.
  Eg: LoanWindow.

Design Engineering

Components of a design model:

- Data design - Transforms information domain model into data structures required to implement software.
- Architectural Design - Defines relationship among major structural elements of a software.
- Interface Design - Describes how the software communicates with systems that interact with it and humans.
- Components level design - Transforms structural elements of the architecture into a ~~for~~ procedural description of software components.

Quality attributes

1) Functionality
2) Usability
3) Reliability
4) Performance
5) Supportability - Resolving issue when system fails

Design Concepts:

1) Abstraction

- Procedural Abstraction - Named sequence of instructions that has specific and limited function
  ex. word OPEN for a door
- Data Abstraction - Named collection of data that describes a data object.
  ex. Door attributes - Door type, swing direction, weight, dimension.

2) Modularity

- Software is divided into separately named and addressable components called modules.

Criteria:

i) Modular understandability: it should be understandable as a standalone unit.

ii) Modular continuity: If small changes to system requirements result in changes to individual modules rather than system wide changes.

iii) Modular protection: If an error occurs within a module those errors are localized and don't spread to other modules.

iv) Modular Composability:

- Design method should enables reuse of existing components.

v) Modular Decomposability

- Complexity of the overall problem can be reduced if the design method provides a systematic mechanism to decompose a problem into sub problems

3) Information Hiding

- Information within a module should be inaccessible to other modules that have no need for such information.
- Hiding defines and enforces access constraints to both procedural detail within a module

4) Functional Independence

- It is achieved by developing modules with single minded function and an aversion to excessive interaction with other modules.

Qualitative criteria:

1) Cohesion: Measure of relative strength of module.
2) Coupling: Measure of the relative interdependence among modules.

5) Cohesion

- The degree to which all elements of a component are directed towards a single task.
- The degree to which all elements directed toward a task are contained in a single component.
- The degree to which all responsibilities of single class are related.

Types:

i) Functional Cohesion: It contains elements that all contribute to the execution of one and only one problem-related task.

Ex: Compute cosine of an angle
Calculate net employee salary

ii) Sequential Cohesion: A sequentially cohesive ~~model~~ module is one whose elements are involved in activities such that output data from one activity serves as input data to the next.

Ex: Module read and validate customer record:
i) Read record
ii) Validate Customer record.

iii) Communicational cohesion: A communicationally cohesive module is one where elements contribute to activities that use the same input or output data.

ex: Find Title of Book
" Price "
" Author "

All these activities are related because they work on the same dataset.

iv) Procedural Cohesion: A procedurally cohesive module is one whose elements are involved in different and possibly unrelated activities in which control flows from each activity to the next.

ex: Add record
change"
delete"

v) Temporal Cohesion: A temporally cohesive module is one whose elements are involved in activities that are related in time.

ex: An exception handler that

- Closes all open files.
- Creates an error log
- Notifies user.

vi) Logical Cohesion

A logically cohesive module is one whose elements contribute to activities of the same general activity or activities to be executed are selected from outside the module.

A logically cohesive module contains a number of activities of the same general kind.

Ex: Read from file
Read from database.

vii) Coincidental Cohesion

A coincidentally cohesive module is one whose elements contribute to activities with no ~~meaningf~~ meaningful relationship to one another

Ex: When a large program is randomly modularized.

b) Coupling

Measure of interconnection among modules in a software structure.

Strive for lowest coupling possible.

Types

1) Data coupling
Ex: an integer, a float, a character etc. Data required by 2 modules

⇒ Same as data by data structure passed.

ii) Stamp Coupling: Two modules are said to be stamp coupled if a data structure is passed as a parameter but the called module operates on some but not all of the individual components of the data structure.

iii) Control Coupling - If one module passes a control element to the other module. The control elements affects/controls the internal logic of the called module

eg: If/Else in one module but function in the condition present in another module.

iv) Common Coupling - Takes place when a number of modules access a data item in a global data area.

v) Content Coupling (Least Desireable):

It occurs when one module branches into another module or modifies data within another

eg: Part of a program which searches for a customer and on not finding adds customer by directly modifying data.

6) Refinement

Process of elaboration

Start with statement of Function in a stepwise fashion until programming language statements are reached.

7) Refactoring

∴ Reorganization technique that simplifies the design of a component without changing its function or behavior.

Design Classes

1) UI class - define all abstractions that are necessary for human-computer interaction.

2) Business domain classes - Refinements of the analysis class

3) Process classes - Implement lower level business abstractions required to fully manage the business domain classes

4) Persistent classes - Represent data stores (eg: database) that will persist beyond execution of the software.

5) System classes - Implement software management and control functions that enable the system to operate and communicate within its computing environment and with outside world.

Control Hierarchy

Represents the organization of program components

- Depth - No. of levels of control
- Width - Span of control
- Fanout - No. of modules that are directly controlled by another module.
- Fan-in - How many modules directly control a given module.
- Superordinate - module that control another module
- Subordinate - module controlled by another.

Software Testing

Test case design:

1) White Box - Cotrol flow graph, basic path testing
2) Black Box - Equivalence Class, Boundary value Analysis

Characteristics:

- Operable
- Observable
- Controllable
- Decomposable
- Simple
- Stable
- Understandable

Test characteristics:

- Good test has high probability of finding error
- Good " not redendant
- " should be "best of breed" - Tests should be likely to uncover a whole class of errors.
- " neither too simple nor too complex

Testing Techniques

- Black-box testing - Knowing the specific function that a product has been designed to perform, test to see that function is fully operational and error free.

- Includes tests conducted on software interface surface.
- Not concerned with internal logical structure of the software.

2) White-box testing

- Knowing the internal workings of a product, test that all internal operations are performed according to specifications and all internal components have been exercised.
- Involves tests that concentrate on close examination of procedural detail.
- Logical paths through software tested
- Test cases exercise specific sets of conditions and loops.
- Testers have access to source code.
- Based on coverage of Code, Branches, Paths, Conditions.

Control Flow Graph

- Abstract representation of structured program.
- 2 major components:
  i) Node: Represents a stretch of sequential code statements with no branches.

ii) Directed Edge - Represents a branch, alternate path in execution

Path: A collection of nodes linked with Directed Edges.

Notation guide for CFG

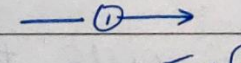
CFG should have:
i) 1 entry arc
ii) 1 exit arc

All nodes should have
i) At least 1 exit and 1 entry arc.

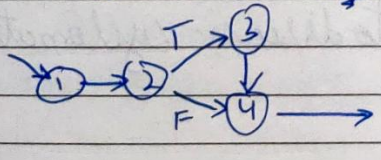
. A logical node that does not represent any actual statements can be added as a joining point for several incoming edges.
-> Represents a logical closure.
Ex: Last node after if-else

Step 1 Number of paths

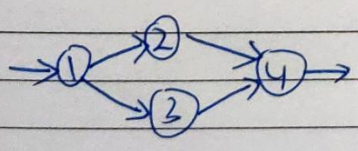
. Given a CFG how do we cover all arcs and nodes

Eg: 

1 Path needed



2 paths 1-2-4
1-2-3-4



2 paths 1-2-4
1-3-4

Step2 Steps to find paths:

1. Draw a CFG for code
2. Compute cyclomatic complexity number C for the CFG.
3. Find at most C paths that covers nodes and arcs in the CFG, also known as Basic Path set
4. Design test cases to force execute in all basic paths

No. of predicates + 1 or
No. of edges − No. of nodes + 2 (E − N + 2)
or
Number of regions in the graph.
↓
Enclosed area in the CFG.

Predicates: Nodes with multiple exit arcs.

Step3 Independent Path:

- An executable or realizable path through graph from start to end node that has not been traversed yet.
- Must move along at least one arc that has not yet been traversed
- Number of independent paths to discover = cyclomatic complexity number.

Step 4 Prepare a test case for each independent path.

