SW- Engineering

→ Intro, need - ?, Imp
Evolution -

→ SDLC

→ Software Process

→ SDLC Models.

→ Prototype.

→ Comparison of all models

programs and related
documentation such as

Software Engineering

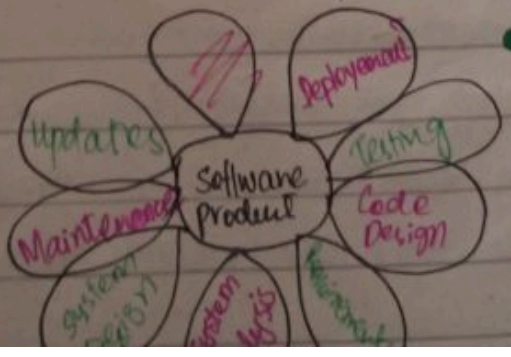
It's a portmt of two words:

Software is a collection of integrated programs, computer programs and related documentation such as requirements, design models and user manuals.

Engineering

Engineering is the application of specific and practical knowledge of invent, design, build, maintain and improve frameworks, processes etc.

Software Engineering is an engineering branch related to the evolution of software product using well defined scientific principles, techniques. The result of software engineering is an effective and reliable software product. or



- It's a systematic, disciplined, cost-effective techniques for software developed Engineering approach to develop a software.

Evolution

1945—65 → Origin — many dev started — not approached development.
— software dev
— build & fix, sell

1965—85 → crises — not dev by full approach. — 50% not deliver
— only 2% software useable — OS/360

1990—2000 → Internet — Windows — Web-browsers

2000—2010 → light weight — mobile generation — internet,

2010—Till → AI, DL, ML — machine, deep learning
— self learn concept, AI

:Note: Biggest benefit, evolution — most of the softwares are free. to use, ppl can use and learn working.

Why SE required?

for following reasons:

- To manage large softwares
- For more scalability
- Cost Management
- Better Quality Management
- To manage the dynamic nature of software

Need of SE

necessity of SE appears because of a higher rate of progress in user requirements & environment. on which the program is working.

- Huge programming: as simpler to build a wall than house, similarly, as the measure of programming became extensive engineering has to step to give it a scientific process.

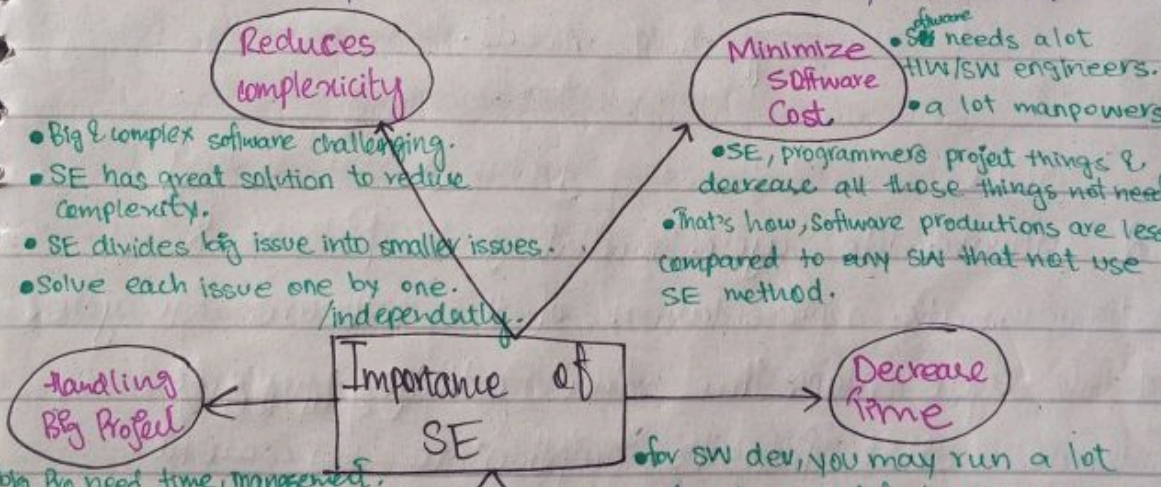
- Adaptability: If softw... on scientific and engineer... to re-create new soft...
- Cost: As manufactu... and electronic hardwa... high if proper proces...
- Dynamic Nature: C... of programming huge... which client works... changing, new updat...
- Quality Manageme... development pro...

Reduces complexicit...
- Big & complex software challe...
- SE has great solution to complexity.
- SE divides big issue into...
- Solve each issue one ...

Handling Big Project
- big Pro need time, manage...
- And to invest $ in company ... requires heaps of planning, direction & maintenan...
- Pro must be complet... after investing 4-5 company months
→ SE method...

Reliabl... Softwa...

- **Adaptability:** If software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
- **Cost:** As manufacturers let down the cost of PC and electronic hardware · but cost of programming remains high if proper process not adapted.
- **Dynamic Nature:** Continually growing & adapting nature of programming hugely depends upon the environment in which client works. If Quality of software continually changing, new upgrades need to be done in existing.
- **Quality Management:** Better procedure of software development provides a better & Quality software product.



- Big & complex software challenging.
- SE has great solution to reduce complexity.
- SE divides big issue into smaller issues.
- Solve each issue one by one. /independantly.

- SW needs a lot HW/SW engineers.
- a lot manpowers.
- SE, programmers project things & decrease all those things not need.
- That's how, Software productions are less. compared to any SW that not use SE method.

- big Pro need time, mangement.
- And to invest 6,7 company months, it requires heaps of planning, testing direction & maintenance.
- Pro must be completed after investing 4,5 company months.
→ SE method.

- for SW dev, you may run a lot codes to get definite run code
- This is time consuming if not well-handled
- IF you make SW according to SE method, it decrease time.

- SW should be secure, work until subscription/time.
- IF any bugs come, company is responsible for solving.
- Bcz in SE, testing & maintenance is given, so no worry.

- Effectiveness comes if made according to standard
- SW standards are companies big targets.
- SW become more effective with help of SE.

Software Processes.

Software —

Software process is the set of activities & associated outcome that produce a SW product. SW engineers do:

- Software specifications:
The functionality of SW & constraints on it's operation must be defined.
- Software Development: Software to meet the requirements must be produced.
- Software Validation: Software must be validated to ensure, it does what customer wants.
- Software Evolution: The software must evolve to meet changing client needs.

SDLC

A software life cycle model is a pictorial & diagrammatic representation of the software life cycle. A life cycle maps the various activities performed on software product from it's inception to retirement. The software is said to have a life cycle composed of several phases. Each phase results in the development of either a part of the system or something associated with system, such as a test plan or user manual.

System Development Life Cycle

Planning & requirement analysis

— requirement analysis most imp stage in SDLC.
— senior members of team perform it with inputs from all stakeholders & domain experts.
— Planning for quality assurance & identifications of the risks associated with project done here.

Defining Requirements

— Once requirement analysis is done, next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.
— This is accomplished through "SRS" - Software Requirement Specification) document which contains all product requirements to be be constructed & developed during project life cycle.

Designing the Software

— Next phase is to bring down all the knowledge of requirements, analysis and design of software project.
— It's a product of last two, input from ~~user~~ customer & requirement gathering.

Developing the Project

— In this phase of SDLC, actual development begins
— Programming is built.
— Implementation of design begins concerning writing code.
— Dev follow coding guidlines by thier management & programming tools like compilers, interpreters, debuggers, etc — used to dev & implement code.

Testing

— After code generated.
— it's Tested against reqiurements
— to make sure products working on reqiurements & solving needs.
— unit testing, integration testing, system testing, acceptance testing done.

Deployment

— One software is certified, no bugs, no error are stated, it's deployed.
— based on assessment, software may be released as it is or with suggested enhancement in object segment.
— After project is deployed, it's maintenance begins.

Maintenance — once client starts using developed systems, then real issues come up. & requirements to be solved time to time.
— This where care is taken for the developed product known maintenance.

Classical Waterfall model.

- Originate in 1970's era.
- Only appreciable use in small projects.
- Software team adapts waterfall model for production of SW, then proper detailed planning & proper accurate work is needed in every phase of model.
- bcz. while following this model, SW team is not allowed to move to previous phase.

1 Feasibility study

- whether project is feasible or not.
- technically or financially. feasible?
- whether project is too lag lost budget is too low.
- is feasible? is team available? — to the budget? — can you make?

2 Requirement Analysis & specifications

- feasibility study report is build from gathering information.
- After gathering information, it is determined whether the requirements can be fulfilled or not.

3 Design.

- before, all info gathered & requir Ana is complete.
- SW design is drawn.
- different diagrams are also drawn for SW.
- Some of it are class D, Activity D, DFD, & state transition diagram
- When Team satisfied with proposed design, move to next phase.

Coding, Unit, 4

Implementation

- The system is first developed in small programs called units,
- which are integrated in next phase.
- before starting implementation, SW team completely design the system. & now ready to develop.
- Programs are written
- All designs are converted into computer programs, call SW.
- unit testing, of small modules.
- then modules are integrated into system.

System 5

Testing

- Before start testing.
- SW team completely develop all SW.
- Now time to test SW using various SW testing techniques
- like white box, black box, gray box testing techniques
- when Team satified with testing & passes the SW for implementation, move to next.
- Alpha testing ↙ developers.
- Beta testing ↙ Customer checking
- Acceptence testing ↙ customer decides whether to accept or not.

Deployment 6

- before deployment.
- SW team completely tested all software.
- Now time to install or deploy the software in it's working environment.

Maintenance 7 Imp

- most effort phase.
- fixing technical or non-technical bugs appearing
- when SW deployed there can be many bugs.
- new modification needs?
- Maintenance includes debugging & new feature addition.

Advantages

- Base model — as further modification & features introduced in next model keeping it base.
- Simple & Easy, Good for small projects.
- When you move to next phase, you have a clear picture of all previous phases.

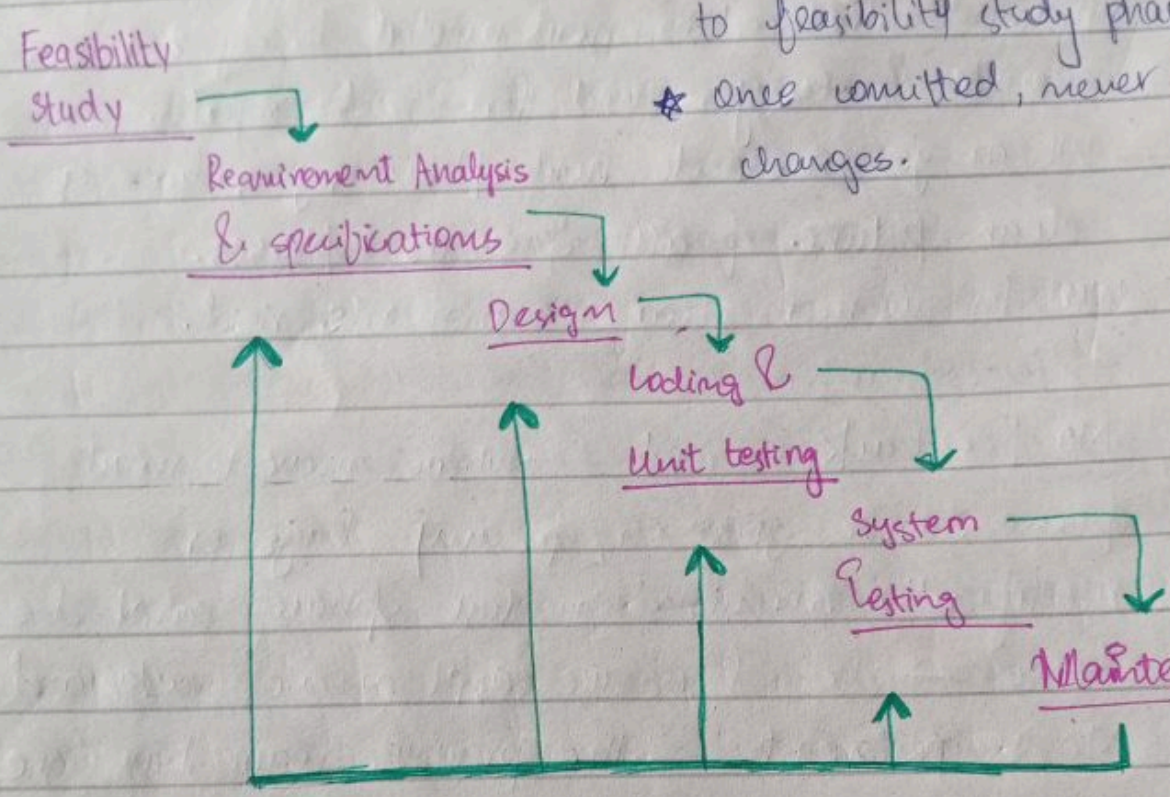
Work for customer's clear requirements, rigid. ← work.

Dis-advantages

- No feedback — rigid — when move to next phase, work goes on, if any bugs or modification required in last phase can't be done. — bcz here we can't move backward.
- No Experiment. — the previous team has done can't be alter when hand-over to next phase.
- No parallelism — no multiple-team work. only one team is working at time.
- High Risk.
- Big - problem if testing fails SW.
- Maintenance required more time 60%.

Iterative Waterfall Model

* Modified version of classical waterfall model.
* One of the major modification is feedback.
* In case there's problem occur is one phase, we can give feedback to that respective team to remove, change.



* there's no feedback to feasibility study phase.
* once committed, never changes.

V-

* also

Advantages:

- Base model for small projects
- Simple & Easy
- feedback.

Dis-advantages

- No phase-overlapping — one phase is completed at time.
- Rigid (changes not).
- No intermediate delivery (whole system delivered) i.e some deliver prototype first.

Testing-focused model.

V-Model

★ also known as verification & validation model.
verifying things produced during step-by-step based on testing of each phase.
after developing the whole system, verifying whether it's working according to needs or not.

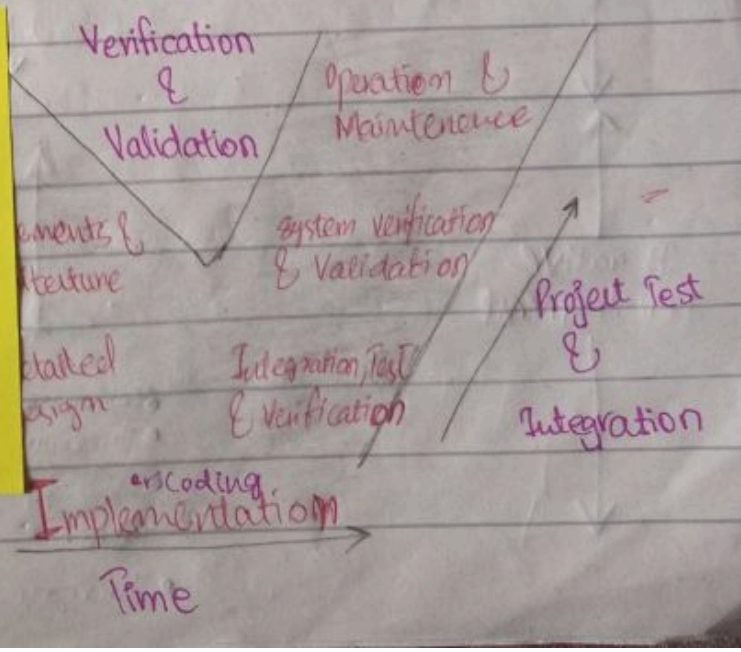
★ Extension of Waterfall model — follow all the steps of SDLC, bit extra steps here testing is perform at every step. , unit testing

★ Testing is associated with every phase of lifecycle. — parallel testing.

Verification phase (Requirement analysis, System Design, Architecture Design, Module Design).
full system + module, unit designing designing sub-module designing

★ Validation phase (Unit testing, Integration testing, System testing, Acceptance testing).
sub-module module-
whole system. client acceptance.

- In V model all phases in design in a parallel way. testing will be performed on each step.
- The verification from one side of V and Validation is performed on another side of V.



every step. , unit testing

★ Testing is associated with every phase of lifecycle. — parallel testing.

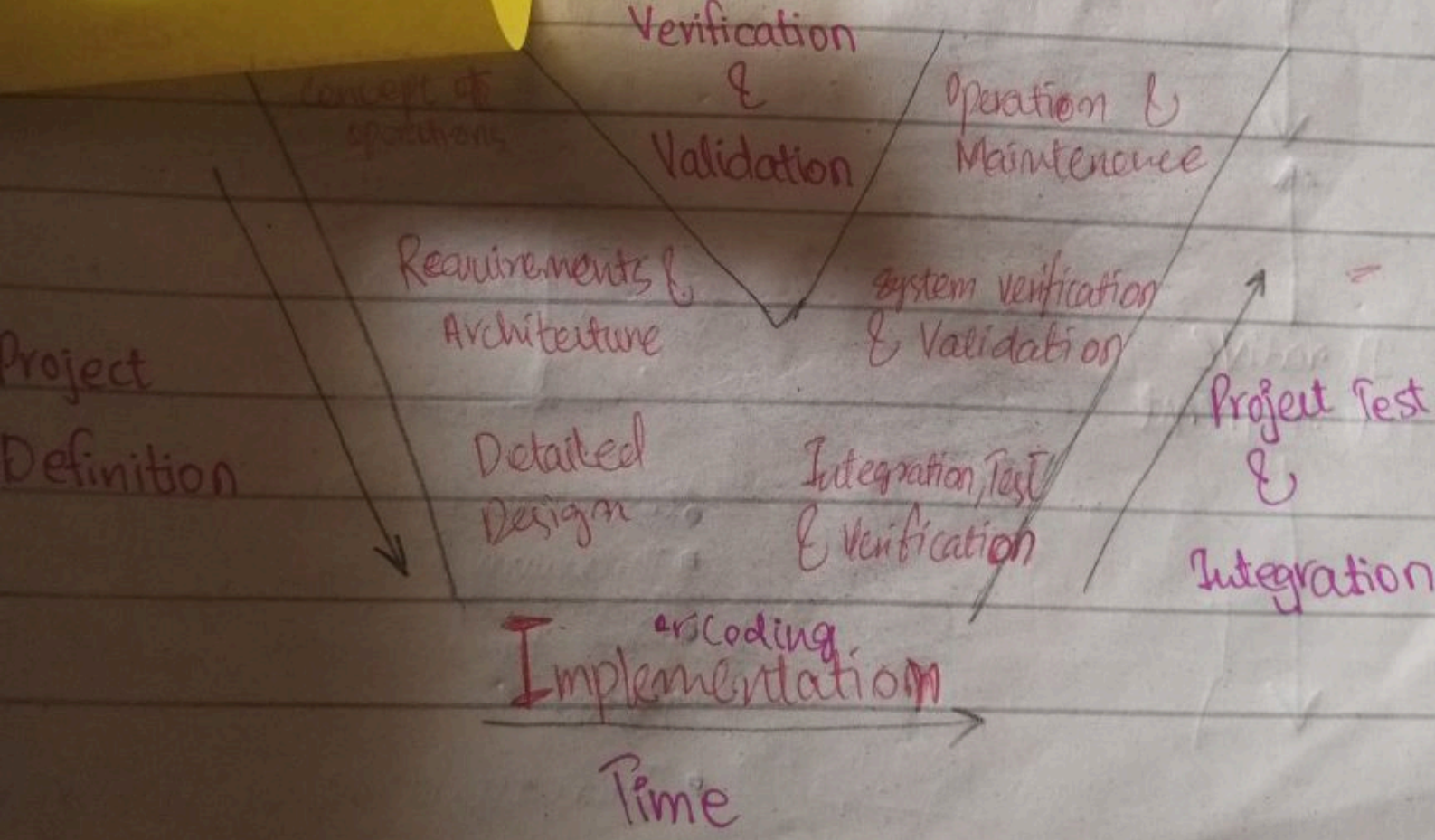
Verification phase (Requirement analysis, system Design, Architecture Design, Module Design).

full system + module, unit — sub-module designing

designing sub-module designing

(Unit testing, Integration testing , Acceptance testing). module-

client acceptance.