4: Example at slide: 205

Graph Matrices

- To develop a software tool that assists in basis path
- Square Matrix - Tabular representation of flow graph.
- Each node represents by number and each edge represented by letter
- Link weight is 1 (a connection exists)
- Link weight is 0 (a connection doesn't exist)

Link weights can be assigned other:
- → Probability of edge will be executed
- → Processing time during traversal link
- → Memory required during traversal a link
- → Resources required during traversal a link

To improve quality of white box testing:
- Condition testing - Simple and compound testing
- Data Flow Testing - selects test paths of a program according to variables
- Loop Testing - simple, nested, concatenated, unstructured

Black Box Testing

- Engineers have no access to source code or documentation of internal working
- BlackBox can be Single unit, Subsystem, whole system
- Tests are based on - Specification of black box, providing inputs and inspecting outputs.

Equivalence Partition Technique

- To ensure correct behaviour of blackbox both valid and invalid cases needs to be tested.

Eg: boolean isValidMonth(int m)
Functionality: check m is [1 . . . 12]
O/P: True if m is 1 to 12
False otherwise

- Usually exhaustive testing is not feasible or necessary. An error in the code would have caused the same failure for many input values If 240 fails then 241 is likely to fail too.

Equivalence Partition

- For a method it is common to have a number of inputs that produce similar outcomes.
- Picking few test cases from each category better method
  15, 9, 1, 12 → ValidMonth(m) ⇒ True.
  ↳ All produce true

- Partition input data into equivalence class.

For isValidMonth(int n) Test cases:

- [-∞ .. 0] Should produce invalid
- [1 .. 12] Should produce valid
- [13, .. ∞] Should produce invalid.

Pick one from each partition eg: {-12, 5, 15}

Combination of Equivalence Class:

- When number of test cases grow very large.

Eg: (XXX) XXX - XXXX
Area code Prefix Suffix → 4 digit number. - 3 classes
(optional) ↳ 3 digit not beginning with 0 or 1 → 3 classes
↳ Present or not Boolean 2 class,
↳ 3 digit [200-999] except 911, 5 classes

Total cases: 2×3×3×5 = 90 test cases

- For critical systems all cases must be tested
- Less stringent testing for normal systems
- Test cases can be combined needing more cases

cases	Area code	Value	Prefix	Suffix
1	F	[-∞ .. 199]	[-∞ .. 199]	[-∞ .. -1]
2	T	[200, 910]	[200 .. 999]	[0000 .. 9999]
3	T	[911]	[000 .. ∞]	[1000 .. ∞]
4	T	[912 .. 999]	[200 .. 999]	[0000 .. 9999]
5	T	[1000 .. ∞]	[000 .. ∞]	[10000 .. ∞]

- In this example area code present affects interpretation of Area code, so all combinations b/w these 2 parameters should be tested.
- 2 x 5 = 10 testcases.

Boundary Value Analysis

- Most errors are caught at boundary of equivalence class

For data structures:
String: Empty, String, 1 char
Array: Empty, 1 element, Full array

Functionality Testing

eg:1) Searching:

Boolean Searching (List aList, int key)
Pre condition - aList has at least one element
Post condition - true if key is in the list aList
false if key is not in aList

2) Stack-Push method

Void push(Object obj) throws FullStackException

Precond - !Full().
Postcond - if !Full() then top() == obj && size == old size() + 1

Invariant: Size() >= 0 && size() <= capacity().

2 equivalence classes:

Valid case: Stack is not full
Input: a non-full stack, and an object obj
Expected result: top() == obj

Invalid case: Stack is full.
Input: "
Expected result: FullStackException thrown.

Metrics for Process and Projects

- Software metrics are quantitative measures.
- Done to assess the quality of technical work products.

Used for:

- Characterize, evaluate, predict, and improve
- Metric - Quantitative measure of the degree to which a system, component or process possesses a given attribute
- Indicator - Metric that provides insight into software process

xx

- Process Metrics - Set of indicators that lead to long-term software process improvement
- Project Metrics - Allows a project manager to assess status of project, track potential risks, uncover problem areas, adjust work flow or tasks.
- Ability to control quality, estimate effort and time, measure input, output and result.
- Production Rates
- Errors uncovered are measured

Software Measurement

- Direct Measures - Software Process (cost), Software product, execution speed.
- Indirect measures - Software product (Functionality, quality, complexity, efficiency)

Size Oriented Metrics

Errors per KLOC - Errors per person-month
Defects per KLOC - KLOC per person-month

Function Oriented Metrics

- Measure of functionality delivered by application
- No. of input, no. of o/p, no. of files etc

Function point = count total * [0.65 + 0.01 x sum(value adjustment)]

Object-Oriented Metrics

- Number of scenario scripts - Use Cases
- Number of key ~~cases~~ classes - Effort needed to develop the software.
- Number of support classes - Amount of effort as potential reuse.

MTTC - Mean time to change - The time to analyse, design, implement, test and distribute a change to all users.

Defect Removal Efficiency (DRE) = E / (E+D)
E - No. of errors before delivery
D - No. of defects after delivery.

Empirical Estimation Method (COCOMO II)

The constructive cost model is an algorithmic software cost model.

Effort = $a * KLOC^b$ (in person/months)
Duration = $c * effort^d$ (in months)
Staffing = Effort / Duration

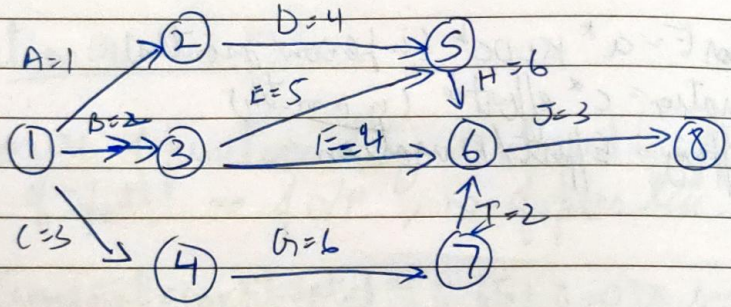
Project Scheduling

- Planning and scheduling occurs after you have decided how to do the work.

Network Diagrams

- It is a schematic display of the logical relationships, among, or sequencing of project activities.

Activity Network Diagram for Project x



Assume all durations were in days, A=1 means activity has duration of 1 day.

PERT/CPM Approach

- Uses network representation of the project
  - → Reflect activity precedence relations
  - → Activity completion time
- Minimizes completion time.

Critical Path Method

- Chain of activities that determines duration of the project

Parameters:

i) Earliest Start (ES) Time
ii) Latest Start (LS) Time
iii) Earliest Finish (EF) Time
iv) Latest Finish (LF) Time
v) Slack Time (ST) = LS - ES, OR LF - EF

- Critical task is one with zero slack time
- Path from start to finish with only critical task is critical path.

To determine optimal schedules: Identify all the project's activities and determine precedence relation among activities

Steps to estimate earliest start time

- Make forward pass through the network

1) Evaluate all activities which have no immediate predecessor.
   Earliest start ⇒ ES = 0
   Earliest Finish ⇒ EF = Activity duration

ii) Evaluate ES of all nodes for which EF of all immediate predecessors is determined

ES = max EF of all immediate predecessors
EF = ES + Activity Duration.

iii) EF of finish node is earliest finish.

Latest Start time / Finish Time

Make a Backward pass through the network;

i) Evaluate all activities that immediately precede the finish node
→ LF = Minimal project completion time
→ LS = LF - Activity Duration

ii) Evaluate LF of all nodes for which LS of all immediate successors has been determined
→ LF = Min LS of all immediate successors
LS = LF - Activity Duration

iii) Repeat till all nodes have been evaluated

Slack Time - Amount of time an activity can be delayed without delaying the ~~project~~ project completion date, assuming no other delays take place.

Slack Time = LS - ES = LF - EF

Critical Path - Set of activities that have no slack, connecting start node with finish

⇒ Critical path is longest path in CPM network.

~~Gant~~ Gantt chart

It illustrates start and finish dates of terminal elements and summary elements of a project

PERT Chart

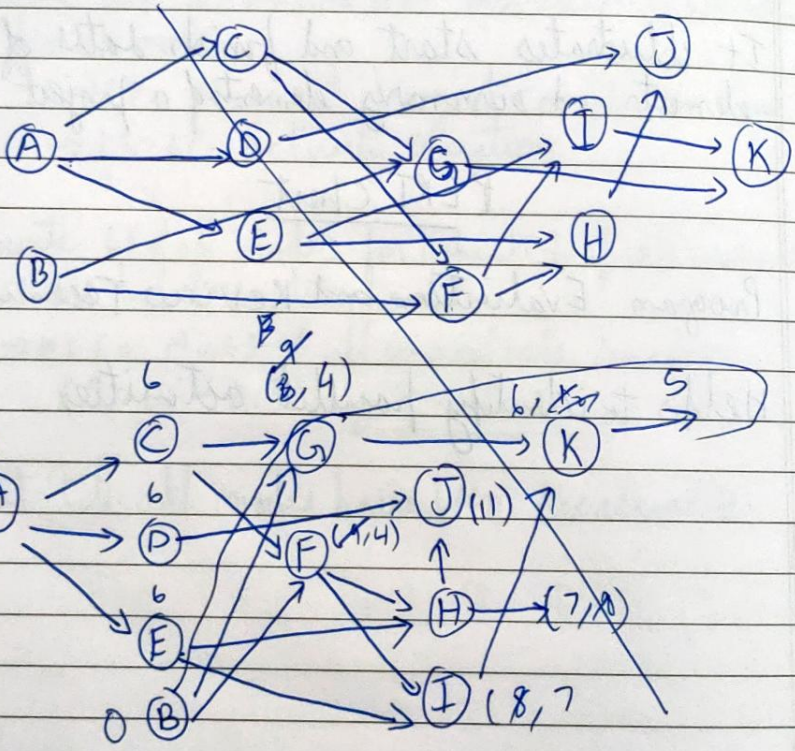
Program Evaluation and Review Technique

Helps to identify parallel activities

Finding CPM

eg:

Activity	Predecessor	Time
A	-	6
B	-	4
C	A	3
D	A	5
E	A	1
F	B, C	4
G	B, C	2
H	E, F	6
I	E, F	5
J	D, H	3
K	G, I	5

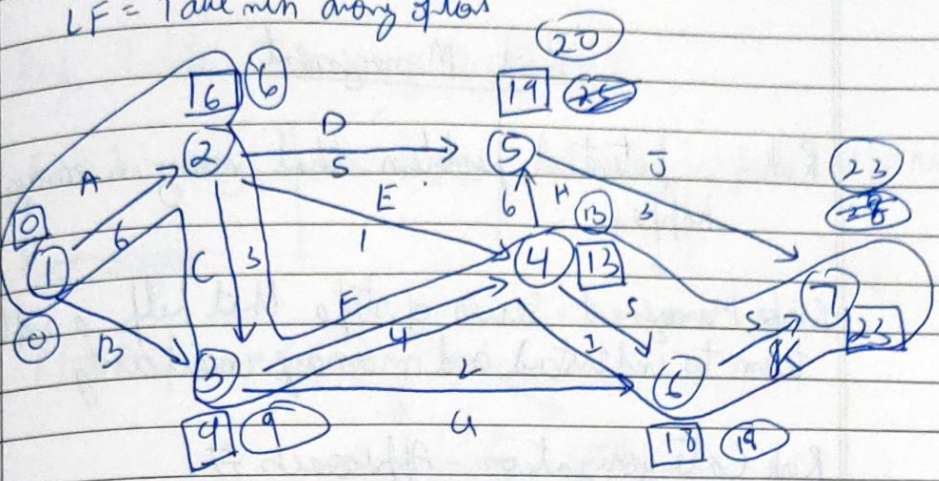


○ → LF

□ → ES

ES = Take max among of tail

LF = Take min among of tail



EF = ES + t, LS = LF - t, Slack = LF - EF or LS - ES

Activity	Time	ES	EF	LS	LF	Slack
✓ A	6	0	6	0	6	0
B	4	0	4	5	9	5
✓ C	3	6	9	6	9	0
D	5	6	11	15	20	9
E	1	6	7	12	13	6
✓ F	4	9	13	9	13	0
G	2	9	11	16	18	7
H	6	13	19	14	20	1
✓ I	5	13	18	13	18	0
J	3	19	22	20	23	1
✓ K	5	18	23	18	23	0

The estimated project completion time is max EF at node 7 = 23

Risk Management

Risk - a potential problem that may or may not happen.

Risk Management - Series of steps that help a software team to understand and manage uncertainty

Risk Categorization - Approach #1

- Project Risks
- Technical Risks
- Business Risks

Approach #2
- Known Risks
- Predictable Risks
- Unpredictable Risks

Risk Identification

- Generic Risks - Potential threat to every software project
- Product-specific risks - Risks that can be identified only by those with a clear understanding of the technology, people and environment specific to software to be built.

Risk Table

Risk Summary	Risk category (7 categories)	Probability	Impact (1-4)	RMMM
~~Risk Info~~				

Risk Exposure (RE) = P x C
↓ ↘
Probability Cost

Strategy - Risk Mitigation, Monitoring and Management

Change Management

SCM - Software Configuration Management

Software Configuration

The o/p of s/w process makes up software configu

$n <= y$ $-5 <= y <= 4$

Test Cases	n	y	result expected
1)	$[-\infty, -6]$	$[-\infty, -6]$	False
2)	$[-\infty, 4]$	$[-5, 4]$	True
3)	$[-\infty, \infty]$	$[5, \infty]$	False

Equivalence classes

1) $y = [-\infty, -6]$, $n = [-\infty, -6]$
2) $y = [-5, 4]$, $n = [-\infty, 4]$
3) $y = [5, \infty]$, $[\infty, \infty]$

Boundary values $n \Rightarrow -6, 4$

$y \Rightarrow -6, -5, 4, 5$

User I/P = 5 X 3

O/P = 10 X 5

inq = 5 X 6

files = 8 X 7

int int = $\frac{3 \times 10}{181}$

Adjustment = 1.07

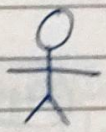
Use case diagram

The requirement specification uses natural language.
The analysis model uses a formal or semi-formal notation we use UML.

Use Case

- Use case describes interactions of actors with the target system to perform a function.
- The use case model can be documented by drawing a use case diagram and writing an accompanying text elaborating the drawing
- The use case diagram consists of:

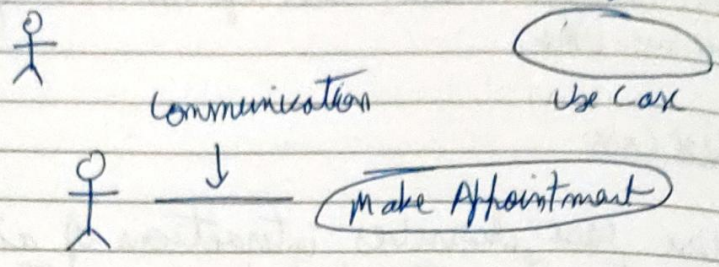
1) Actor - External objects that produce or consume data, must be external to system.
eg: humans, machines, sensors etc.
- Each actor must be linked to an use case
- Labelled using a descriptive noun or the phrase.
eg: Librarian, Doctor etc

~~Symbol~~ Symbol 

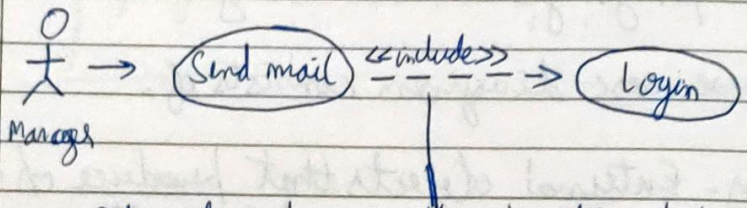
2) Use case - Labelled using a descriptive verb-noun phrase

- Represented using an ellipse with name of the use case written inside or below the ellipse.