Advantages:

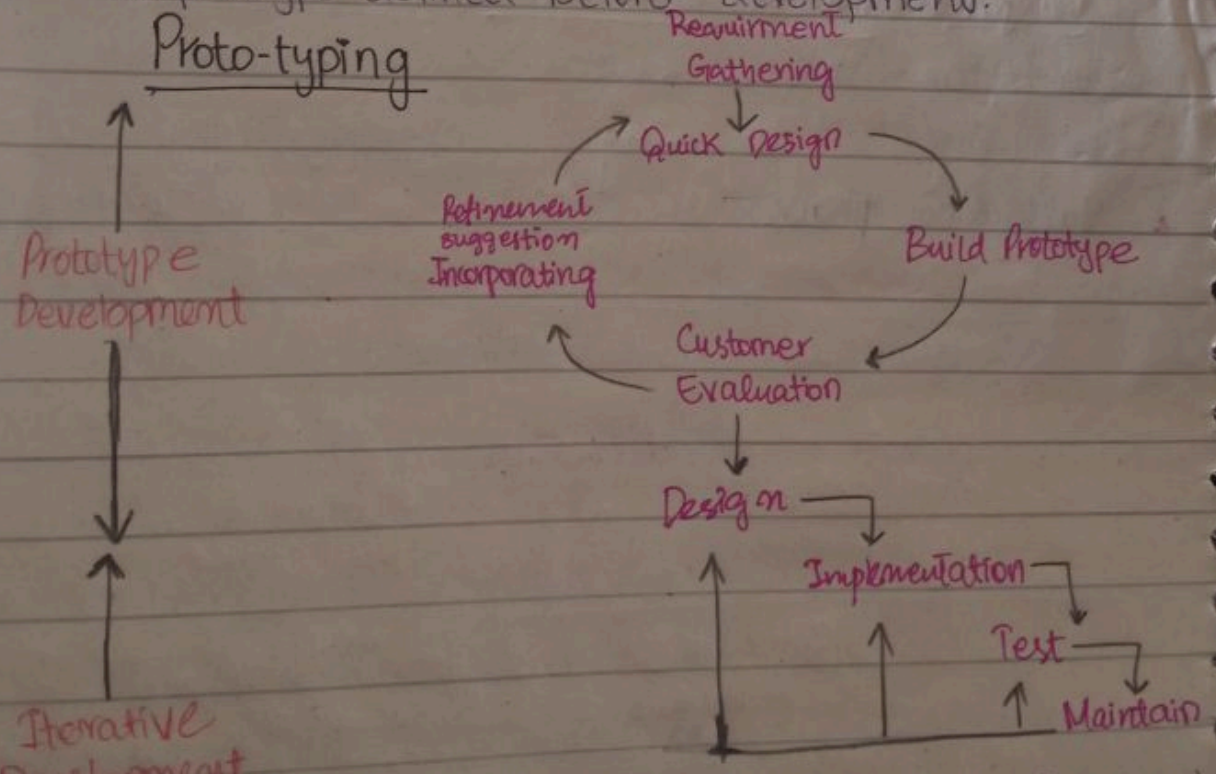
- Time Saving — as te
- Good understanding
  as after R-Analysis
- Every component sm
  Progress can be tra
- Proactive defect tra
- less complicated than waterfall / success over waterfall.

Where to use V—
- mostly used in small/medium sized organizations.
- model mostly used in Org where all requirements are easy together & understand by developer.
- Customer mostly use V-Mod for performance.
- chance of risk like there's no prototype.

Disadvantages

- No feedback, so less scope of changes.
- Risk analysis not done. — Evolution concept X
- Not good for big or object-oriented projects.
- no prototype defined before development.

Proto-typing



- Customer not clear with ideal (then this best mo)
- Throwaway Model (If any model not liked)
- Good for technical & requirement risks.

Disadvantage
- Increase in cost of Development.

Word Prototype.

↳ A kind of Dummy, toy model.

↳ Just a structure — if customer checks & say yes, we go to next step.

— First gather requirement.
— Quick Designing - rough model.
— Evaluate prototype to customer
— If customer give aueptence testing then we move to design → Implement → Test → Maintain

— If changes are reasuined in prototype it's renewed

Advantages

* Improve Software Design Quality
* Reduce software Development effort.

Types of Prototyping

Throwaway prototyping — also known as rapid or close-ended.

— In this we create a model that will eventually be discarded rather than becoming part of final delivered software.

Throwaway prototyping

When to use Incremental Model

- When major requirements are understood but some requirements may evolve with the passage of the time.
- When customer has no problem with the budget but he demands more & more Quality in SW.

Evolutionary Model

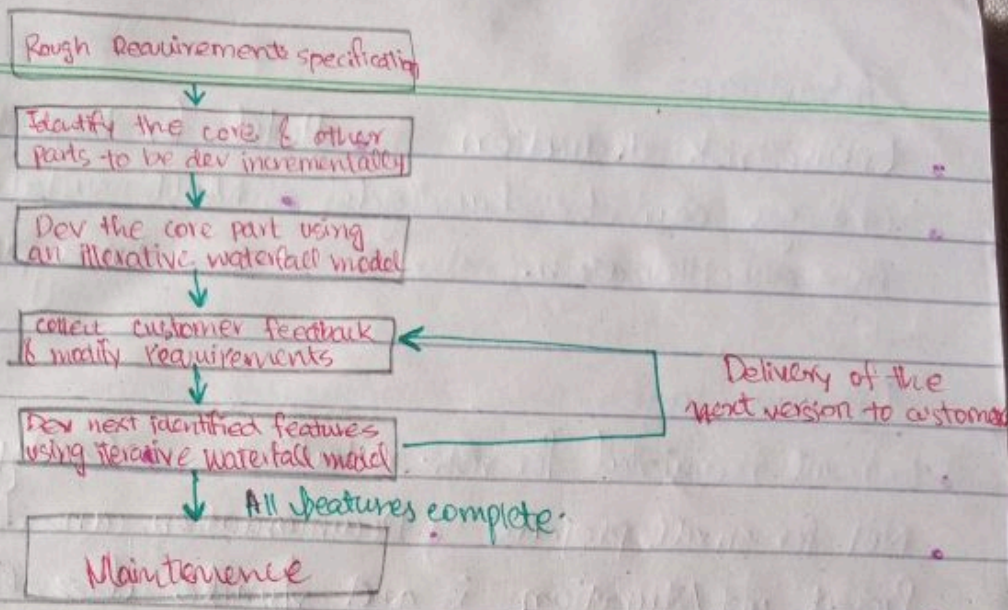
- Combination of 'iterative' & 'incremental' model of SDLC. — combination of these approaches.
- Requirements are initially gathered & remains same in other models but here requirements/ technology approaches may change so provide this flexibility and iterative model provides feedbacks — leads to requirement / or any phase modification. Also known as "Design a little, build a little, test a little, deploy a little model".

Advantages

- Customer requirement is clearly specified.
- Risk Analysis is better.
- Better suit for large mission-critical projects.
- feedback
- supports changing environment

Dis-advantages

- Cost affected
- Highly skilled resources required.



Spiral Model

→ The spiral model is a combination of waterfall, incremental, iterative and prototype model.

→ Mostly used for big prodjects where continuous changes required.

Phase → Planning

Activities
- Requirement gathering
- Requirement Analysis
- Requirement Specification
- Requirement Negotiation
- Requirement Modeling
- Requirement Management.

Deliverable – SRS requirment specification Doc (SRS)

Phase → Risk Analysis

Activities
- Risk Identification.
- Prioritization of Risk.
- Risk Reduction (Mitigation)
- Risk Impact Reduction
- Risk Monitoring.

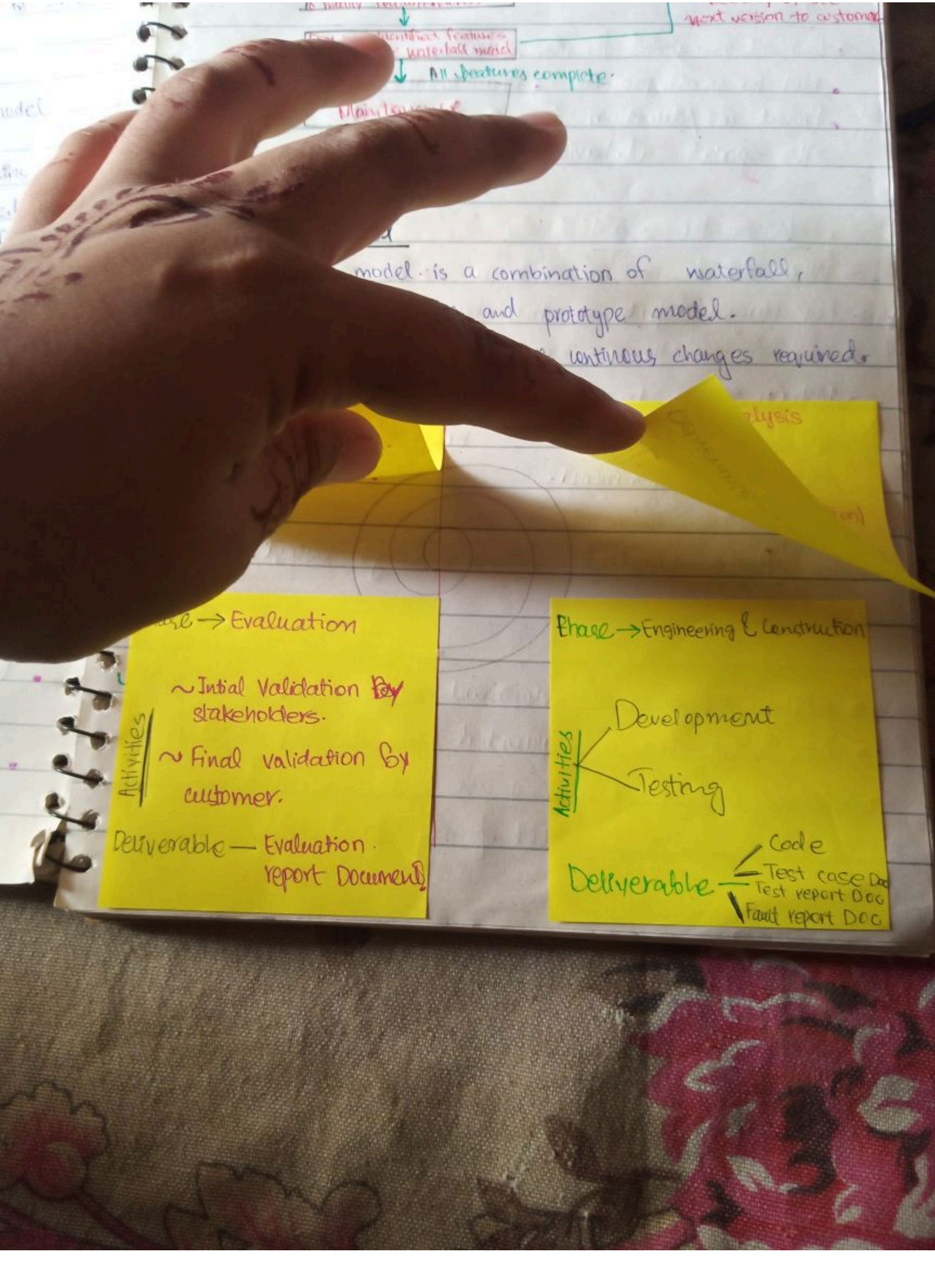
Deliverable → Risk Detailed Doc

Activities
- ~ Final validation by stakeholders.
- ~ Final validation by customer.

Deliverable — Evaluation report Document

Activities
- Development
- Testing

Deliverable
- Code
- Test case Doc
- Test report Doc
- Fault report Doc


All features complete.
model is a combination of waterfall,
and prototype model.
continous changes required.
Evaluation
~Intial Validation By stakeholders.
~Final validation by customer.
Activities
Deliverable — Evaluation report Document
Engineering & Construction
Development
Testing
Activities
Deliverable
Code
Test case Doc
Test report Doc
Fault report Doc

Advantages

- Ensure risk deduction
- Changes can be handled in next Iteration.
- Useful for large Projects.
- Most suitable for real-time projects.

Dis-advantages

- Expert required for risk reduction
- Costly
- Not for small projects
- Documentation can be large
- Project cost/duration is not finite bcz of its spiral behaviour.

When to use

→ When risk is medium or high → For real-time SW.
→ For large projects → When changes in SW can be expected at any time.
→ When SW is to be used in sensitive & dangerous environment. → When requirements are not clear.

~~Agile Model (More Quickly) Latest Tech.~~

RAD Model

→ Rapid application Development model.
→ based on linear sequential SW development process
→ concise Dev cycle, If Req are well described —
RAD enables team to built fully functional

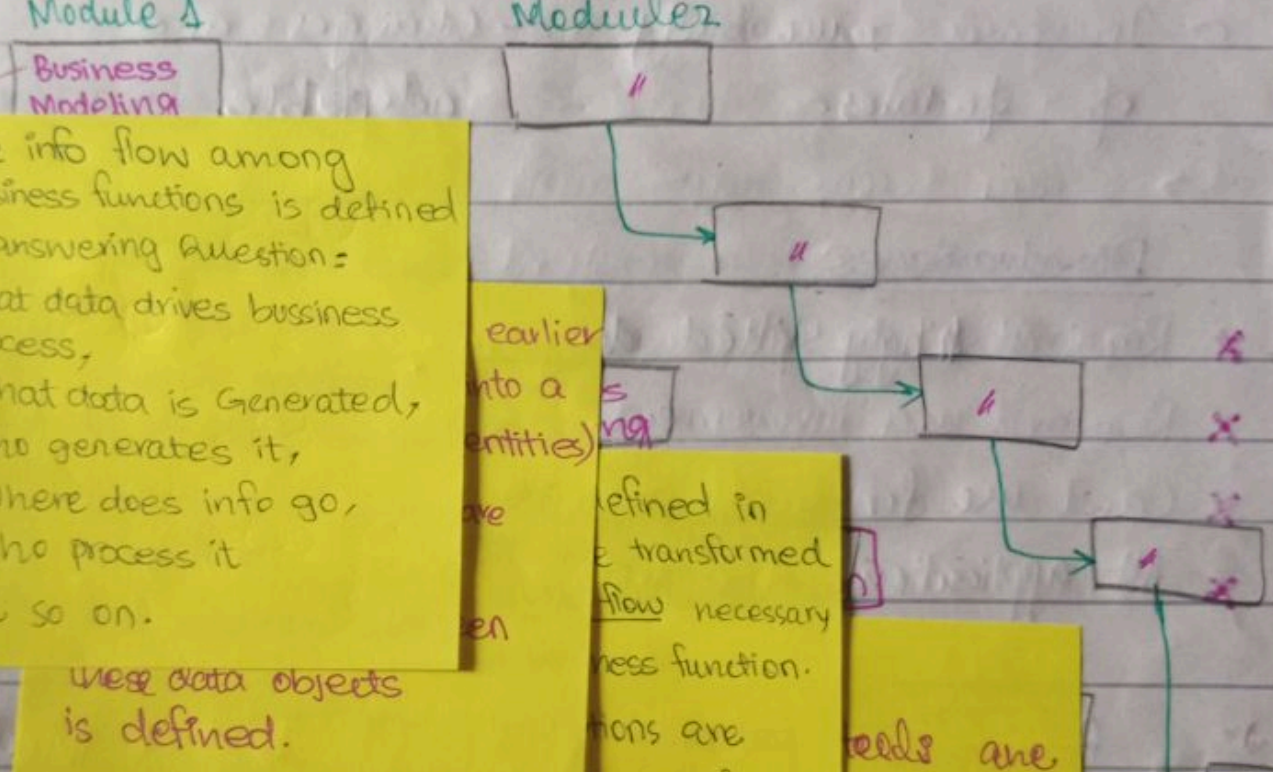
system within a concise time.

RAD — that products can be developed faster & of higher quality through:

~ Gathering Req using workshops

~ Prototyping and early, reiterative user testing of Design.

~ Re-use of SW components.



- The info flow among bussiness functions is defined by answering Question:
  - What data drives bussiness process,
  - What data is Generated,
  - Who generates it,
  - Where does info go,
  - Who process it

  & so on.

earlier
into a
entities)
ng

these data objects is defined.