Eg: A patient calls the clinic to make an appointment for a yearly check up.



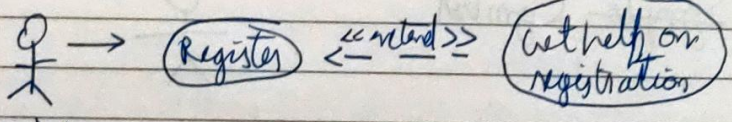
Relationships represent communication between actor and the use case.



Represents inclusion of the functionality of one use case within another.

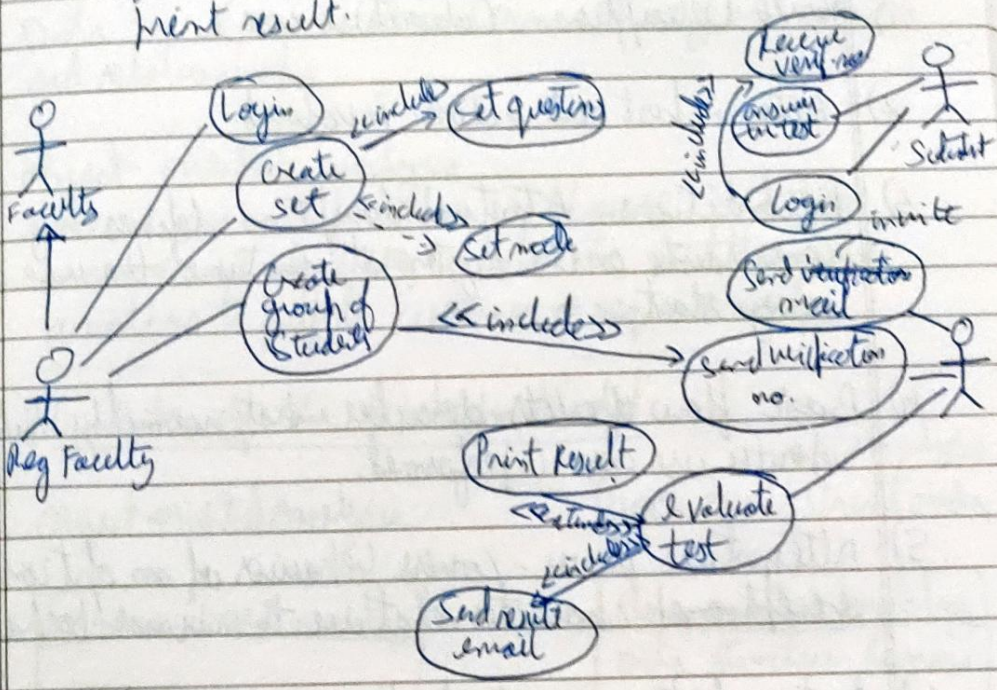
Extend relationship

Represents extension of use case to include optional functionality



Q.1) PSI

Actor - Faculty, reg faculty, students
Usecase - Exams, login, create, set q, set mode of test, groups of students take a test, send mail login using verification no.
System evaluate answers and send result via mail or print result.



Functional Requirements

- Describe the interactions b/w system and its environment independent from the implementation
eg operator must be able to define a new game

Nonfunctional requirements

- Aspects not directly related to functional behaviour.

eg. Response time must be less-than 1s

Constraints or Pseudo Requirements

Imposed by client or ~~requirement~~ environment.
Eg:- Implementation language must be java.

Use Case Specification

1) Brief Description - (abstract use case)
2) Actors - List out actors involved.
3) Preconditions - A textual description defines any constraints on the system at the time the use case may start.
4) Basic flow of events - describes what normally happens when the use case is performed.
5) Alternative flows - covers behaviors of an optional or exceptional character relative to normal behavior.
6) Post conditions - A textual description defines any constraints on system when use case terminates.

Date: / /

Activity Diagram

Analysis modeling

1) Structured analysis:

Considers data and the process that transform the data as separate entities.
Data objects are modeled in terms of only attributes and relationships

2) Object-oriented analysis:
Focuses on the definition of classes and the manner in which they collaborate with one another to fulfill customer requirements

Elements of the Analysis Model

Object-oriented analysis

Analysis
Procedural / structured

Scenario-based modeling
Use case text Use case diagram Activity Swimlane

Flow-oriented modeling
Data flow diagram Data flow diagram Control-flow Processing narratives

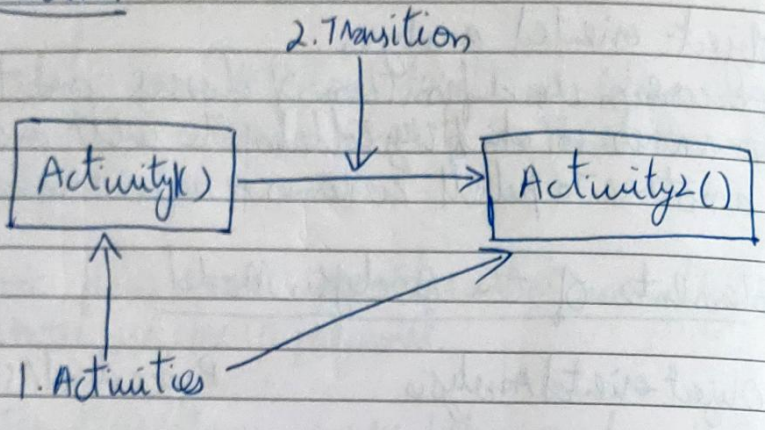
Class based
Class diagram Analysis CRC Collaborative

Behavioral
State Diagram Sequence diagram

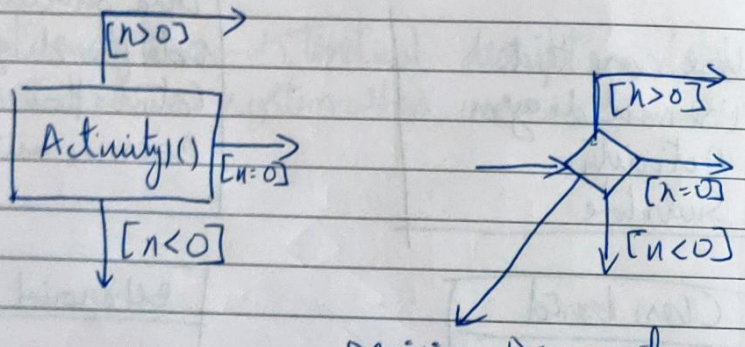
Developing an Activity Diagram

- Useful when you know that an operation has to achieve a number of different things and you want to model what essential dependencies b/w them are before you decide in what order to do them
- Represents the workflow of the process.

Notation

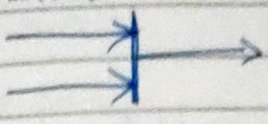


Notation 2

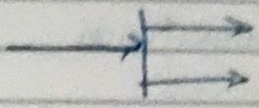


Decision Diamond.

Notations



Synch Bar (Join)



Splitting Bar (Fork)

Notation-3

 → Start Marker

Stop Marker

Notation-4

Developers	Testers	Markers
swimlane	Swimlane	Swimlane

Activity diagram contains

- Activity states and action states
- Transitions
- Objects

~~Activity states~~

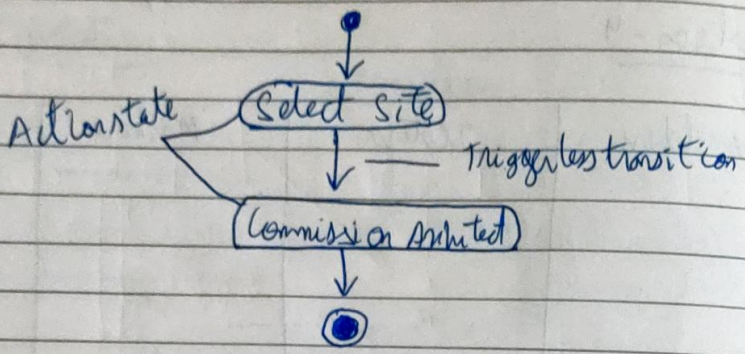
Actions states are atomic and can't be decomposed

Activity states can be further decomposed.

Transitions

- Action or activity of a state completes, flow of control passes immediately to next action or activity state.

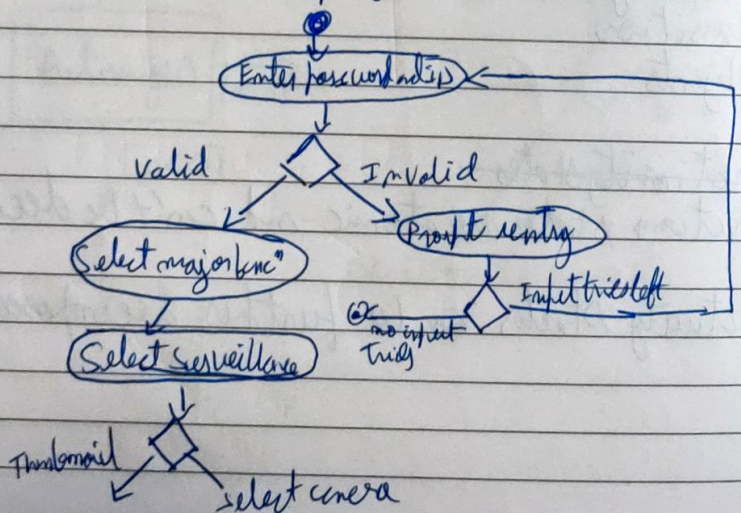
- a initial state -- a solid ball.
- a stop state -- a solid ball inside circle



Branching:

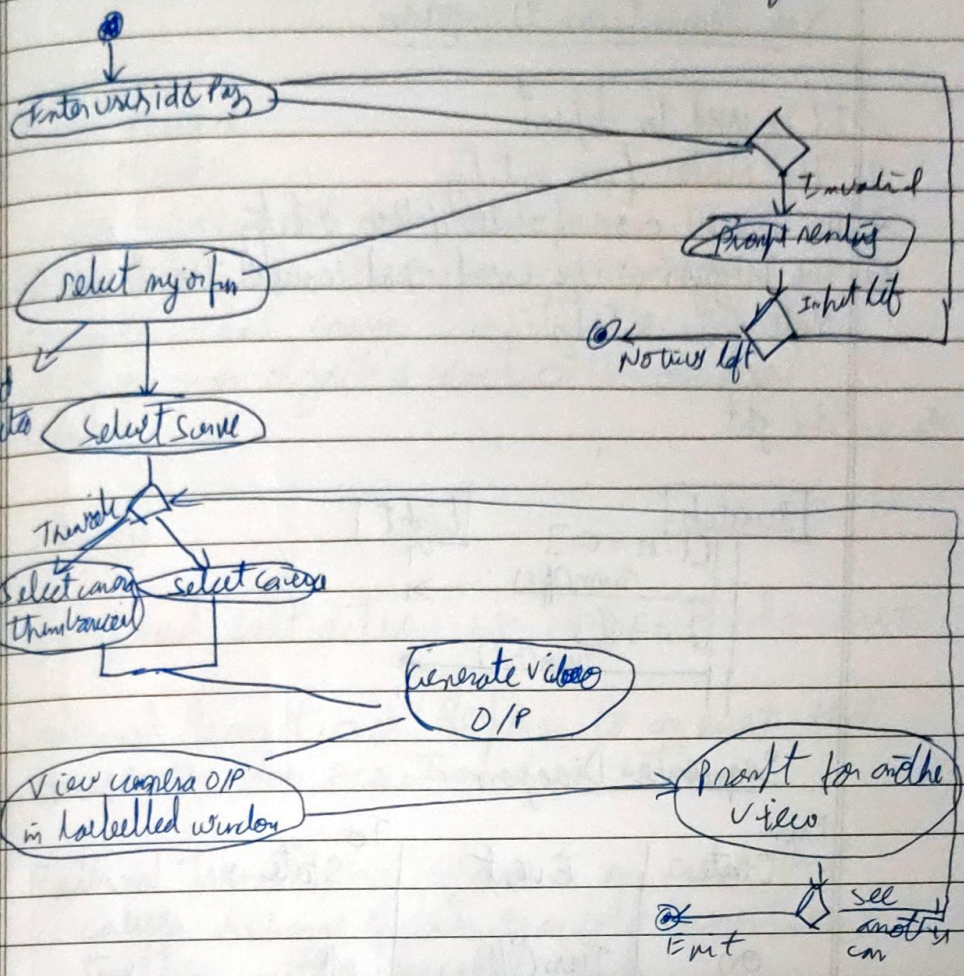
Paths taken on some Boolean expression

eg. Camera surveillance display



Swimlane

Homeowner | Camera | Interface

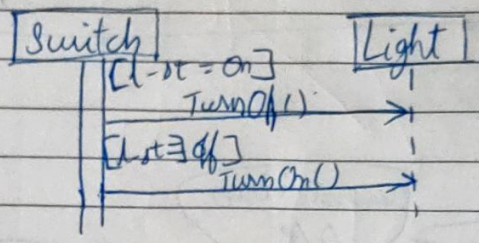


State Diagram

State Transition Diagram

It is used to depict:
1) The states of an entity
2) The transitions of states of the entity
3) The Trigger or the event that caused the transition of state of the entity.

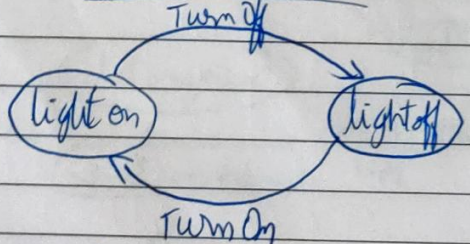
Ex: Light



Sequence Diagram

From State	Event	To State
on	TurnOff	off
off	turnOn	on

State Transition Table



State transition diagram

State Diagrams are used to show possible states a single object can get into

Elements:
- Start Marker
- Stop Marker
- States - box with rounded corners
- Transitions - Shows arrows b/w states
- Events - that cause transition b/w states
- Action - an object's reaction to an event
- Guard

States:

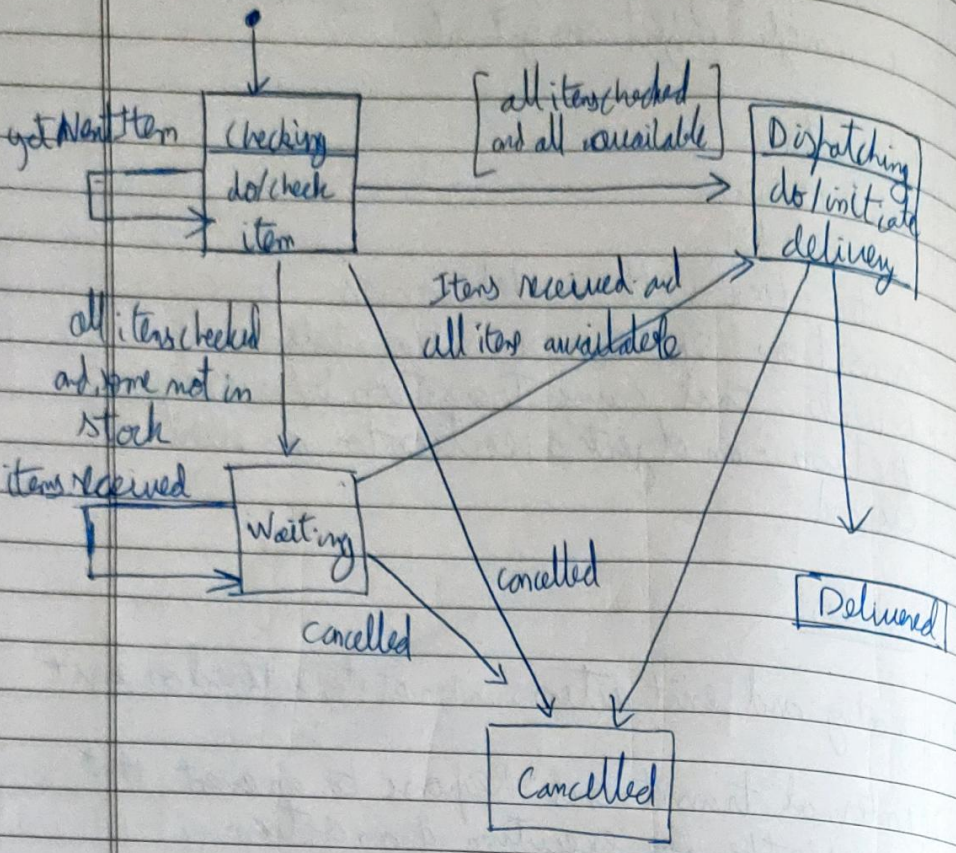
1) Entry and exit actions - when state entered or exit