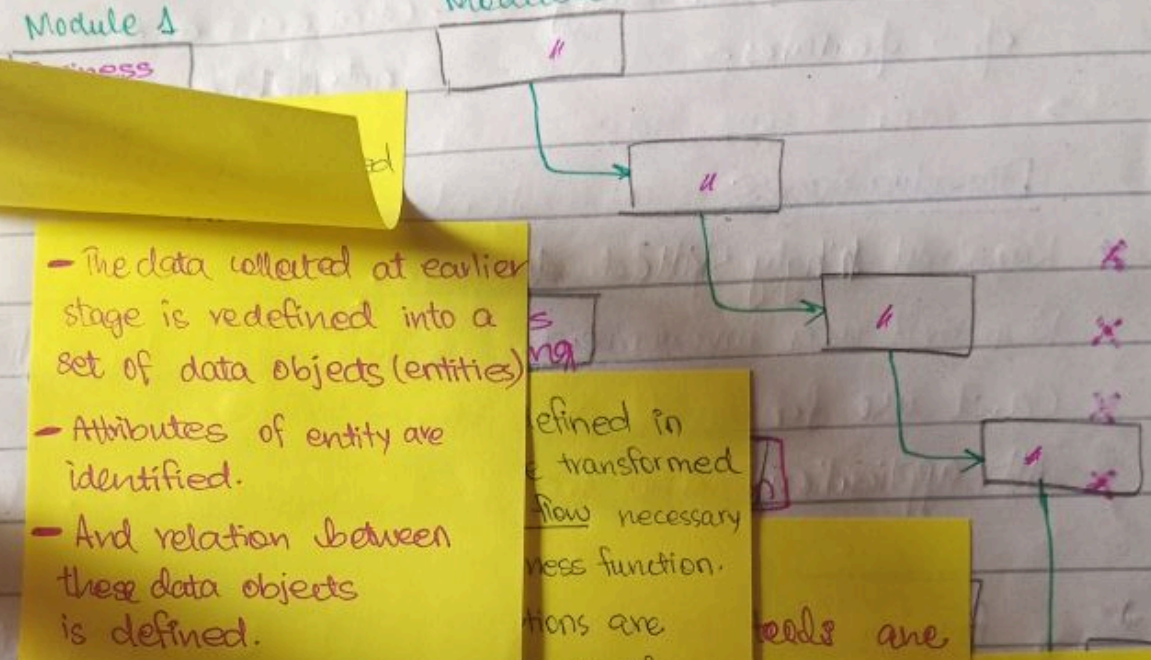
efined in
transformed
flow necessary
ness function.
ions are
, modifying,
retrieving a
data obj.

eeds are
e the
f SW;
the
ques.

programming
ve already
spike
.
verall

testing time.
- but new part must be tested, & all interfaces must be exercised

system within a concise time.
RAD — that products can be developed faster & of higher quality through :
~ Gathering Req using workshops
~ Prototyping and early, reiterative user testing of Design.
~ Re use of SW components.



- The data collected at earlier stage is redefined into a set of data objects (entities)
- Attributes of entity are identified.
- And relation between these data objects is defined.

efined in
transformed
flow necessary
ness function.
tions are
, modifying,
data obj.

eeds are
e the
f SW ;
the
iaves.

programming
ve already
share
overall
testing time.
- but new part must be tested, & all interfaces must be exercised

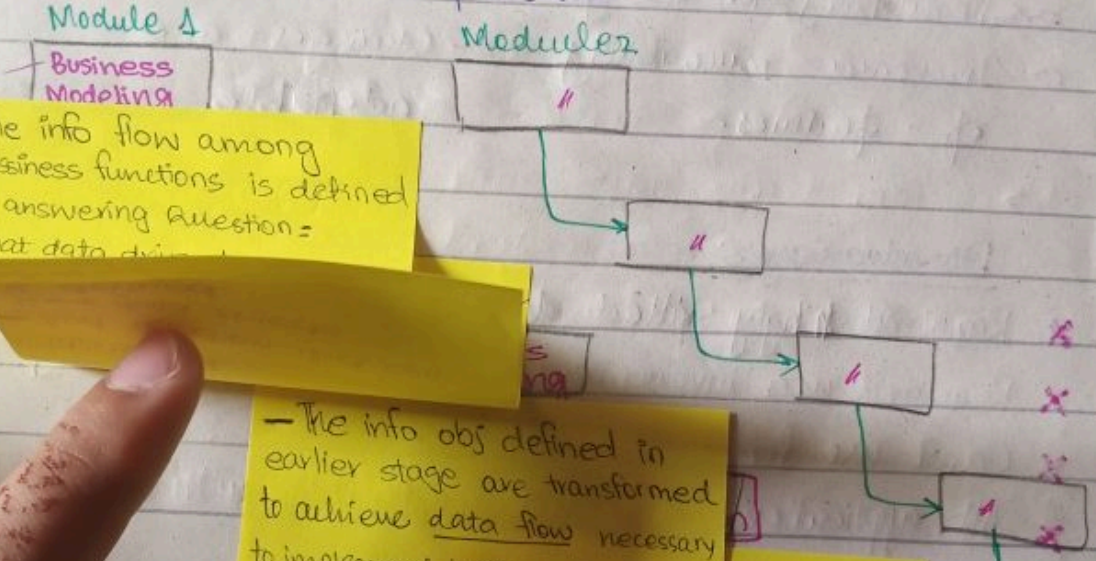
system within a concise time.

RAD — that products can be developed faster & of higher quality through:

~ Gathering Req using workshops

~ Prototyping and early, reiterative user testing of Design.

~ Re-use of SW components.



• The info flow among bussiness functions is defined by answering question:
— What data

— The info obj defined in earlier stage are transformed to achieve data flow necessary to implement bussiness function.

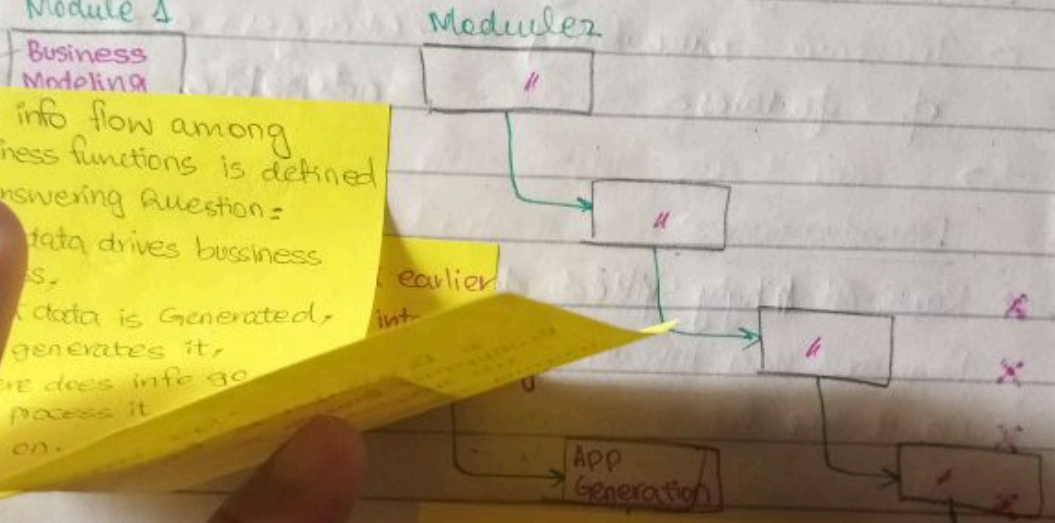
— Processing descriptions are created for adding, modifying, deleting or retrieving a data obj.

programming
ve already
structure
overall
testing time.
— but new part must be tested, & all interfaces must be exercised

system within a concise time.

RAD — that products can be developed faster & of higher quality through:

~ Gathering Req using workshops
~ Prototyping and early, reiterative user testing of design
~ Re-use of SW components.



- The info flow among bussiness functions is defined by answering Question:
- data drives bussiness
- data is Generated,
- generates it,
- are does info go
- process it
- on.

earlier

int

- Automated tools are used to facilitate the construction of SW; even they use the 4G GL techniques.

programming
we already
share
overall
testing time.
- but new part must be tested, & all interfaces must be exercised

system within a concise time.

RAD — that products can be developed faster & of higher quality through:

~ Gathering req using workshops

~ Prototyping and early, reiterative user testing of design

~ Re use of SW components.

Module 1

Business Modeling

Module 2

•The info flow among bussiness functions is defined by answering question:

—What data d... ussiness process,

...t dat... ...erated,

ge... ...t,

...ere... ...o go,

...earlier

...into

—Many of the programming components have already been ~~used~~ tested since RAD emphasis reuse.

— This reduces overall testing time.

— but new part must be tested, & all interfaces must be exercised

When to use RAD

~ When technical risk is limited
~ It should be used only if the budget allows the use of automatic code generating tools.

Advantages

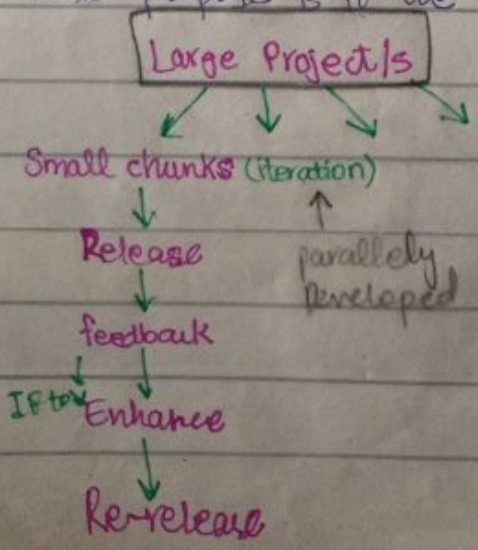
~ flexibility for change
~ Increases reuseability of features.
~ Reduce Dev Time
~ Changes are adaptable

Dis-advantages

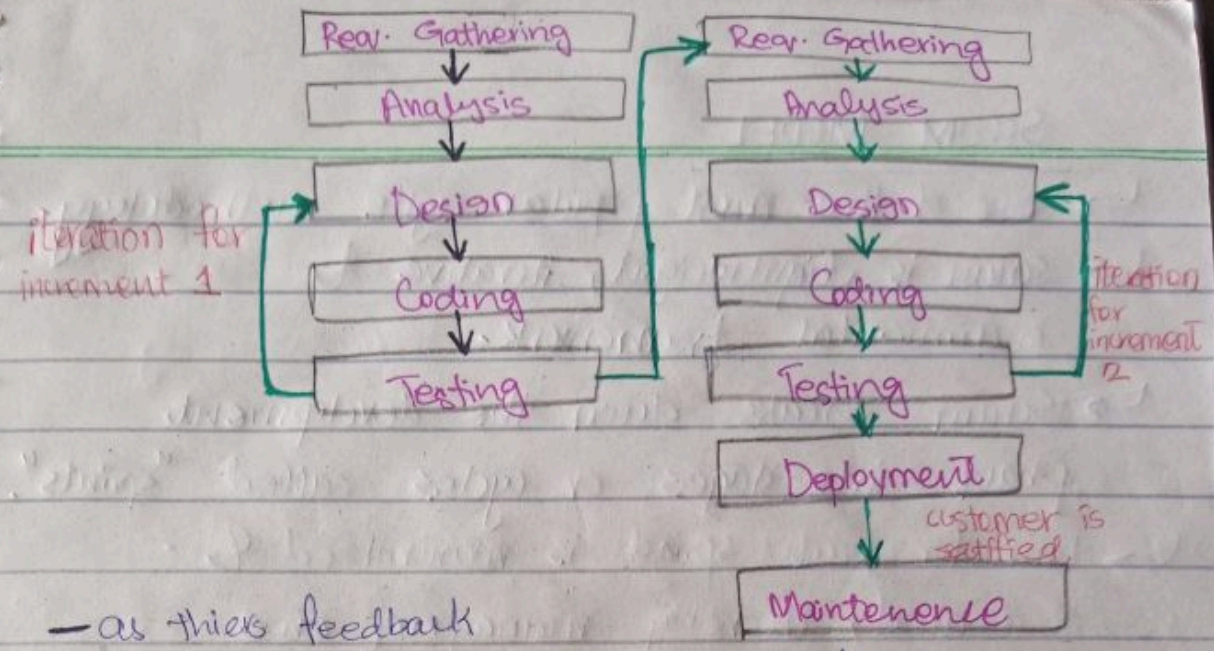
✖ Required highly skilled designers.
✖ Required user involvement.
✖ Not suitable for high risk.
✖ Can't use for small projects.
✖ All applications are not compatible with RAD.

Agile Model (More Quickly) Latest Model

→ Combination of both incremental & iterative models & promotes more involvement of customer in SW Dev.
→ Basic purpose is to be rapid in all activities.



• SW divided into small components.



- as thiers feedback
- so, if any changes are requried after testing, it goes back & modify.
- Each comp of SW is developed & validated by customer, if theres any problem goes under another iteration.

Advantages

- Freavent Delivery
- Freavent changes flexibility
- face to face communication with client.
- Rot-time affected

Dis-advantages

- Less documentation — focus on implementing product.
- Maintenance problem — when there's no proper documentation, and we modify the product after some time, the maintenance problems occur.

SCRUM MODEL

↳ one of the most popular agile methodology.
↳ Scrum is lightweight, iterative & incremental framework.
↳ Scrum breaks down the development phases into stages or cycles called "sprints".
↳ time for each sprint is maximized & dedicated, thereby managing one sprint at a time ie one week one sprint.
↳ Scrum team has scrum master & product owner with constant communications on the daily basis.

Baklog scrum:

Where the designers, stakeholders have defined the requirements about what product is about to be built?

Daily scrum: — all crew fully involved.

a daily small meeting on what to extend efforts are going on project?

Scrum master:

managing stakeholders, developers work-

Produd owner:

Client.

Advantages

↳ Freedom & Adaption.
  ↳ to everyone, as all crew is fully involved.
↳ High Quality – Low risk product.
↳ reduce dev time upto 40%.
↳ reviewing the current sprint before moving on to new one.

Dis-advantages

↳ More efficient for small team size.
  ↳ as we develop in small 'sprint' form.
↳ No changes in the sprint.
  ↳ but can renew in next one.

Comparison of All SDLC models

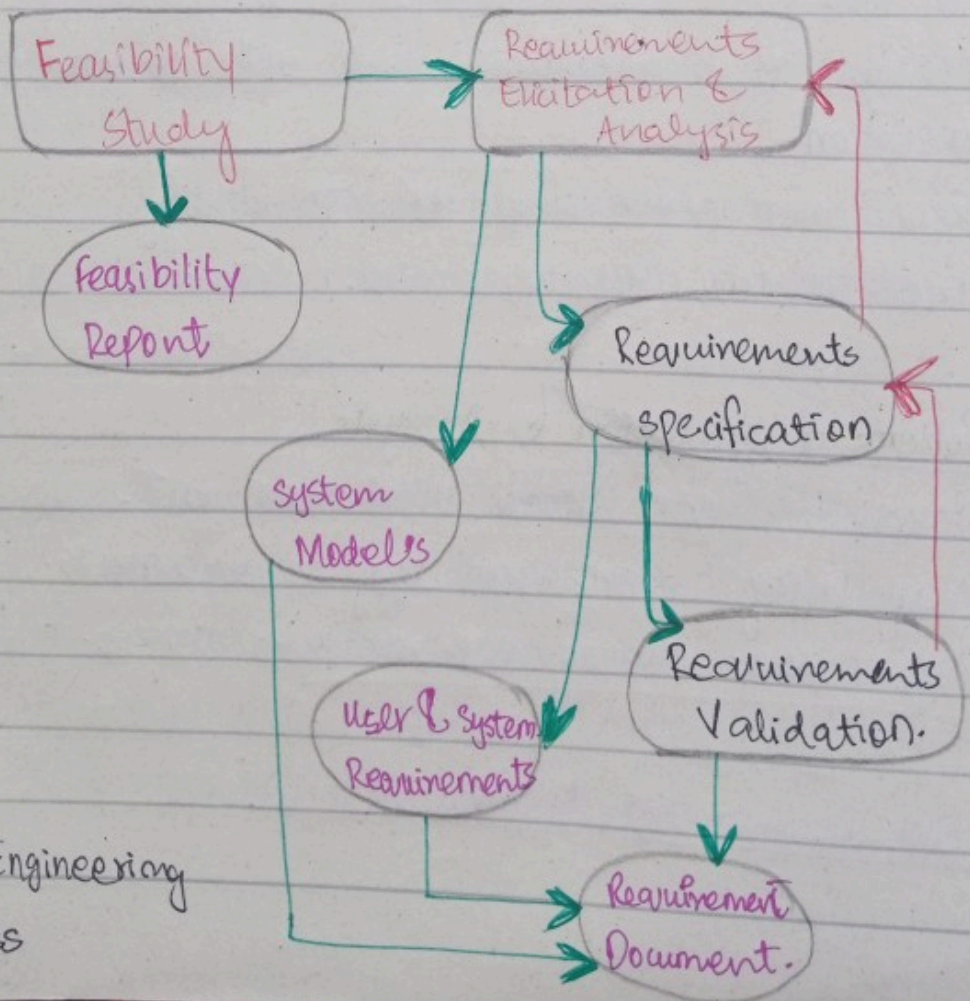
Classical Waterfall	Iterative Waterfal	Prototype Model	Incremental Model	Evolutionary Model	RAD Model	Spiral Model	Agile Model
Basic model,	Basic model,	User-requirment not clear— not confident.	Module by module delivery,	Large projects.	Time & cost constraint,	Risk-handling model	Flexible,
Rigid,	Problem is well understood	Costly,	Easy to test & Debug.	requirements are locked for first increment.	time given project.	Not for small projects,	Latest,
Inflexible,	Feedback is given to previous stage,	No early lock of requirements			cost		Parallelism
Not good for real-time projects		High user involvement			user interaction at all levels	No early look at requirements,	Process divided into sprints
No-parallelism		Reuseability	requirements are lock	requirements can be change in next increments	Reuseability	team can be less experience	divided into time given module
							equal interest of all crew

Software Requirements/-Engine

SW Requirement
→ SW-req. Engineering
→ SRS (Software Req.)
→ Req. Analysis
→ SW reuse Engg
→ DFD
→ Data Dictionary
→ E-R Diagram.
→ Structure Charts
→ System Modeling
Req. Elicitation Process — Pdf.
→ Properties of good SRS

↳ It's the description of ...ities of the targeted system
↳ It's the description of ... do.
• Requirement Enginee... process defining, documenting & maintaining requirements in the engineering design process.

It's a four step process;
- Feasibility study.
- Requirement Gathering & Elicitation.
- SW Requirement specification.
- SW Requirement Validation.



• Req. Engineering Process

♦ Feasibility Study — tells is system feasible - ...
↳ is system is ~~able to~~ deserve to be built or not.
↳ Objective is to create reasons for developing SW that is acceptable to user
↳ & flexible to change
↳ Comfortable to estab

Economical Feasibility

• Decides whether the necessary SW can generate financial profits for an organization.

♦ Requirement Gathering /E
- Gathering of requirements.
- Gathered from client/user.
- brainstorm whether the following requirements will be fullfilled or not.
- Analysis of requirements starts with requirement elicitation
Requirements are analyzed to identify inconsistencies, defects, omission etc.

Problems of Elicitation & Analysis

often
Stakeholders don't know what they want.
They may have conflicting requirements.
Req. change during the analysis process.

♦ SW Requirement Specification (SRS)

- We have gathered information/requirement.
- Completing brainstorming & requirement analysis.
- Conducted meetings/conferences with clients & stakholders.
- After doing all these steps we elaborate that we can perform such things & we write all concept/things in an elaboratic way called (SRS) — in technical language.

→ The models used at this stage include;

- DFD - Data Flow Diagrams — widely used for modeling requirements.
- It shows the flow of data through system.

comp. hardware sys company org sw system.

or any combination of preceeding.

- Also known — data flow graph or bubble chart.