2) Internal transitions: Response to an event that causes the execution of an action.

3) External transition - Response to an event that causes a change of state or a self-transition, together with a specified action

4) Activities - Once the system enters a state it sits idle until an event triggers a transition.

Ordering an Object - State diagram



Transitions

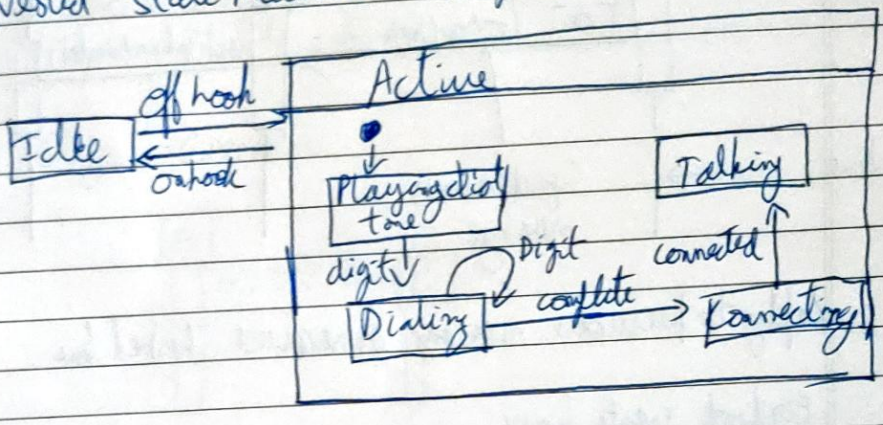
- Source state and target state
- Trigger event - event that makes the transition fire.
- Guard condition - a Boolean expression which must equate to true for trasition to fire.
- Action - An executable atomic computation that can directly act on the object that owns the state machine.

- Initial and final states - shown as filled black circle and a filled black circle surrounded by an unfilled circle

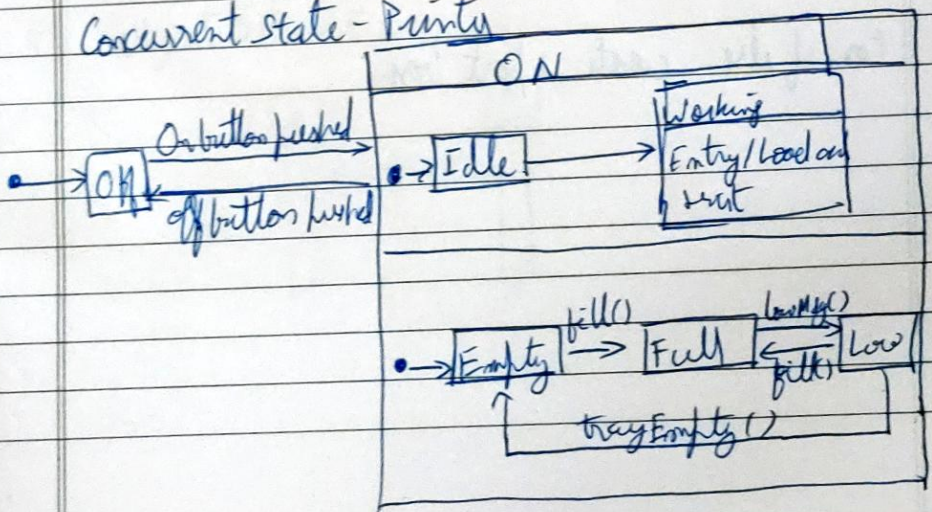
Types:

- Protocol State Machines - These are used to express a usage protocol or lifecycle of some classifier.
- Behavioral State Machines - These are specialization of behavior and is used to specify discrete of a part of designed system

Nested State Machine - Telephone



Concurrent state - Printy

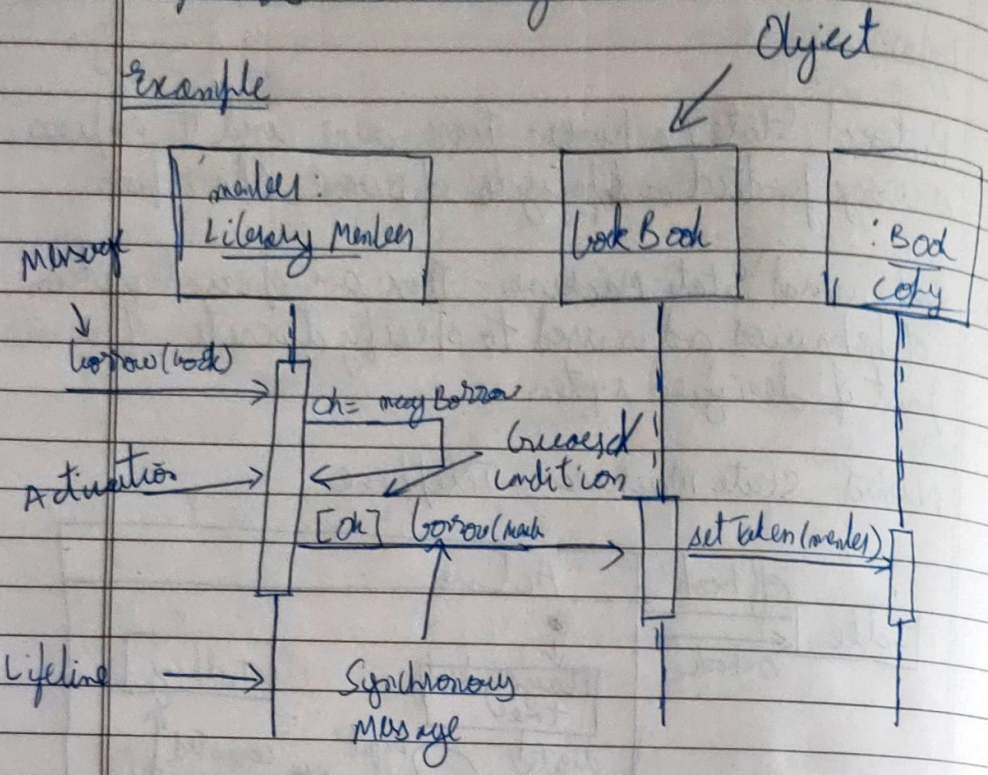


Sequence Diagrams

They illustrate how objects interact with each other

- Emphasizes time ordering

Example



- Reply to previous message or request dashed line.
- For loop create box
- For if else create 2 part box.