Data Dictionaries — are simply repositories to store info. about all data items defined in DFD's.

→ At requirement stage — DD should atleast define customer data items.

→ to ensure both customer & Developers use same definition & terminologies.

ER- Entity-Relationship Diagram

→ another tool for requirement specification.
→ detailed logical representation of data for org.
→ uses 3 main constructs. — data entities, relationships, & thier associated attributes.

SW Req. Validation

- fully gathered info at early stages & written.
- Documentation completed.
- then requirments discussed in this documents are validated.
- User might demand illegal, impossible solutions or expert may misinterpret needs.

Requirements can be checked againts following conditions.

If they can pratically implement.
If there are ambiguities.
If they can describe & they are full.

Requirement Validation Techniques

Requirements reviews/inspections

— Systematic manual analysis of requirements

Software Requirements – What sort of requirements may arise in requirement elicitation phase —? What kind of req. are expected from SW system?

- **Functional Requirements**

They define functions & functionalities within & from SW system.

i.e;

- Search option given to user for search from various invoices.
- User should be able to mail any report to management.

- **Non-Functional Requirements**

– Not related to functional aspect of SW.
– They are implicit or expected characteristics of SW — user assumptions. may include;

- Security
- Storage
- Performance
- Cost
- Flexibility
- Disaster recovery.

Requirements are categorized logically;

ust have – SW can't be said operational without them.
ould have – Enhancing functionality of SW.
uld have – SW can still properly function with these req.
ish list – These req do not map to any objectives of SW.

User Interface requirements

- UI is an imp part of any software or hybrid system.

A SW is widely accepted if;

- Easy to operate
- Quick in response
- effectively handling operational errors
- providing simple yet consistent UI

A well-performing SW system must also be with attractive, clear, consistence & responsive. Otherwise, the functionalities of SW system be used in convenient way.

UI requirements are briefly mentioned,

- Content presentation
- Simple Interface
- Consistent UI elements
- Easy Navigati
- Responsive
- Feedback me

etc.

- **Prototyping**
  - Using an executable model of the system to check requirements.
- **Test-Case Generation**
  - Developing tests for requirements to check testability.
- **Automated consistency analysis**
  - checking for the consistency of structured requirements descriptions.

Prerequisite of SW requirements

Collection of SW requirements is the basis of the entire SW development project.

A complete SRS should be;

characteristics of good SRS

- Clear
- Correct
- Consistence
- Coherent.
- Comprehesiable
- Modifiable
- Unambigious
- Verifiable.
- Traceable
- Credible source.

Functional / Non-funtional Requirements

Requirements, which are related to funtional/ working aspect of SW fall into funtional Req. funtional requirements are describing the behaviour of the system as it correlates systems funtionalities.

Non-functional Req. are expected characteristics of target SW (security, ~~cost~~ flexibility, Disaster recovery, storage).

Execution Qualities

like security, useability, which are observeable at run time.

Evolution Qualities

like testability, maintability, extensibility, scalability that embodied in the static structure of SW system.

SRS & characteristics of good SRS

— SRS is a description of SW system to be developed.

before designing → built in Req. Specification phase.

— It lays out functional & non-functional requirements of SW to be developed.
It may include a set of use cases (not coding, not designing) that describe user interations that SW must provide to the user for perfect interaction.

SRS - Structure

1. Introduction

- **Purpose** — what is the purpose of development.
— what to develop? — How? — How will it work?
— How will it interface with client.
- **Intended Audience**
— What is the intended Audience/ user of SW.
i.e Uni student SW system ← here the students / Teachers are the intended Audience, students can check thier attendence / marks. and also teacher can mark them etc.
- **Scope**
— what are the benefits / objectives of system in future.
- **Definitions**
— Define who is the director, principal, Head etc — in Uni system.
- **References**
— references are put.
— ~~Who~~ a developer must have inspired by some work and referenced his project to that project & can also modify / built its scope to evalute functionalities than the refrenced system in near future.
— Online/offline media references.

— How much work has done on that specific project — ? what I'm modifying — ? what will be the intended audience. — ? scope in future.

2. Overall Description

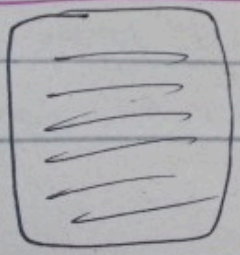
- **User Interface** — how student / Teacher interact with system.
  — their access to specific info.
  — Mention all acess & functionalities of how user / system will interact.
- **System Interfaces / Hardware Interface.**
  — Mention here, what protocols / servers we are using.
- **Constraints, Assumptions, Dependencies**
  i.e student can see the website / enroll in the system only when his age is — ? age constraint.
- **User Characteristics**
  — i.e users — Teacher / librarian / student / HOD / principal.

3. Features & Requirements

If have to do this then in SRS.

- Functional Requirements
  - How App/SW will work.
  - i.e first register → then login. etc.
- Use Cases.
- External Interface Requirements
  - third party App Requirements.
  - i.e - in Uni system, we can add that student can submit his fee (using paypal) ← third party.
- Logical Database Requirement
  - Structured Data, Un-structured Data.
  - Language → MySQL, etc.
  - Can also mention where ~~would~~ data would be saved.
- Non-functional Requirements
  - storage — How much RAM you need?
  - which resources are needed?
  - Safety/Security — Availability
  - Portability.

4. Delivery for Approval



- once completed documented — SRS.
- then send for approval.
- for next stage — designing —

User Requirements

SW Reuse Engineering

- Developing the SW by using the existing SW components.

Some of components;

- Source code
- Design & Interfaces
- User Manuals
- SW Documentation
- SRS & many more.

Advantages of SW reuse

→ **Less Effort** — less effort as many components are ready made.

→ **Time Saving** — Re-using the ready-made components is time saving.

→ **Reduce Cost** — less effort, time leads to overall cost reduction
— Increases SW productivity

→ **Utilize fewer resources**
— Re-useability saves many sources i.e effort, time, money etc.

DFD – Data Flow Diagram

- A graphical, visual representation/tool of info, useful for communicating with users, managers & other personnel.
- also known as Bubble chart.
- shows how data enters & leaves the system,
- what changes the info, where data is stored.
- Useful for analyzing existing as well as proposed systems.

Imp. observations about DFD

- All names must be unique, this makes refering elements easier.
- Remember DFD isn't a flow chart, arrows in DFD represents flowing data not order of events. doesn't involve order of event.
- A diamond-shaped is used to represent decision points with multiple exit point out of which only one is taken. ← implies order of event which makes no sense in DFD.

Why DFD?

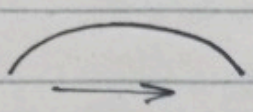
provides an Overview of:

↳ what data a system processes.

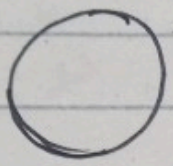
↳ what data is stored

↳ what results are produced.

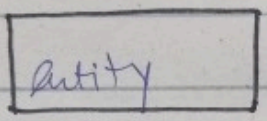
DFD Elements



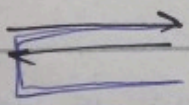
Data Flow — curved line shows the flow of data into or out of a process or data store.



Process — A circle shows a process that transforms data inputs into data outputs.



Source/Sink (External Entity) — acts as a source of system inputs or sink of system outputs.



Data Stores — set of parallel lines shows a place for collection of data items. The data store can have element/s.

External Entities — rectangle represents external entity.

- They either supply or recieve data.
- They don't process data.

Source — entity that supplies data to system.

Sink — entity that recieves data from system.

Rules of DFD

↳ Data can flow from

- External entity to process
- Process to external entity.
- Process to store & back.
- Process to process.

↳ Data can't flow from

- External entity to external entity.
- External entity to store. — direct DB entry with some process
- Store to external entity.
- Store to store.

DB — DB

X

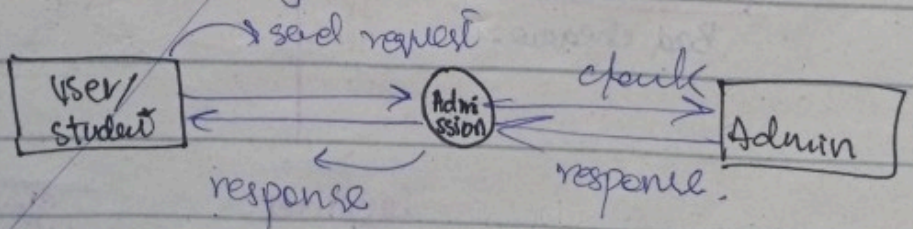
not direct but with some process.

Levels of DFD

DFD may be used to perform a system or SW at any level of abstraction.

0-level DFD

also known as fundamental system model. or context diagram.



Level-1, level 2

Logical & Physical DFD

↳ DFD's considered so far, are called logical DFD's logical — actually what are we doing.

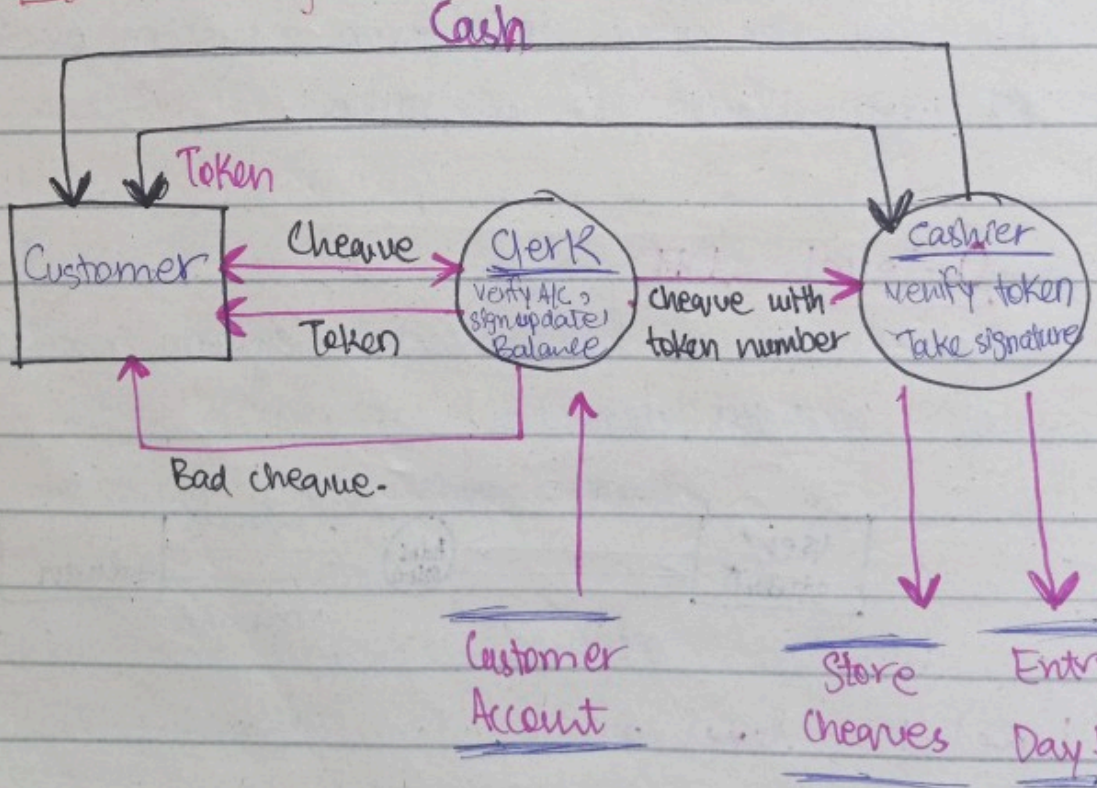
↳ A physical DFD is similar to a document flow diagram — Implementation.

↳ It specifies who does the operations specified by the logical DFD.

↳ Physical DFD may depict physical movements of the goods.

↳ physical DFD's can be drawn during fact gathering phase of a life cycle.

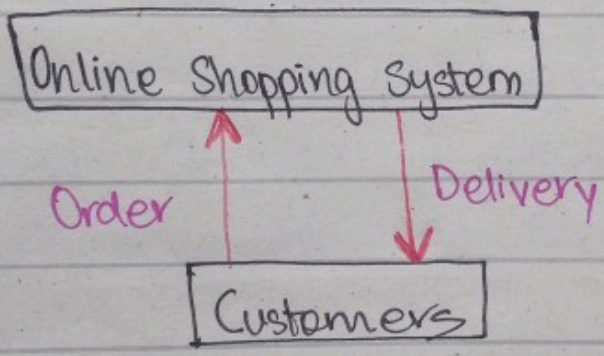
physical DFD for cheque Encashment



Levels of DFD

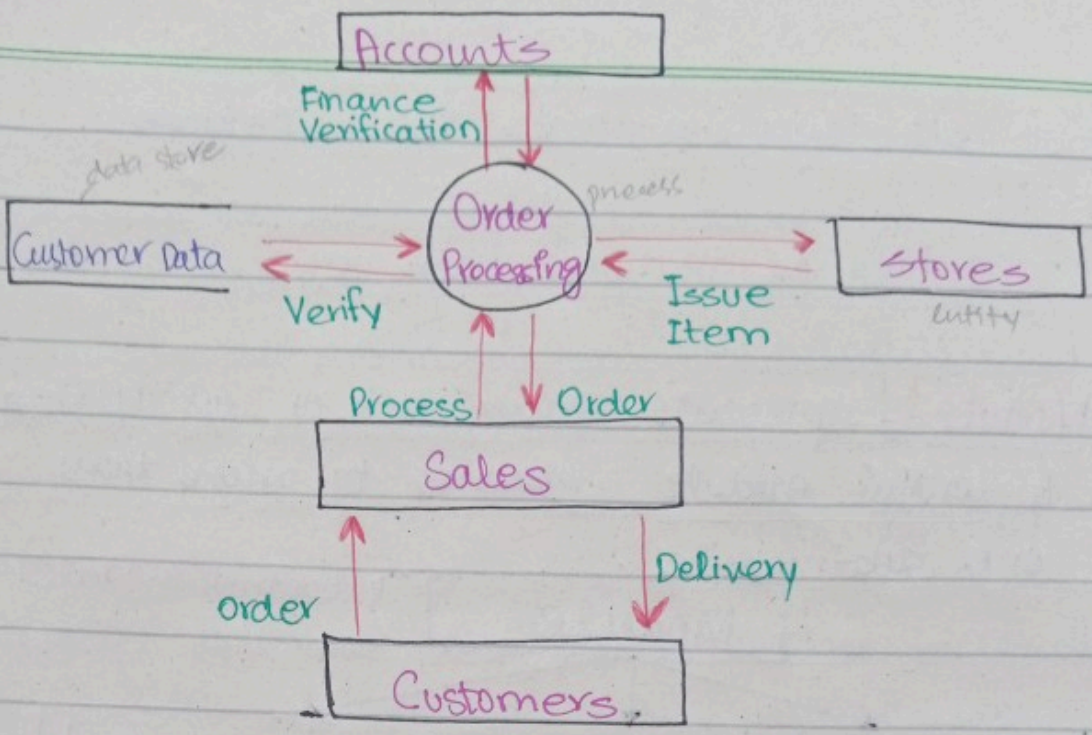
Level-0

- Highest abstraction level DFD is known as level-0 DFD.
- which depicts the entire information system as one diagram concealing all the underlying details.
- level-0 DFDs also known as content level DFDs.



Level-1

- Level-0 DFDs is broken down into more specific, level 1 DFD.
- It depicts basic modules in the system & Flow of data among various modules.
- level-1 DFD also mentions basic processes & sources of information.



Level-2 DFD

- At this level, DFD shows how data flows inside the modules mentioned in level-1.
- Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding upless the desired level of specification is achieved.

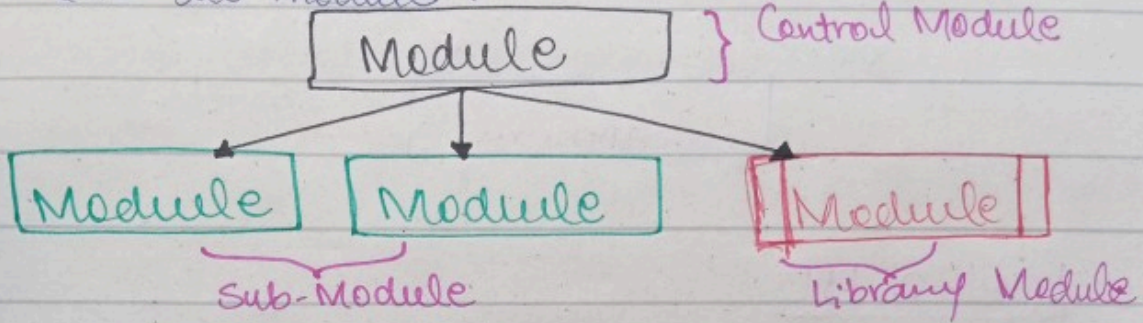
Structure Charts – a chart derived from DFD.

- It represents the system in more detail than DFD.
- It breaks down the entire system into lowest functional modules, describes functions & sub-functions of each module of system to a greater detail than DFD.
- It represent hierarchical structure of modules.

- At each layer, a specific task is performed.

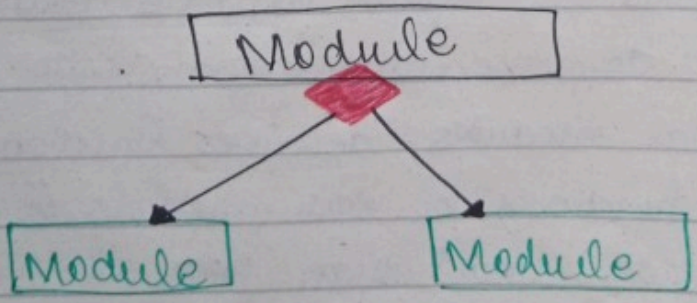
- Symbols in structural charts are;

Module – Represents process, task or sub-routine.
- A central module branches to more than one sub-module.

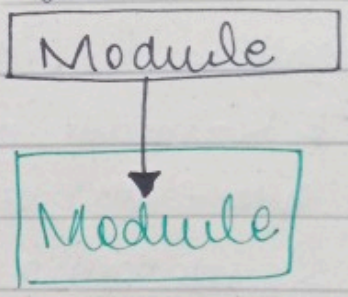


- Library modules are re-useable & invokable from any module.

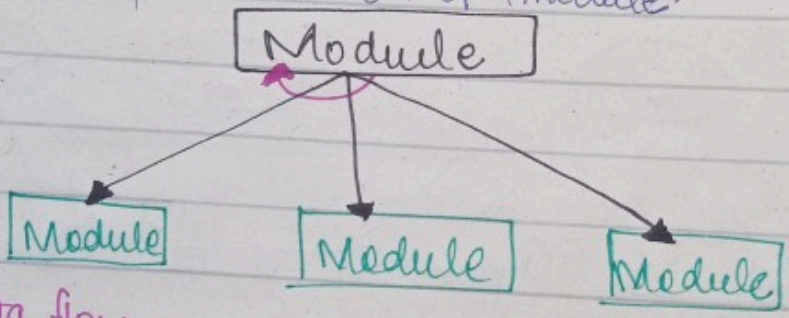
Condition – represented by a small diamond at the base of module.
- It depicts the control module can select any sub-routine based on some condition.



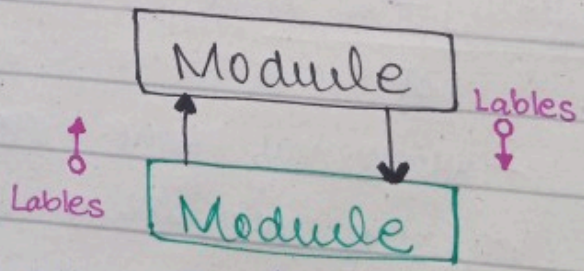
Jump – An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.



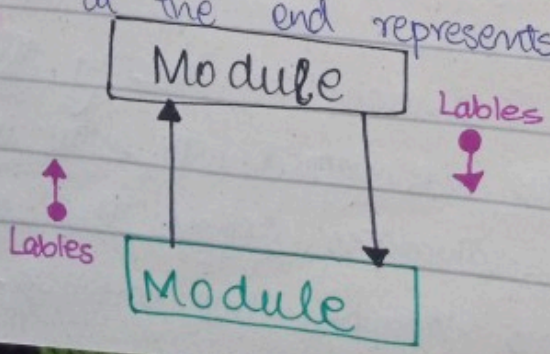
Loop – A curved arrow represents loop in middle.
– All sub-modules covered by the loop repeat execution of module.



Data flow
– A directed arrow with empty circle at the end represents data flow.



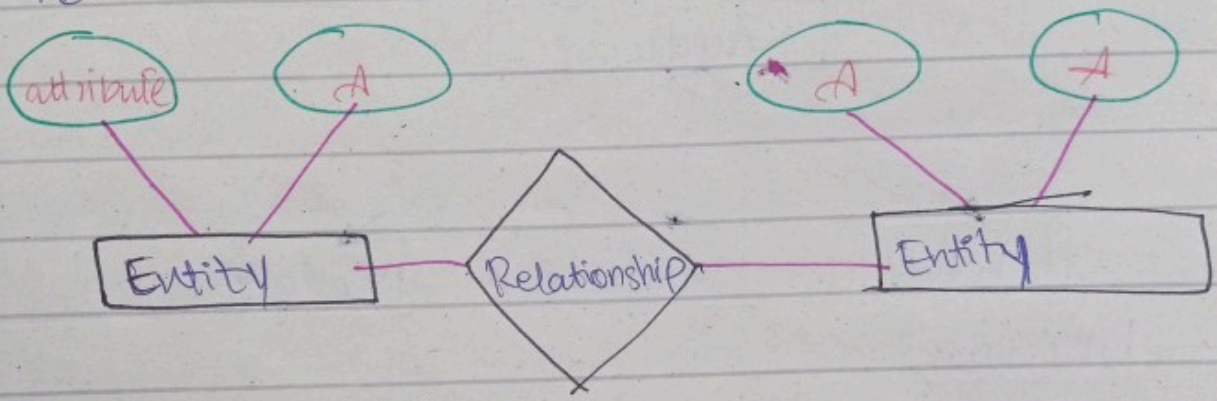
Control Flow – A directed arrow with filled circle at the end represents control flow.



ER Model

- Entity-Relationship model is a type of database model based on the notion of real world entities & relationships among them.
- E-R model creates a set of entities with thier attributes, a set of constraints & relationship among them.
- E-R model is best used for the conceptual design of Database.

i.e



Entity — An entity in model is a real world obj
- which has some properties called attributes.
- Every attribute is defined by its corresponding set of values, — called domain.

Relationship — the logical association among the entities.
- relationships are mapped with entity in many ways.
- Mapping cardinalities define the number of associations between two entities.

Mapping Cardinalities :

- one to one.
- one to many.
- many to one.
- many to many.

Degree of Relationship

- Unary
- Binary
- Ternary

...

Cardinality

One to one: - One entity from entity set A can be contained with at most one entity of entity set B & vice versa.

- **One to many:** - When single instance of an entity is associated with more than one instances of another entity.

- **Many to many:** - More than one entity from entity set A can be associated with at most one entity of entity set B & vice versa.

- **Many to Many:** - One entity from A can be associated with more than one entity of B & vice versa.

- Keys = main

- Types of Attributes

Data Dictionary

- it's the centralized collection of info about data.
- It's stores meaning & origin of data & relationship with other data, data format for usage etc.
- Data-Dictionary is often referenced as meta-data repository.
- it's created along with DFD model of SW program & expected to be updated whenever DFD is changed or updated.

Data-Dictiory, often includes info about:

- Name of Data item — " " is self-explanatory.
- Aliases — include other names by which this data item is called DEO (for Data Entry Operator) & DR (Deputy Register)
- Description/Purpose — is a textual description of what the data item is used for or why it exists.
- Related data items — capture relationships between data items. e.g, total-marks must always equal to internal-marks plus external-marks.
- Range of Values — records all possible values, e.g - total marks must be positive.

- Data Structure Forms — capture the name of processes that generate or recieve the data items.

Requirement of Data Dictionary

- The data is referenced via data dictionary while designing & implementing SW.
- Data dictionary provides a way of documentation for the complete database system in one place. Validation of DFD is carried via using DD.

Content

It should contain info about following:

- Data Flow — DF is described by means of DFDs, & represented in algebraic form.

=	composed of
{}	Repitition
()	Optional
+	And
[/]	Or

i.e

Address = House No + (Street/Area) + City + State

Course ID = Course No + Course Name + Course Level + Course Grades.

- **Data Elements** – consists of name & descriptions of Data & Control items, internal or external data stores etc – with following detail.
  - Primary Name
  - Secondary Name (Alias)
  - Use-case (How & where to use)
  - Content Description (Notation, etc)
  - Supplementary Info (present values, constraints)

- **Data Store** – store the info from where the data enters the system & exits out of system may include
  - **Files**
    - Internal to SW.
    - External to SW but on same machine.
    - External to SW & system, located on Different machine.
  - **Tables**
    - Naming Convention.
    - Indexing property.

- **Data Processing** – There are 2 types
  - **Logical** – As user sees it.
  - **Physical** – As SW sees it.

SRS - part - Req. Engg

Properties of good SRS doc

→ **Consise** - SRS doc should be concise along with must be unambiguous, consistence & complete.
- irrelevant descriptions reduces readability & also increases error possibilities.

→ **Structured** - should be well-structured, it's easy to understand & modify.

→ **Black-box view**
- SRS doc should define the external behaviour of the system & not discuss implementation issues.
- SRS doc should view system to be developed as a black-box & should define externally visible behaviour of system.
- For this, SRS also known black-box specification of system.

→ **Conceptual Integrity**
- should show CI so that reader can merely understand.
- should characterized acceptable responses to unwanted events - called sys response to exceptional conditions.

→ **Verifiable**
- All requirements of system, as documented in SRS Doc, should be correct.

Project Management

→ SPM.
→ Project Manager.
→ Components of SPM
→ Need of SPM
→ SPM activities
→ Project estimation Tech.
→ SPM tools

enable a group of deve
towards successful

SW Project Management

- SPM is management of SW producing activities in such a way that SW should be created within time, within budget & with less effort.
  - planning & leading SW projects.
- Main goal is to enable a group of developers to work effectively towards successful completion of project.

Prerequisite of SPM - There are 3 needs for SPM;
- Time - Cost - Quality

- It's essential for SW organization to deliver a quality product, keeping the cost within the clients budget & deliver project as per schedule.

Project Manager - a character who has the overall responsibility for the planning, design, execution, monitoring, controlling

& closure of project/s.
- responsible for giving decisions for both large & small project/s.
- he lead his team to make them understand what is expected from all of them.
- he's a medium b/w his clients & his team. & report to senior management.

Responsibilities of PM

- Managing risks & tasks
- Activity planning & sequencing.
- Create project team & assign tasks to them.
- Monitoring & reporting process.

Components of SPM

- Planning — PM along with his team plan all the activities that will or can be involved in SW production.
- Organizing — In here, project team is decided, organized & get assigned responsibilities.
- Staffing — SW team is selected for doing the assign tasks & training of team also is paid.
- Leading — ability to work with team, to get desired work from them, motivate them in a way that can be fruitful & SW should be produced within time & budget.

Controlling is an activity in which we control all the things involved in SW production.

Controlling activities if activities take more time than assign time.

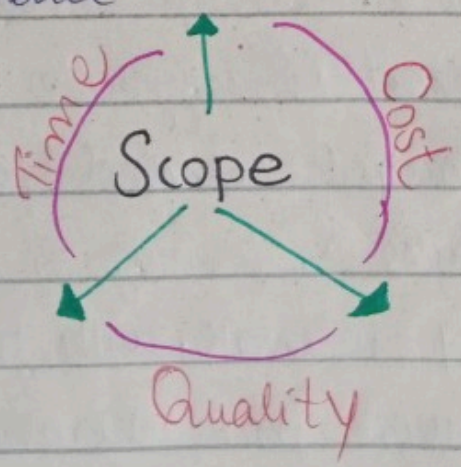
Controlling risk by risk assessment.

Controlling faults & errors by SW testing etc.

Need of SPM – SW is said to be an intangible product.

Most SW are tailor made to fit client's requirements.

The most important is that, the underlying technology changes/advances rapid and frequently that experience of one product may not be applied to the other one. All such bussiness/environmental constraints bring risk in SW Dev hence it is essential to manage SW projects efficiently. with old experience & knowledge



It shows triple constraints of SW projects.

- It's essential part of Organization to deliver Quality product, keeping the cost within client's budget constraint & deliver the project as per scheduled.
- Therefore, SPM is essential to incorporate user requirements along with budget & time constraints.

SPM - Activities - It may

- **Project Planning** - a task
  before the production of
  - It's a set of processess, u
  SW production.

Project Planning
→ SW project Planning
→ Cost Estimation
→ Tech (Locomo, Putnam)

- **Scope Management** - It defines the scope of the project; this includes all activities/ processess need to be done in order to make desired/deliverable product.

user requirements along u
& time constraints.

SPM - Activities — It may include

- **Project Planning** — a task which is performed before the production of SW actually starts.
  — It's a set of processess, which failitates SW production.

- **Scope Management** — It defines the scope of the project; this includes all activities/ processess need to be done in order to make desired/deliverable product.
  — it's essential bcz it creates boundries of projects by clearly defining what would be done in project & what wouldn
  During this phase, it's necessary to —
  - Define the Scope.
  - Describe it's verification & control.

- Verify the scope.

- Project Estimation — With correct estimation managers can manage & control project more efficiently & effectively.

- SW size Estimation — Software size may be estimated either in terms of KLOC (Kilo line of code) or by calculating number of function points in SW. — LOC depends upon coding practices & FP depends on user or SW requirements.
- Effort Estimation — Managers estimate efforts in terms of personnel requirements & man-hour required to produce the SW. For EE, SW size should be known.
- Time Estimation — Once SE and EE done, time required to produce SW can be estimated.
  - SW tasks are divided into smaller tasks, & are scheduled on day-to-day basis — calender months.
  - The sum of time required to complete all tasks in hours or days is total time invested.
- Cost - Estimation — most difficult to all.
  - required to consider.
    - Size of SW
    - SW Quality

- Hardware
- Addition s/w/ tool, licenses etc ..
- skilled personnel
- Travel involved
- Communication
- Training & support.

Project Estimation Techniques

- We discussed various parameters like time, cost etc.
- PM can estimate these factor using 2 broadly recognized techniques :-

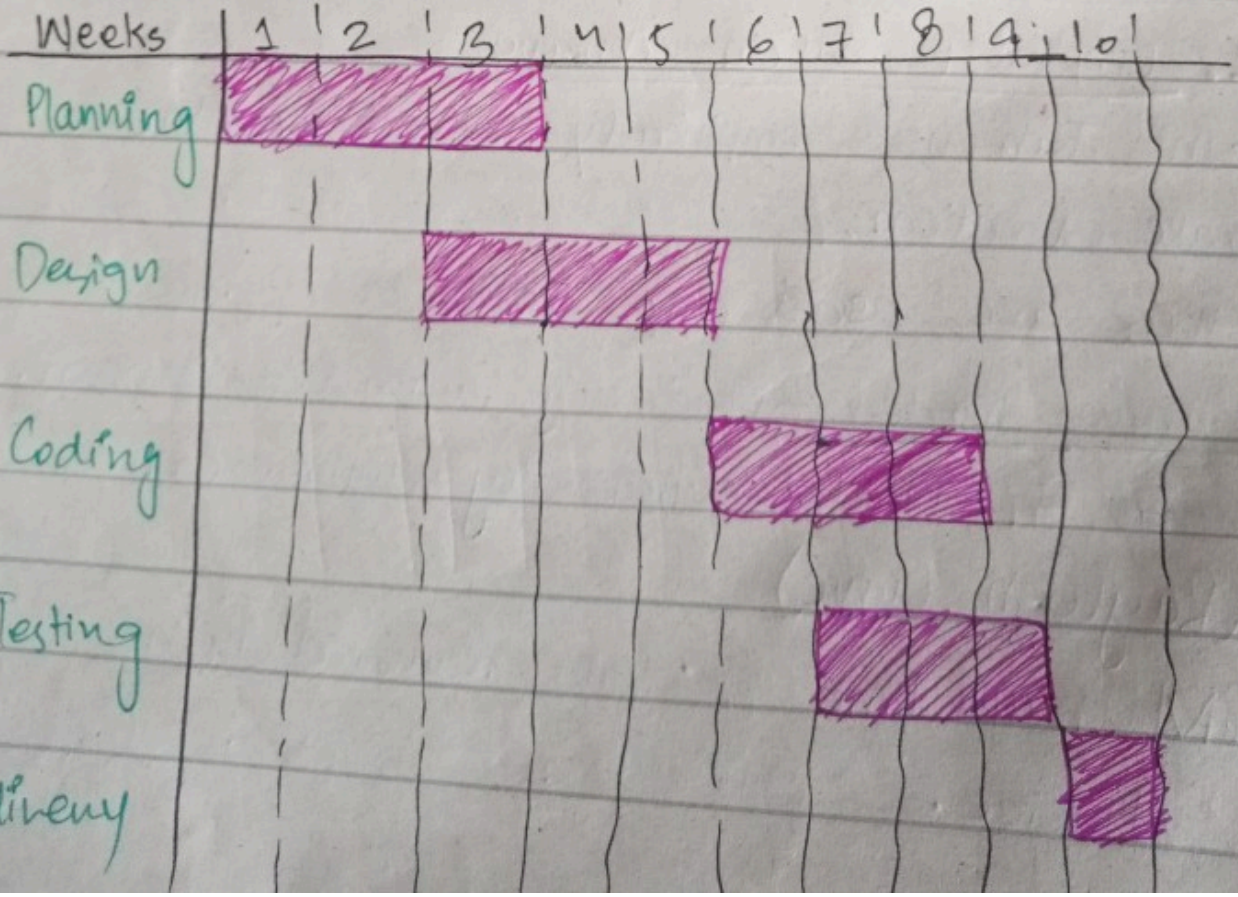
1. **Decomposition Technique** – this techniques assumes the SW as a product of various compositions.
   There are 2 main models.
   - **LOC (line of Code)** – Estimation is done on behalf of number of line of codes in SW product.
   - **Function Points** – Estimation is done on behalf of number of FPs in the SW product.
2. **Empirical Estimation Technique**
   - This Tech uses empirically derived formulae to make estimation.
   - These are based on LOC or FPs.
   - **Putnam Model** – Made by Lawrence H. Putnam.
   - based on Norden's frequency distribution (Rayleigh Curve)
   - PuM maps time & efforts required with SW size.

- **COCOMO** — stands for COnstructive COst MOdel,
  — developed by Barry W. Boehm.
  — It divides SW product into 3 categories of software: organic, semi-detached & embedded.

Project Managment Tools

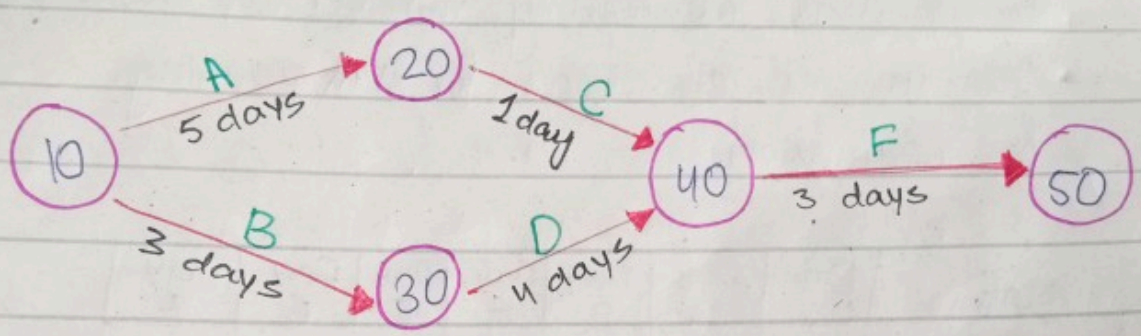
The risk & uncertainty rises ~~over~~ multifold with respect to size of the project, even when the project is developed according to set methodologies. There are effective tools available.

- **Gantt chart** — Devised by Henry Gantt (1917)
  — It represents project schedule with respect to time
  — It's horizontal bar chart — with bars representing activities & time scheduled for project activities.