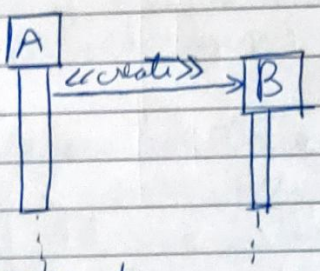
CRC - Class Responsibilities Collaborator

~~Copy~~ CRC for copy and Book

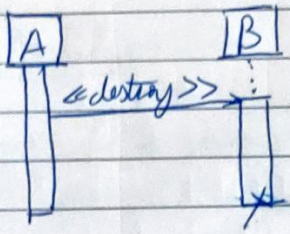
Copy	
Responsibility	Collaborators
Maintain data about a particular book copy. Inform corresponding book when borrowed ad returned	Book

Book	
Responsibility	Collaborators
Maintain data about one book Know whether there are borrowable copies	

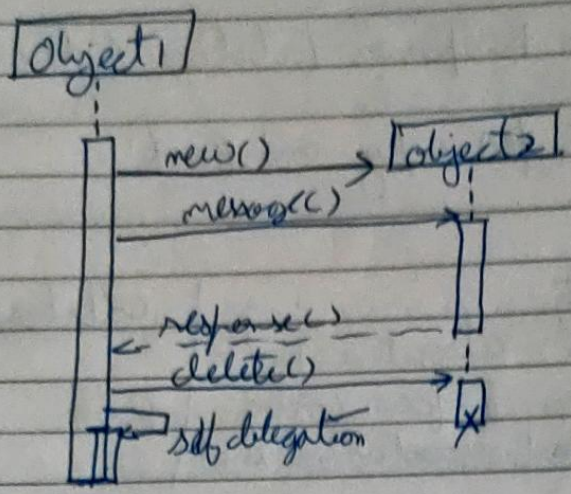
Object Creation



Object Destruction



Ex:

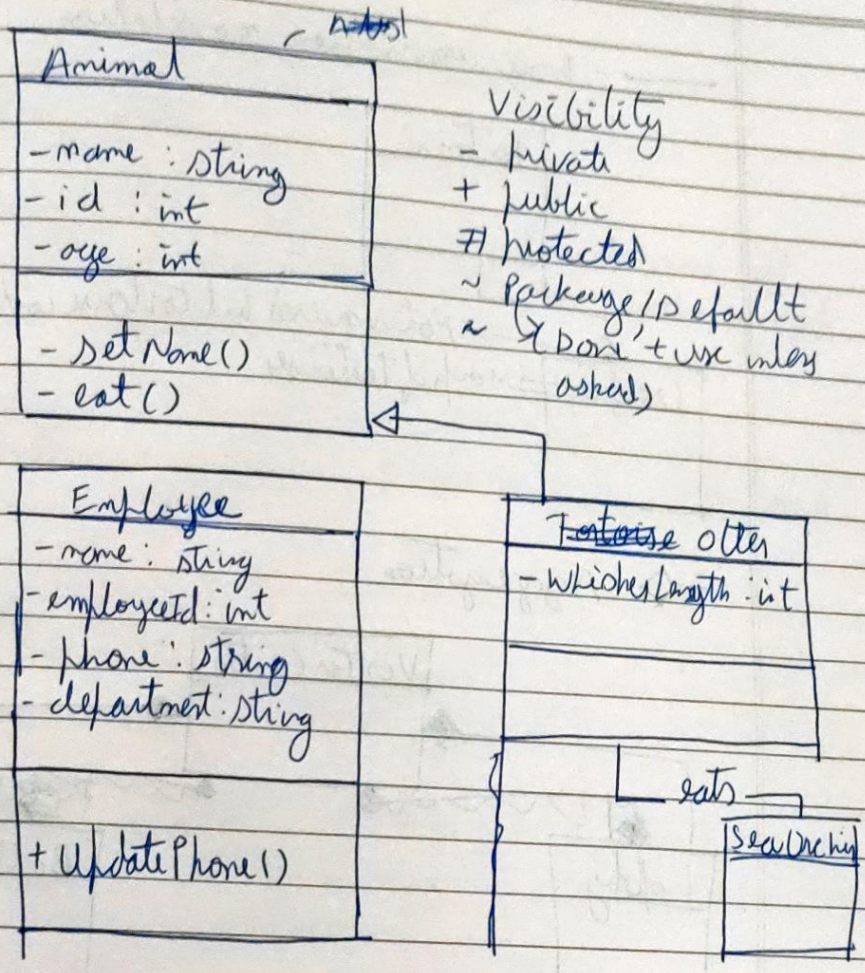

Object1
new()
Object2
message()
release()
delete()
self delegation

Class

~~Use Case~~ Diagram

papergrid

Date: / /



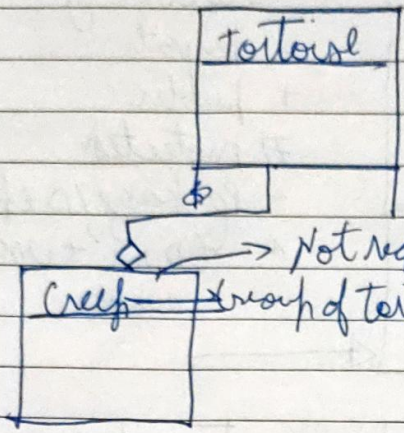
→ Inheritance → child class inherits all of parent attributes

Abstraction: For example Animal is abstract class as we don't initiate animal class we always use the animal such as tortoise, lion etc.

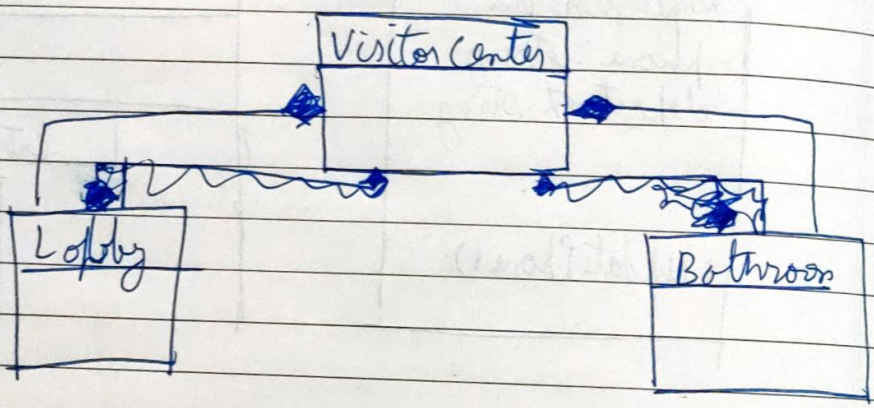
Here Animal to be in Italics → Animals or <<Animal>>

Class Names – Noun phrases from text

——— - Basic association, no relation



◇— Aggregation



If visitor center broken down lobby and bathroom can't exist this is composition

◆— composition

1 / m or 1...* / 0...* / 0..1 → Represents many

Multiplicity

Use Case Diagrams

Ex: Banking ATM

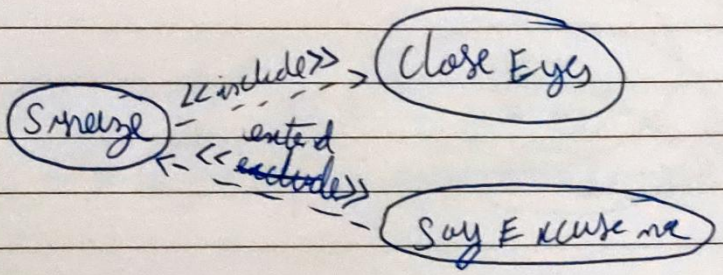
Primary Actor -> Initiates use of system or left
Customer does something on Banking atm
first / On left side

Secondary -> Reactionary / On right side
Bank when customer checks balance then bank reacts

-------> Association

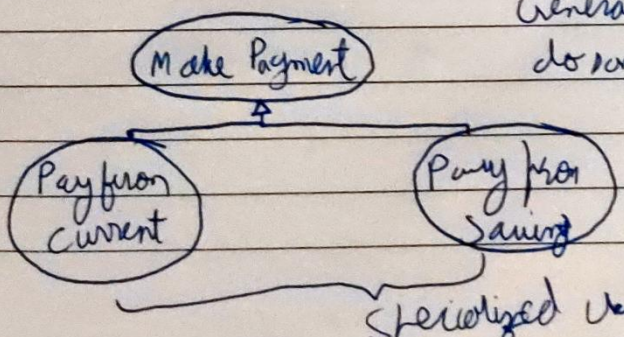
<<include>>
- - - -> Include use casse means included
case needed to complete base use case

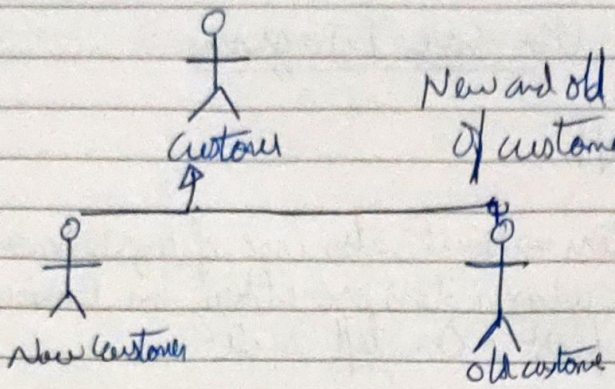
<<extends>>
<- - - - - Extend use case is means extend case need not happen every time, certain condition must be met



Multiple base use cases can point to same include case.

Generalization as both do same function





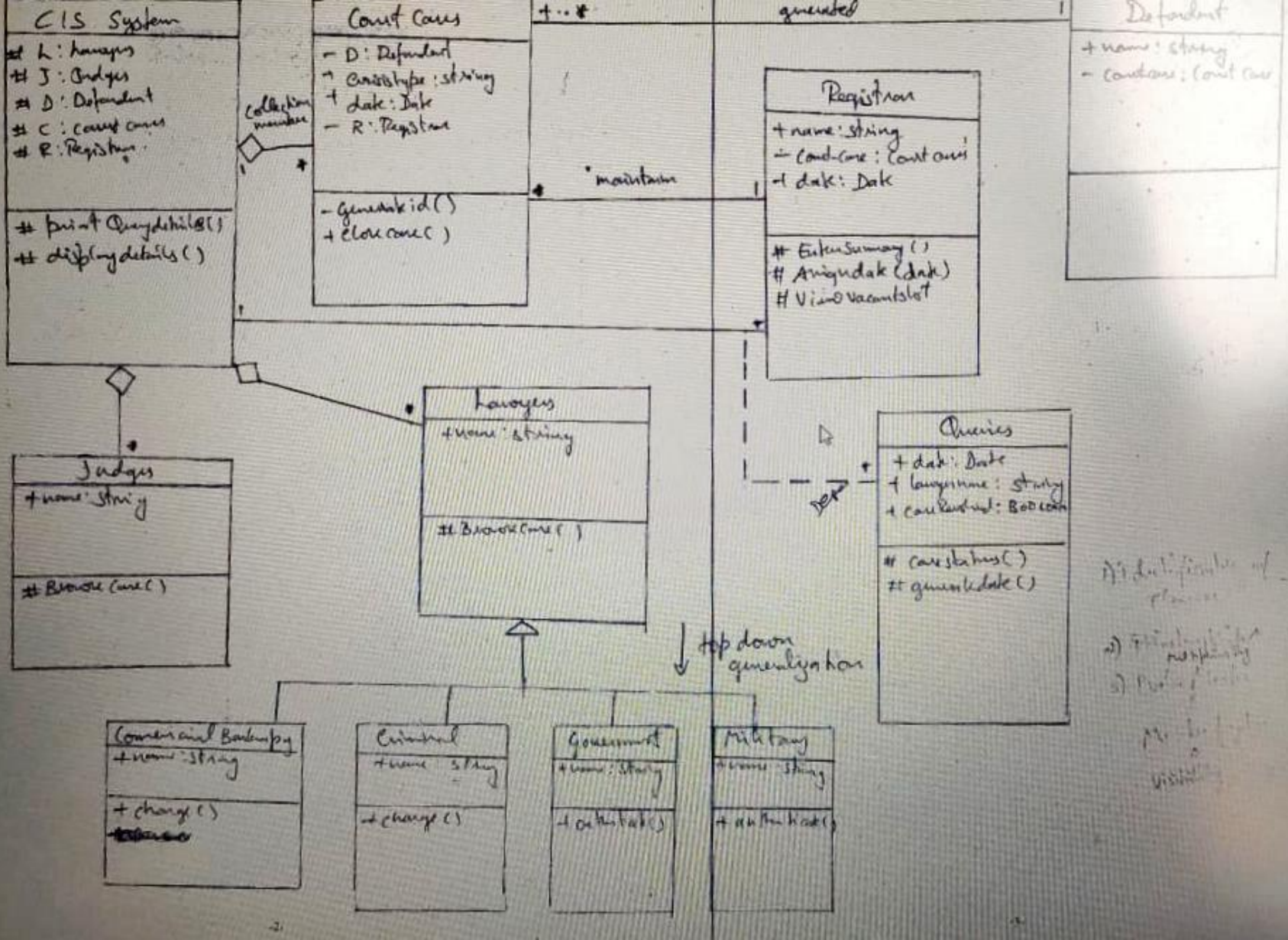
User I/P = 5 x 3
O/P = 10 x 5
inq = 5 x 6
files = 8 x 7
int int = 3 x 10
181

Adjustment = 1.07

and replicated copies of the document in any of the above-mentioned states.

5. Draw a class diagram for the following Criminal Information System (CIS) software. 4
The attorney general's office wants to develop a CIS, to help handle court cases and also make the past court cases easily accessible to the lawyers and judges. For each court case, the name of the defendant, defendant's address, the crime type (Eg. theft, arson etc), when committed (date), where committed (location), name of the arresting officer, and the date of the arrest are entered by the court registrar. Each court case is identified by a unique case identification number (CIN) which is generated by the computer. The registrar assigns a date of hearing for each case. For this the registrar expects the computer to display the vacant slots on any working day during which the case can be scheduled. Each time a case is adjourned, the reason for adjournment is entered by the registrar and he assigns a new hearing date. If hearing takes place on any day for a case, the registrar enters the summary of the court proceedings and assigns a new hearing date. Also, on completion of a court case, the summary of the judgment is recorded, and the case is closed but the details of the case are maintained for future reference. Other data maintained about a case include the name of the presiding judge, the public prosecutor, the starting date, and the expected completion date of a trail. The judges should be able to browse through the old cases for guidance on their judgment. The **Commercial Bankruptcy** Lawyers and **Criminal Lawyers** should also be permitted to browse old cases but should be charged for each old case they browse. Government and military lawyers need to be authenticated to browse old cases without any charge.

6. What are the criteria defined for good software design with respect to coupling and cohesion? 2


CIS System
L: lawyers
J: Judges
D: Defendant
C: court cases
R: Registrar
printQuerydetails()
displaydetails()
Court Cases
- D: Defendant
- CaseStatus: string
+ date: Date
- R: Registrar
- generateid()
+ closecase()
collection number
maintain
generated
Registrar
+ name: string
- cand-case: Court cases
+ date: Date
# EnterSummary()
# Assigndate(date)
# ViewVacantslot
Defendant
+ name: string
- courtcase: Court case
Lawyers
+ name: string
# BrowseCase()
Judges
+ name: String
# BrowseCase()
Queries
+ date: Date
+ lawyername: string
+ caseRandval: Boolean
# casestatus()
# generatedate()
ref
top down generalization
Commercial Bankruptcy
+ name: string
+ charge()
Criminal
+ name: string
+ charge()
Government
+ name: string
+ authorise()
Military
+ name: string
+ authorise()