Pert Chart (Program Evaluation & Review Technique)

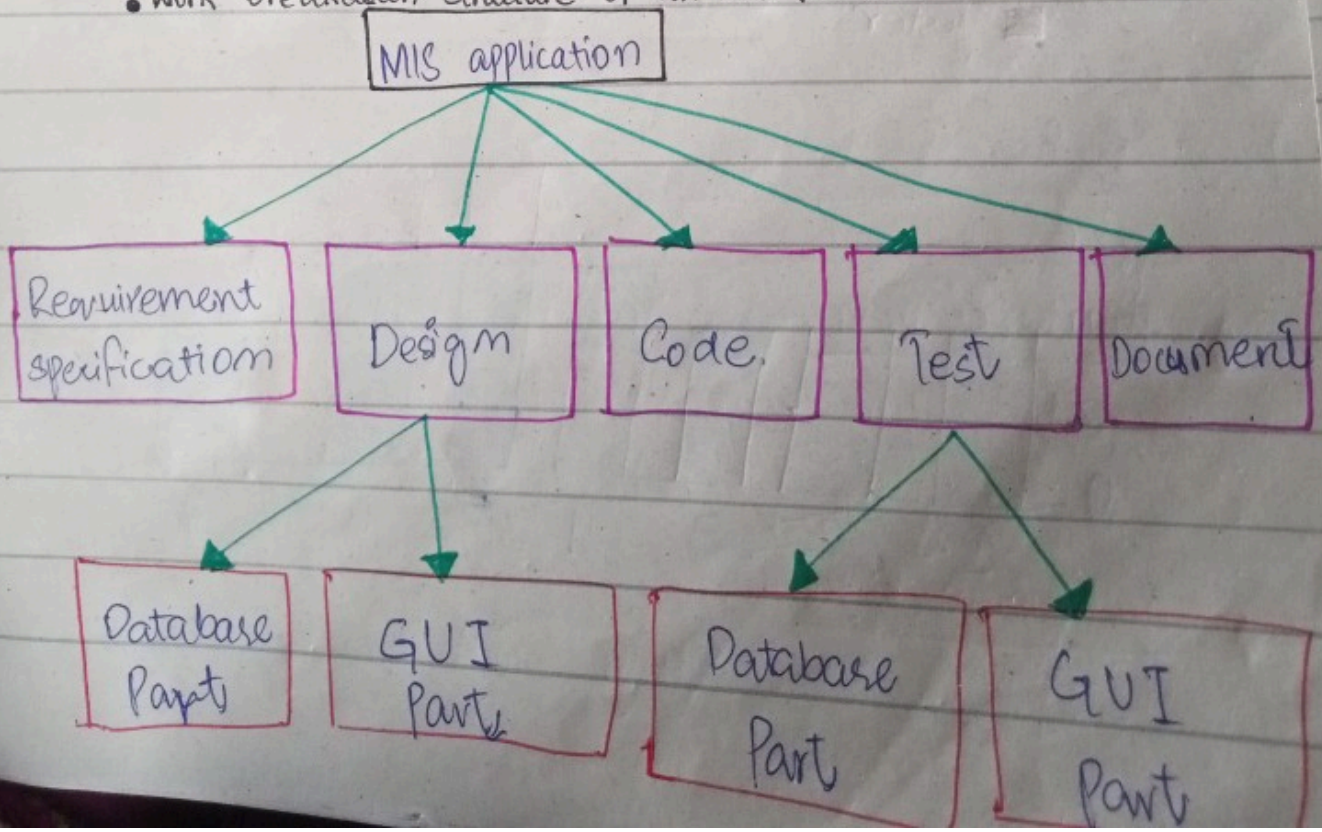
- a tool that depicts project as a network diagram.
- It represents main events of projects in both parallel & consective way.
- Events that occur one after another, show dependency of the later event over previous one.



- Events are shown as numbered nodes.
- They are connected by labled arrows depicting sequence of tasks in project.

Product breakdown Structure

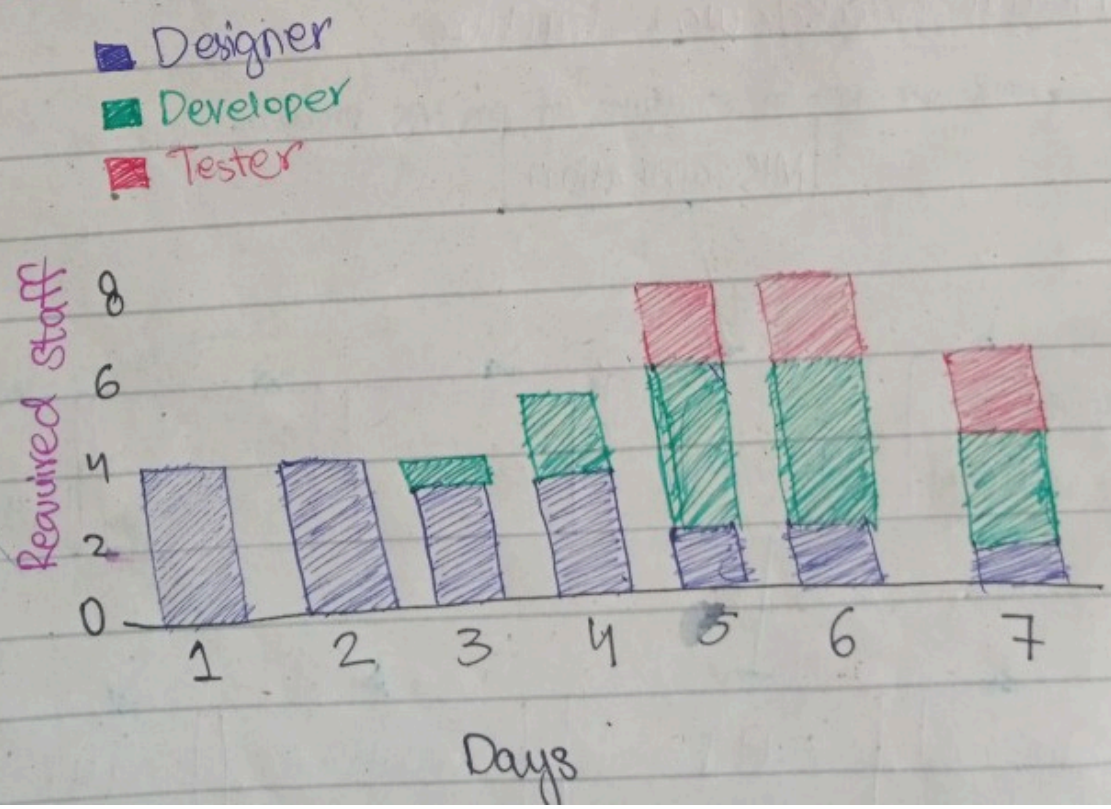
- work breakdown structure of an MIS problem.



- WBS is management tool & necessary part of project designing.
- It's task Oriented system for sub-dividing a project into product parts.

- **Resource Histogram** – Graphical tool that contains bar or chart representing number of resources.
  - It's an effective tool for staff planning & coordination.

Staff	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Designer	4	4	3	3	2	2	1
Developer	0	0	1	2	4	4	3
Tester	0	0	0	0	2	2	2
Total	4	4	4	5	8	8	6



Risk Management

→ Overview.
→ Classification of risk.
→ Reactive & Proactive Risk management strategies.
→ Types of Risks.
→ Examples.
→ Risk Assessment (Risk Mangt Activities.)
→ Risk Control vs Risk Mitigation.
→ Stratigies / Plans
→ Project scheduling
→ ~~Project~~ Planning Personnel.

...row problems are
...'s risk."

Risk Management

- A risk is "an uncertain event or condition that, if it occurs, has a positive or negative effect on a project's objectives.
  better say:
  "Tommorow problems are today's risk."
- A risk management plan is a document that a project manager prepares to forsee risks, estimate impacts, & define response to risks.

Risk Categories

1. Known Risks — that can be uncovered after careful assessment of project program, the bussiness & technical environment in which the plan is being developed.
2. ~~Un~~Predictable Risk
   — The risks that are hypothesized from previous project experience. (e.g Past turnover).
3. Unpredictable Risk
   — The risks that can & do occurs, but are extremely tough to identify in advance.

Reative & Proative RM strategiese (Risk Identification)

- Reactive Risk Management tries to reduce the damage of potiential threads & speed an organization's recovery from them, but assumes that those threats will happen eventually.
  ↳ like — we assume risk happens, our strategies are we reduce risk. e.g if (risk) happens at one company - we must do strategies & try to reduce risk, we shift to another location in case.
- Proative risk management identifies threats & aims to prevent those events from ever happening. In the first place.
  ↳ firstly taking safety measures to prevent risk, damage. e.g, safety measures while driving a car.

Types of Risks

- Bussiness Risks — this contains building an excellent project/product that no one needs, losing budgetary or personnel commitments.
- Technical Risks
  — Concerned with Quality, Design,

implementation, interface, maintanance problems.

- Most technical risks appear due to the development team's insufficient knowledge about project.

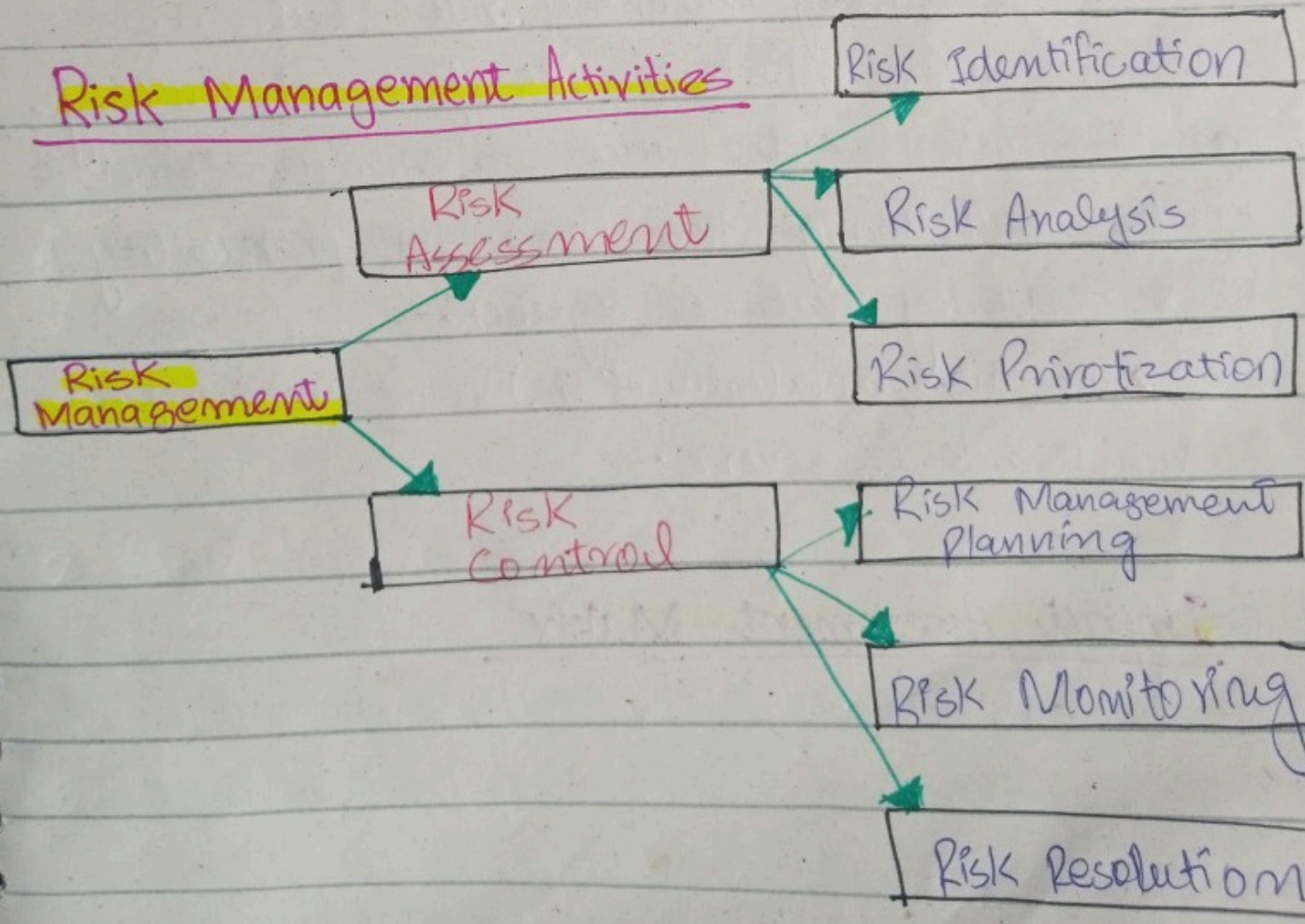
Project Risk

- Concerns different form budgetary, schedule, personnel, resource, customer-related problems.

Examples.

- On January 28, 1986, "the challenger" space shuttle exploded just 76 seconds after take off.

Risk Management Activities



Risk Assessment

- It's a process to rank the risks in terms of thier damage
- Determine the average probability of occurence value for each risk.
  - ↳ determine whether the risk / probability value of whether the risk is occuring frequently or rarely.
- Determine the impact for each component based on impact assessment matrix (Severity Value).
  - ↳ e.g, rarely occuring risk but severe effects, like fire at company.
- Risk assessment values are determined by multiplying the scores for probability & severity values together.
  - ↳ on this base (calculations), we can priotize which risk is severe.

Impact Assessment Matrix

Probability	Negligible: 1	Marginal: 2	Critical: 3	Catastrophic: 4
Frequent: 5	medium-5	High-10	v-high-15	Very-high-20
Likely: 4	medium-4	High-8	High-12	v-high-16
Occassional: 3	Low-3	medium-6	High-9	High-12
Unlikely: 2	Low-2	medium-4	medium-6	High-8
Rare: 1	Low-1	Low-2	Low-3	medium-4

Risk Control (Management) vs Risk Mitigation

- Risk Control is basically the specific actions to reduce a risk event's probability of happening. i.e; correct inspection & maintenance of car reduce the likelihood of mechanical failures.

- Whearas Risk Mitigation is the set of actions to reduce the impact of Risk Event. i.e; Airbags are used to reduce the impact of an accident on the passengers & driver.

Strategies/Plans

- Avoid the Risk — This may include — discussing with client to change the requirements to decrease the scope of work, ~~giving~~ etc.
- Transfer the Risk — this involves getting the Risky element developed by a 3rd party, buying insurance cover etc.

- **Risk Reduction** – means planning method to include the loss due to risk. i.e; if there's a risk that some key personnel might leave, new recruitment can be planned.

- **Risk Monitoring**

  — it's the method king that your assumption about product, process & bussiness risks has not changed.

Project scheduling → from Pdf

Project planning → from Pdf.

SCM

→ Overview
→ Need of SCM
→ Importance of SCM.
→ SCM process.
→ SCM Quality Assurance.
→ Importance of Quality.
→ SW Quality Assurance!
→ SQA Activities.
→ QA & QC Difference.
→ Process Monitoring & Control.

termediate version of modul

uring debugging, parts

Software Configuration Management

When we develop SW, product undergoes many changes in maintenance phase.
Several individuals (programs) produces several work product (SC items) i.e; intermediate version of modules or test data used during debugging, parts of the final product.

"The elements that comprise all info produced as a part of SW process are collectively called a SW Configuration".

As SW dev progresses, the number of SW Configuration elements (SCI's) grow rapidly.

SCM is the discipline which;

- Identify change.
- Monitor & Control change.
- Ensure the proper implement of change made to the item.
- Auditing & reporting on the change made.

Need of SCM

- multiple ppl are working on SW which is consistently updating.
- It may be a method where multiple version,

branches, authors are involved in SW project, & the team is geographically distributed & works concurrently.
- It changes in user requirements, & policy, budget, schedules need to be accommodated.

Importance of SCM

- It is practical in controlling & managing the access to various SCI's e.g, by preventing the two members of a team for checking out the same component for modification at the same time.
- It provides the tool to ensure that changes are being properly implemented.
- It has the capability of describing & storing various constituent of SW.
- SCM is used in keeping a system in a consistent state by automatically producing derived version upon modification of the same component.

SCM Process

It uses tools which keep the ~~u~~necessary change

has been implemented adequately to the appropriate component. It define number of process;

- Identification
  - Basic Object: Unit of text created by a SW Engineer during Analysis, Design, Code or Test.
  - Aggregate Object: collection of essential objects & other aggregate objects.
  - Design specification is an aggregate object.

- Version Control — Version Control combines procedures & tools to handle different version of configuration objects that are generated during SW process.
  Clemm defines version control in the context of SCM: allows to specify alternative configuration of SW system through selection of appropriate versions.
  — This is supported by associating attributes with each SW version, then allowing a configuration to be specified by describing the set of desired attributes.

- Change Control — James Bach describes CC in context of SCM is: Change Control is Vital.

But the forces that make it essential also make it annoying.

- We worry about change bcz a single rogue Dev could sink the project, yet brilliant ideas originate in mind of those rogues &
A burdensome change control process could ~~process~~ effectively discourage them from doing creative work.
- A change request is submitted & calculated to assess technical merit, potential side effects etc.
- The results of evaluations are presented as a change report, which is used by a change control authority (CCA).
- The "check ~~out~~" & "check-out" process implements two necessary elements of change control- access control & synchronization control.

o Access Control - governs which SW engineer has the authority to access & modify a particular configuration object.

- Synchronization Control - helps to ensure that parallel changes, performed by two different people, don't overwrite one another.

- **Configuration Audit** — SCM audits to verify that the SW product satisfies the baselines requirements & ensures that what is built & what's delivered.
  - SCM audits also ensures the traceability is maintained between all CIs & that all work requests are associated with one or more CI modification.
  - SCM audits are "watchdogs" that ensures the integrity of projects

- **Status Reporting** — Configuration status reporting (sometimes also called accounting) providing accurate status & current configuration data to developers, testers, & users, customers & stakeholders through admin guides, user-guides, FAQ's, Release Notes, Installation Guide, Configuration Guide, etc.

SW Quality Assurance

Quality defines to any measureable charateristics such as correctness, maintainability, portability, testability, usability, reliability, efficiency, integrity, reuseability & interoperability.

Quality of Design

Quality of Design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, & performance specifications that all contribute to the quality of Design.

Quality of Conformance

Quality of conformance is the degree to which the design specifications are followed during manufacturing. Greater the degree of conformance, the higher is the level of quality of conformance.

- **Quality Assurance** — Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully.
  - focuses on how the engineering & management activity will be done?
  As anyone is interested in the quality of final product, it should be assured that we are building the right product.
  - It can be assured only when we do inspection & review of intermediate products, if there are any bugs, then it is debugged. This quality can be enhanced.

Importance of Quality

- **Increasing criticality of SW** — The final customer/user is naturally concerned about the quality of SW, especially it's reliability. This is increasing in the case as organizations become more dependent on thier computer systems & SW is used more & more in safety-critical areas. e.g. to control aircraft.
- **The intangibility of SW**

This makes it challenging to know that a particular task in a project has been completed satisfactorily. The results of these tasks can be made tangible by demanding that the developers produce "deliverables" that can be examined for quality.

Software Quality Assurance — it's a planned & systematic plan of all autions necessary to avoid adequate confidence that an item/product conforms to establish technical requirements.

A set of activities designed to calculate the process by which the products are developed or manufactured.

SQA Encompasses

- A Quality Management Approach.
- A multitier testing strategy.
- Control of SW documentation & changes made to it.
- measuring & reporting mechanisms.

SQA Activities

- Prepares SQA plan — The plan governs quality assurance activities performed by the SW engineers & SQA group, plan identifies calculation to be performed, audits & reviews to be performed etc.
- Review SW engineering activities to verify compliance with the defined SW process.

The SQA group identifies, reports, & tracks deviations from the process & verifies that corrections have been made.

- Audits

SQA group reviews selected work products, identifies, tracks deviations, " " & periodically reports the results of its work to the project manager.

- Ensures that deviations are handled according to document

Deviations may be encountered in the project, method, process description, applicable standards, or technical work products.

- Records any non-compliance — Non-compliance items are traked until they are resolved.

QA	QC
Quality Assurance is set of actions including facilitation, training, measurement, & analysis to provide adequate confidence that processess are establis & continously improved to produce products or services that conform to specifications and are fit for use.	Quality control is described as the processess & methods used to compare product quality to requirements & applicable standards, & the actions are taken when a non-conformance is detected.
— QA helps establish process.	— QC relates to particular product or service.
— It's a managerial tool	— It's a corrective tool.
— Verification is an example of QA.	— Validation is an example of QC.

Process Monitoring & Control

- **Monitoring & control project work** — The generic step under which all other monitoring & controlling activities fall under.
- **Perform integrated change control** — When changes to schedule, cost, or any other area of the project management are necessary, the program is changed & re-approved by the project sponsor.
- **Validate scope** —activities involved with gaining approval of the projects deliverables.
- **Control Scope** — Ensuring that scope of project doesn't change & that unauthorized activities are not performed as part of plan (scope creep)
- **Control schedule** — The functions involved with ensuring the project work is performed according to schedule, & that project deadlines are met.
- **Control Costs** — The tasks involved with ensuring the project costs stay within the approved budget.
- **Control Quality** — ensuring that the quality of project deliverables is to the standard defined in the project management plan.

- Control communications — providing for the communication needs of each project stakeholder.
- Control Risks — Safeguarding the project from unexpected events that negatively impact the project's budget, schedule, stakeholder needs etc.
- Control procurements — Ensuring the project's sub-contractors & vendors meet the project's goal.
- Control stakeholder engagement

— ensuring that all the stakeholders are left satisfied with project work.

SW Quality

project's goal.

ment

akeholders are left satisfied

SW Quality

→ Overview / Example.
→ modern Quality methods.
→ Quality system Activities.
→ ISO 9000 Certification.
→ Types of ISO 9000.
→ How to get ISO 9000.

atisfaction of the

- Control. procurements — Ens
sub-contractors & vendors m
- Control stakeholder engagem
—ensuring that all the stak
with project work.

SW Quality

- SW quality product is defined in terms of it's fitness of purpose. That is, a quality product does precisely what the users want it to do.
- For SW products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in SRS document.
- Although "fitness of purpose" is a satisfactory interpretation of quality for many devices i.e car, a table fan etc — for SW products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example

- Consider a functionality correct SW product.

That is, it performs all tasks as specified in SRS Doc. But, has an almost unuseable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

Several modern Quality methods

- Probability — A SW device is said to be portable, if it can be freely made to work in various OS environments, in multiple machines, with other SW products etc.
- Reuseability — A SW product has excellent reuseability if different modules of product can quickly be reused to develop new products.
- Useability — A SW product has better useability if various categories of users can easily invoke the functions of product.
- Correctness — A SW product is correct if various requirements as specified in the SRS Doc have been correctly implemented.
- Maintainability — A SW product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to product, & functionalities of product can be easily modified, etc.

Quality System Activities

Quality system activities encompass following:

- Auditing of projects.
- Review of Quality system.
- Dev of standards, methods, & guidlines, etc.
- Production of DoS's for the top management summarizing the effectiveness of quality system in organization.

ISO 9000 Certification

- Iso (International Standards Organization) is a group/ consortium of 63 countries established to plan & fosters standardization.
- Iso declared it's 9000 series of standards in 1987.
- The ISO 9000 standard determines the guidlines for maintaining a quality system.
- ISO 9000 defines a set of guidelines for the production process & is not directly concerned about the product itself.

Types of ISO 9000 Quality standards

- ISO 9001 – This standard applies to the

organizations engaged in Design, Development, production & servicing of goods. — it applies to most SW development organizations.

- **ISO-9002**

— This standard applies to those organizations which do not Design products but are only involved in production.

— Examples of this category industries include steel & car manufacturing industries that by the product & plants designs from external sources & are engaged in only manufacturing those products.

— Therefore, ISO 9002 doesn't apply to SW Dev Org.

- **ISO-9003**

— This standard applies to the organizations that are involved in the installation & testing of products.

— Example, Gas companies.

How to get ISO 9000 certification

An organization determines to obtain ISO 9000 certification applies to ISO registrar office for registration. The process;

- **Application** — Once an organization decided to

go for ISO certification, it applies to the registrar for registration.

- **Pre-Assessment** – During this stage, the registrar makes a rough assessment of organization.
- **Doc review & Adequacy of Audit** – the registrar reviews the Document submitted by organization & suggest an improvement.
- **Compliance Audit** – the registrar checks whether the organization has compiled the suggestion made by it during the review or not.
- **Registration** – The registrar awards the ISO certification after successful completion of all phases.
- **Continued inspection** – The registrar continued to monitor the organization time by time.

Software Design

SW Design

→Overview →SW Design Levels
→Objectives of SW Design.
→SW Design Principles.
→SW analysis & Design Tools.
→SW design strategies

n time by time.

SW Design

→ Overview
→ SW Design Levels
→ Objectives of SW Design.
→ SW Design Principles.
→ SW analysis & Design Tools.
→ SW design strategies.
→ Design Verification.
→ User Interface Design on Pdf.

gn

nism

ne

im

enting the client's

described in SRS Document

Software Design

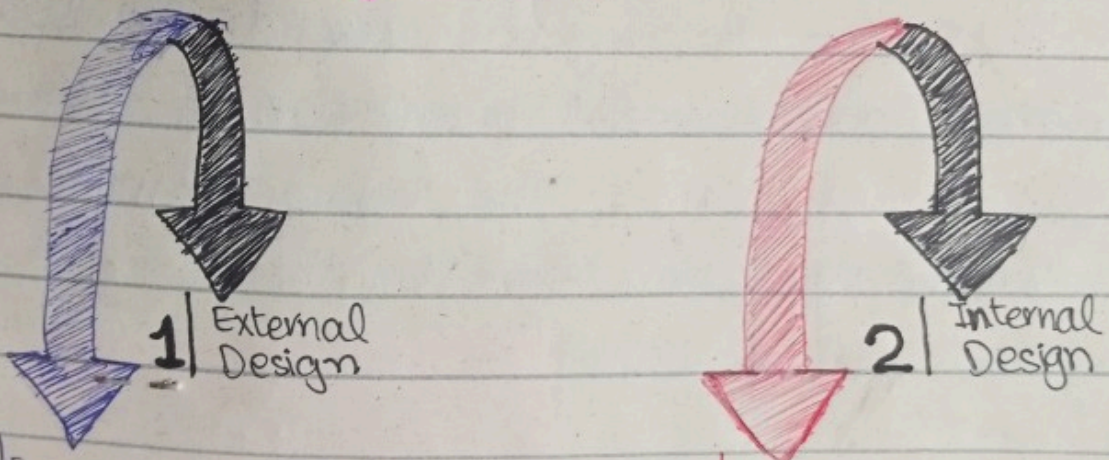
SW design is a mechanism to transform user requirements into some suitable form, which helps the programmer in sw coding & implementation.

- It deals with representing the client's requirement, as described in SRS Document,

into a form i.e, easily implemented using programming language.

- The SW design phase is the first step in SDLC (SW Design life cycle), which moves the concentration from problem domain to solution domain.
- In SW Design, we consider the system to be a set of components or modules with clearly defined behaviours & boundaries.

Software Design Levels : It has 2 levels



1. Focus on deciding which modules are needed, thier specification, ~~thier~~ relationship & interface among them.

2. Focus on planning & specifying the internal structure & processing detail.

Objectives of SW Design

Correctness : SW design should be correct as per requirement.

Completeness : Design should have all components like data structures, modules, & external interfaces, etc.

Efficiency : resources should be used efficiently by the program.

Flexibility : Able to modify on changing needs.

Consistency : There should'nt be any inconsistency in design.

Maintainability : The design should be so simple so that it can be easily maintainable by other designers.

Software Design Principles — provide means to handle the complexity of the design process effectively.

- This will not only reduce effort needed for design but can also reduce the scope of introducing errors during design.

Principles of SW Design

- Problem Partitioning
- Abstraction
- Modularity
- Top down & bottom up strategy

1. **Problem Partitioning** — For small problem, we can handle the entire problem at once but for the significant problem, divide the problems & conquer the problem — it means to divide the problem into smaller pieces so that each piece can be captured seperately.

Benefits of Problem Partitioning

- SW is easy to understand.
- SW becomes simple
- SW easy to test.
- SW easy to modify.
- SW easy to maintain.
- SW is easy to expand.

— These pieces can't be entirely independent

of each other as they together form the system. They have to cooperate & communicate to solve the problem. This communication adds complexity.

2. Abstraction — An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implement.
— Abstraction can be used for existing element as well as the component being designed.

Functional Abstraction

- A module is specified by the method it performs.
- The details of the algo to accomplish the functions are not visible to user of the function.
- (FA) forms the basis fo Function oriented design approaches.

Data Abstraction

- Details of the data elements are not visible to the users of data. ~~Data abstraction for~~ Data abstraction forms the basis for Object oriented design approaches.

3. Modularity — specifies ~~the~~ to the division of SW into seperate modules which are differently named

& addressed & are integrated later on in to obtain the completely functional SW.

- It's the only property that allows a program to be intellectually manageable.
- Single large problems are difficult to understand & read due to large number of reference variables, control paths, global variables, etc.

The desireable properties of modular system:

- Each module is a well-defined system that can be used with other applications.
- Each module has single specified objectives.
- Modules can be seperately compiled & saved in library.
- Modules should be easier to use than to build.
- Modules are simpler from outside than inside.

Advantages

→ It allows large programs to be written by several ppl.
→ simplifies the overlay procedure of loading large program into main storage.
→ provides more checkpoints to measure progress.
→ provides a framework for complete testing.

Dis-advantages

→ storage size perhaps, but is not certainly, increased.
→ Compilation & loading time may be longer.

→ more linkage required, runtime may longer / more source lines must be written & more documentation has to be done.

Modular Design (or Structured Design) — also a design approach.
— reduces design complexity
— results in easier & fast implementation by allowing parallel development of various parts of a system.

1. Functional Independence — it's achieved by developing functions that perform only one kind of task & don't excessively interact with other modules
— Independence is more important bcz it makes implementation more accessible & faster.
— (FI) is good design feature that ensures S.W quality.

It is measured by:

1 - Cohesion — it's a measure that defines the degree of intra-dependability within elements of module.
— The greater the cohesion, the better the program design.

♦ Logical Cohesion
— when logically categorized elements are put together into modules.

♦ Temporal Cohesion — when elements of module are organized such that they are processed at similar point in time.

♦ Procedural Cohesion - when elements of module are organized together, which are executed

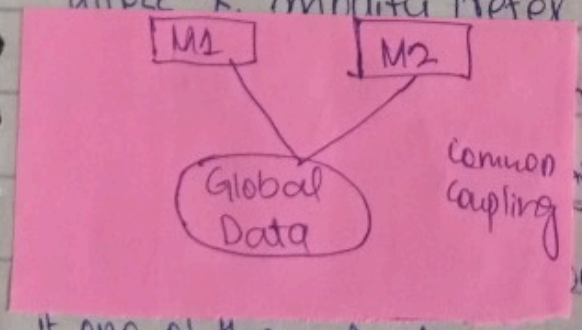
sequentially in order to perform a task.

- Communicational Cohesion – when elements of module are grouped together, which are executed sequentially & work on same data (info).
- Sequential Cohesion – when elements of module are grouped together bcz the output of one element serves as input to another & so on.
- Functional Cohesion – considered as highest degree of cohesion & highly expected.

– here, elements of module are grouped bcz they all contribute to a single well-defined function.
– It can also be reused.

2. Coupling – a measure that defines the level of inter-dependability among modules of program.
– It tells at what level the modules interfere & interact with each other.
– The lower the coupling, the better the program.

- Content Coupling – when module can directly alter & modify/refer to content of another module.



hen multiple modules have read & lobal data.

dules are called controlled-coupled
it one of them decides the function of other module or changes it's flow of execution.

Stamp coupling - when multiple modules share common data structure & work on different part of it.

Data coupling - when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then recieving module should use all it's components.

- Ideally, no coupling is considered to be the best.

2. Information Hiding - modules should be specified that data include within a module is inaccessible to other modules that don't need for such information.

- The use of info hiding as design criteria for modular system provides most significant benefits when modifications are required during testing & later during SW maintenance.

SW Analysis & Design Tools

- DFD → function-oriented
  - levels of DFD
- Data Dictionary - F-O
- E-R model
- Structured charts - F-O

• Defined in SW requirement

↳ Pseudo-Code — PC is written more close to programming language.

— It may be considered as augmented programming language, full of comments & descriptions.

— PC avoids variable decleration but they are written using some actual programming language's constructs, like C, Fortran, Pascal etc.

SW Design Strategies

→ Function-Oriented Design — is a method to SW design where the model is decomposed into a set of interacting units or modules where each module/unit has clearly defined function

Design Notations

- DFD
- DD
- Structured charts
- Pseudo code-
- Top-down approach
- Divide & conquer approach.

- Structured charts –

— It partitioned a system into block-boxes.

— A black-box system that's functionality is known to the user without the knowledge of internal design.

already defined in SW requirement.

— UML is used.

→ **Object-Oriented Design** — bottom-up approach

— here, the system is viewed as a collection of objects (entities)

— The tasks defined for one purpose can't refer/ change data of other objects.

— objects have thier internal data that represent thier state.

— Similar objects create a class.

— Classes may inherit features from the superclass.

terms related to O-O design

- **Objects** — all entities involved in solution design. e.g., Person, company & users considered as objects.

— Every entity has some attributes

— & has some methods to perform on attributes.

- **Classes**

— class is a generalized description of ~~entity~~ (obj).

— obj is an instance of class.

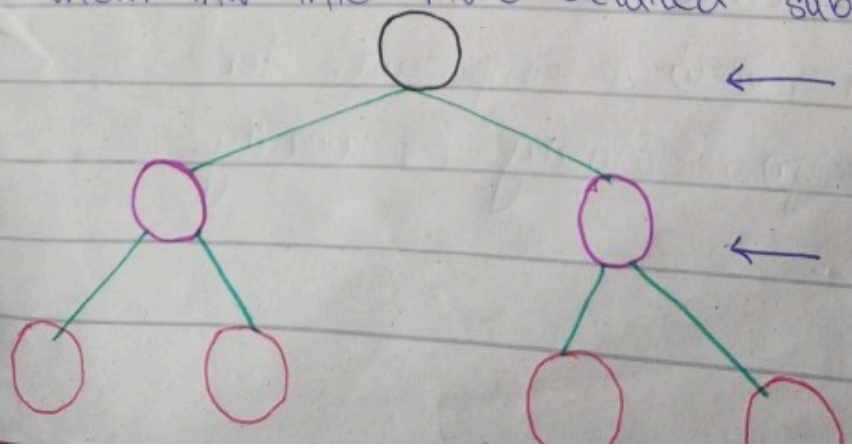
— class defines all attributes & methods of object.

- **Messages** — objects communicate by msg passing.

— Messages are often implemented as procedure or function calls.

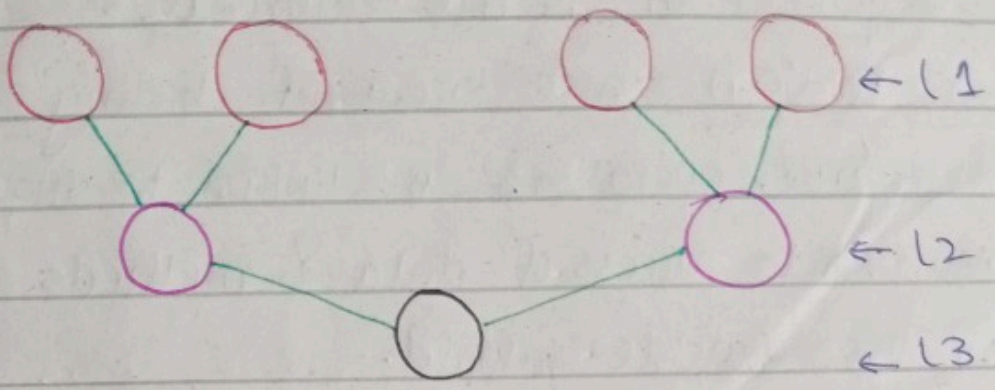
- **Abstraction** – In oo design, complexity is handled using abstraction.
  – abstraction is the removal of the irrelevant & the amplification of the essentials.
- **Encapsulation** – also know info hiding concept.
  – The data & operations are linked to single unit.
  – it also restricts access of data & methods from the outside world.
- **Inheritance**
  – OOD allows similar classes to stack-up in a hierarchical manner where the lower or sub-classes can import, implement, & re-use allowed variables & functions from thier immediate super-classes.
- **Polymorphism** – OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name.

→ **Top-down Approach** – This approach starts with the identification of the main components & then decomposing them into thier more detailed sub-components.



→ Bottom-up-Approach — it begins with the lower details & moves towards up the hierarchy.

— This approach is suitable in case of an existing system.



Design Verification

- The output of SW Design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, & detailed description of all functional & non-functional requirements.
- The next phase which is implementation of SW, depends on all outputs mentioned above.
- It is then becomes necessary to verify the output before next phase. — The early any mistake is detected
- By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions.
- A good design review is important for good SW design, accuracy & quality.

Architectural Design

→ Overview.
→ SW-A Models.
→ Taxonomy of Architectural styles.

Architectural Design

Architectural Design in SE i
the system into interating u
- It's expressed as a block dia
overview of the structure diagram, features of
components,
- It identifies the components that are necessary for developing a computer-based system & communication between them i.e; relationship b/t these components.
- The architectural design process is about identifying the components i.e sub-systems that make-up the system & structure of sub-system & they're relationship.
- It acts as a link between specification requirements & the design process.

~~System properties of A-D~~

SW Architectural Models

To document A-D process, authitectural models are used:

- Static Type of architectural structural model represents the major system components.
- Dynamic Type of architectural ~~mod~~ process model represents the process structure of the system.

- Distribution Type of architectural model represents how the component is distributed across various computers.
- Interface Type of architectural model represents the interface of components.
- Relationship Type of architectural model represents models such as DFD to represent the component interrelationship.

Architectural design models are application domain-specific & most common two types of domain-specific models are;

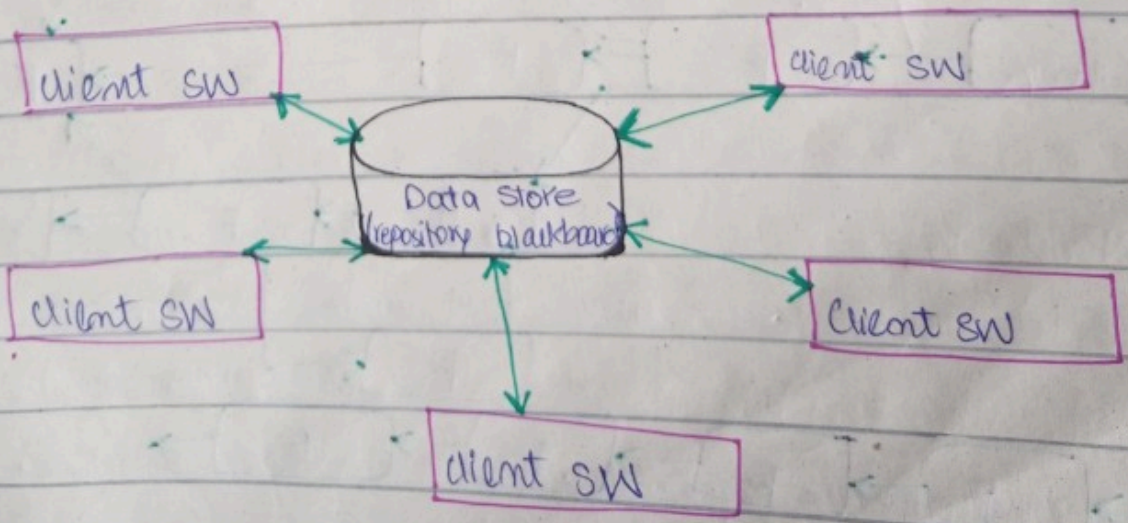
- Generic Model — These models are abstractions derived from a number of real systems & encapsulated the characteristics of these systems. This type of model usually follows a bottom-up approach.
- Reference Models — These models are providing info regarding the class of system. They are derived from application-domain rather than ~~following~~ from existing systems. It usually follows the top-down approach. It provides a comparison between different Software architecture.

Taxonomy of Architectural styles

- **Data centered Architectures** — A data store will reside at the center of this architecture & is accessed frequently by other components that update, add, delete or modify the present within the store.
  - This data centered architecture will promote integrability. This means that existing components can be changed & new client components can be added to architecture without permission & concern of other clients.
  - Data can be passed among clients using blackboard mechanism.

Advantages

- Repository of data is independent of clients.
- It may be simple to add additional clients.
- Modification can be very easy.



Data Flow Architectures

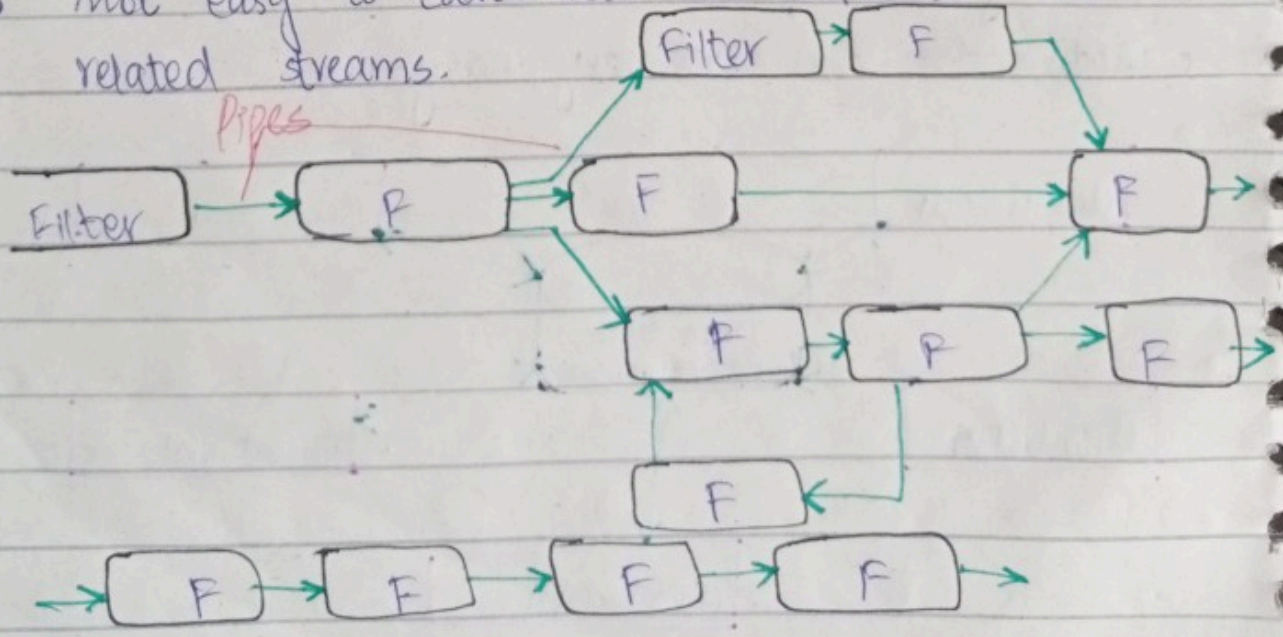
- This kind of architecture is used input data is transformed into output data through a series of computational manipulative components.
- pipe-&-filter architecture —— uses both pipe & filter & it has a set of components called filters connected by lines.
- pipes transmit data from one to next component.
- Filter works independently & take data input of certain form & produces data output to next filter of specified form.

Advantages

- encourages upkeep, repurposing & modification.

Disadvantage

- DFA doesn't allow app's that require greater user engagement.
- not easy to coordinate two different but related streams.



Object-Oriented Architecture

- The components of system encapsulate data & the operations that must be applied to manipulate data
- The cordination/communication b/w components are established via the message passing.

Charauteristics

- Object protects the system's integrity.

Advantage

- enables a designer to seperate a challenge into collection of autonomous objects.

Layered Architecture

- A number of different layers are defined with each layer performing a well-defined set of operations.
- Each layer will do some operations but become closer to machine instruction set progressively.
- At outer layer, components will recieve the user interface operations
- At inner layers, components will perform communication with OS.
- Intermediate layers to utilize services & application SW functions.
- Common example is OSI-ISO communication system

(Android) • Lucid Chart

ual Studio.

Software Implementation

→ Programming methods.
→ Structured programming.
→ Functional programming.
→ SW Documentation.
→ SW Implementation challenges.

SW

next to impossible to remem

if one forgets flow & underly

Software Implementation

programming methods → documentation & challenges in SWI.

Structured programming

In coding, lines of code keep multiplying, size ↑. Gradually, it becomes next to impossible to remember the flow of program. If one forgets flow & underlying programs, procedures are produces, it become difficult to debug, share & modify the program.

- The solution is structured programming — brings clarity in code, improving it's efficiency.
- St-Programmings helps reducing coding time.
- it states how program shall be coded.

it uses three concepts:

- **Top-down analysis**

- A SW is always made to perform some rational work. known as problem.
- Thus, it's very important that we understand how to

solve the problem.

- Under top-down analysis, problem is broken down into small pieces. Each problem is individually solved & stated clearly how to solve problem.

- **Modular Programming**

- code is broken down into smaller group of instructions.
- groups called — modules, sub-routines, subprograms.
- it's based on top-down analysis.
- it discourages jumps 'goto' statements in program. which makes the program flow non-traceable.

- **Structured Coding**

- In reference with top-down analysis, structured coding sub-divides the modules into furthur smaller units of code in order of thier execution.
- It uses control structure, which controls program flow.

Functional Programming

- uses the concepts of mathematical functions.
- A F in maths should always produce the same result on recieving same argument.
- F-P provides means of computation as mathematical functions, which produces results irrespective of program state. This makes it possible to predict the behaviour of program.

It uses concepts:

- First class & High-order functions — are capable to accept another function as argument or they return other Functions as results.
- Pure functions — functions don't include destructive updates.
  - don't affect any I/O or memory, when not in use.
  - easily removed without hampering rest of program.
- Recursion
  - where a function calls itself & repeats program code in it unless some pre-defined condition matches.
  - it's a way of creating loops in functional programming.
- λ calculus
  - Most F.P languages use λ-calculus as thier type system.
  - λ-expressions are executed by evaluating them as they occur.

Examples

- Common Lisp
- Scala
- Haskell
- Erlang
- F#

SW-Documentation

A well-maintained documentation should involve:

- Requirement Documentation
  - a key tool for SW designer, developer & test team to carry out thier respective tasks.

- contains all functional, non-functional & behavioural description of intended SW.
- source of Doc can be previously stored data about SW, already running SW, clients interview, questionnaires & research.
- foundation for SW to be developed.
- majorly used in verification & validation phases.

o SW Design Documentation

- it contains instructions like:

o High-level SW architecture o SW Design Details
o DFD o Database Design.

- These Doc's works as repository for developers to implement the SW.

o Technical Documentation

- these Doc's are maintained by developers & actual coders.
- In these Doc's - while writing code, programmers also mention objective of code, who wrote it, where will it be required, what it does now & how it does - etc.

This Doc increases the understanding between various programmers working on same code. It enhances the re-use capability of code. Make debugging easy

& traceable.
- Java comes JavaDoc tool to generate technical documentation of code.

o User documentation - This is diff from all above Doc
- All previous documentations are maintained to provide info about SW & it's development process.
- User documentations explains how the SW product should work & how it should be used to get desired results.
- These documentations may include — SW installation procedures, how-to guides, user-guides, un-installation method — etc.

SW Implementation Challenges

o Code-reuse
- Programming interfaces of present-day languages are very sophisticated & are equipped high library functions.
- To bring cost down of end product, the organization management prefers code-reuse
- There are huge issues faced by programmers for compatibility checks deciding how much code to re-use.

- **Version Management**
  - Every time a new SW is issued to customer, developers have to maintain version & configuration related documentation.
  - This documentation has to be highly accurate & avalaible on time.
- **Target - Host**
  - The SW program, which is being developed in the organization, needs to be designed for host machines at the customers end.
  - But at times, it's impossible to design a SW that works on target machines.

UML

UI - Design

→ Overview
→ Types
→ Adv/Dis of CL
→ GUI
→ Adv/Dis
→ UI Design Principles

r command line Interface.

User Interface Design

The visual part of application or OS through which a client interacts with a computer or SW.

- It determines how commands are given to computer & how data is displayed on screen.

Types of UI

- Text-Based UI or Command line Interface.
- Graphical user Interface (GUI)

Text-based UI – relies priliminary on keyboard.
– example of this is UNIX.

GUI – relies more heavily on mouse. i.e Windows OS.

Advantages

- Typically capable of more important tasks.
- Easier to customizations option.

Dis-advantages

- Relies heavily on recall rather than recognition.
- Navigation is often more difficult.

GUI characteristics

- **Windows**
  - multiple windows allow different info to be displayed simultaneously on user's screen.
- **Icons**
  - on some systems, icon represent files. On other, icons describes processes.
- **Menus**
  - comman
  - a menu
  - for comm
- **Pointing**
  - pointin
  - selectin
  - items

Adva
- Less
- Easi
- fold
- The
- to
- In

• Menus

- commands are selected from a menu rather than typed in command line.

• Pointing

- pointing device such as mouse is used for selecting choices from a menu or indicating items of interests in a window.

• Graphics

- Graphics elements can be mixed with text or the same display.

Advantages

- Less expert knowledge is required to use it.
- Easier to navigate — easily can look through folders in a guess & check manner.
- The user may switch quickly from one task to another & can interact with several applications.
- interactive interface.

Dis-advantages

- Typically decreased options.
- usually less customizable - Not easy to use one button for tons of different variations.

UI Design Principles

→ Structure - It's overall concerned with UI architecture.

- Design should organize UI purposefully, in usual based on consistent models that are apparent & recognizable to users, putting related things

together & seperating unrelated things :

→ Simplicity

- Design should make this simple, common easy task, communicating clearly & directly in the user's language
- providing good shortcuts that are meaningfully related to no longer procedures.

→ Visibility

- design should make all required options & materials for given function visible

→ Feedback

- design should keep users informed of autions/ interpretations, changes of state or condition.

→ Tolerance

- design should be flexible & tolerant, decreasing the cost of errors & misuse by allowing undoing / redoing while also preventing bugs whenever possible by tolerating varied inputs & sequences.

Thier are different segments of GUI tools according to thier different use & platform.
i.e
Mobile GUI, computer GUI, Touch Screen GUI etc.

here's a list of few tools which come handy to build GUI:

- FLUID
- AppInventor (Android)
- Lucid Chart
- Wavemaker
- Visual studio.

Software Implem

programming methods → docum

SWI.

Software Implementation

→ Programming methods.
→ Structured programming.
→ Functional programming.
→ SW Documentation.
→ SW Implementation challenges.

— But at times, it's i
SW that works on

UML

→ Introduction. → Goals
→ Characteristics.
→ UML Association
→ UML Dependency.
→ UML Generalization.
→ UML Class Diagram.
→ UML object Diagram.
→ UML Component Diagram.
→ UML use-case diagram.
→ UML sequence diagram.
→ UML Activity Diagram.

& James Rumbaugh

- The UML diagrams are made for bussiness users, developers, ordinary people etc.

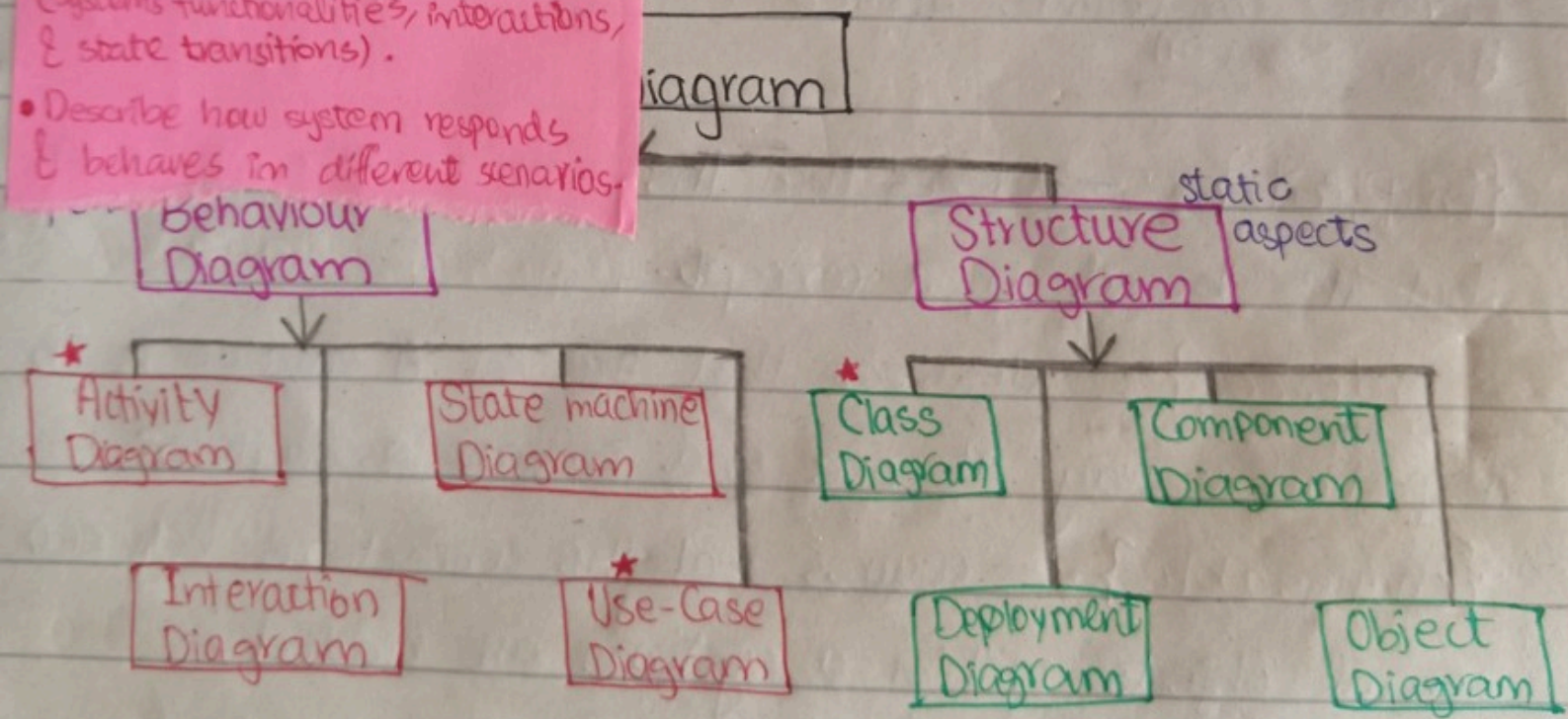
Characteristics

~ It's distinct from other programming languages like C++, Python etc.

~ it's interrelated to OOP analysis & design.

~ visualize the workflow of system.

- Focus on dynamic aspects of system. (system's functionalities, interactions, & state transitions).
- Describe how system responds & behaves in different scenarios.



UML - Association

- Association is the semantic relationship between classes that shows how one instance is connected or merged with other (in) systems.
- The objects are combined either logically or physically.
- It's catogprized as structural relationship – as connects objects of one class to another.

Some constraints applied to association relationship:

{implicit}: – defines that the relationship isn't visible, but it's based on a concept.

{Ordered}: – describes that the set of entities is in a particular way at one end in an association.

{changeable}: – ensures that the connections between several objects within a system are added, improved, & deatached, as & when required.

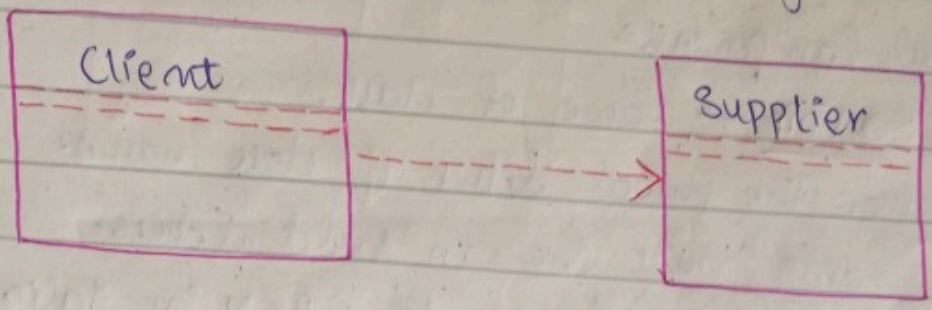
{addOnly}: – specifies that any new connection can be added from an object located at the other end in an association.

{frozen}: – specifies that whenever a link is added between objects, it can't be altered by the time it's activated over the connection or given link.

UML - Dependency

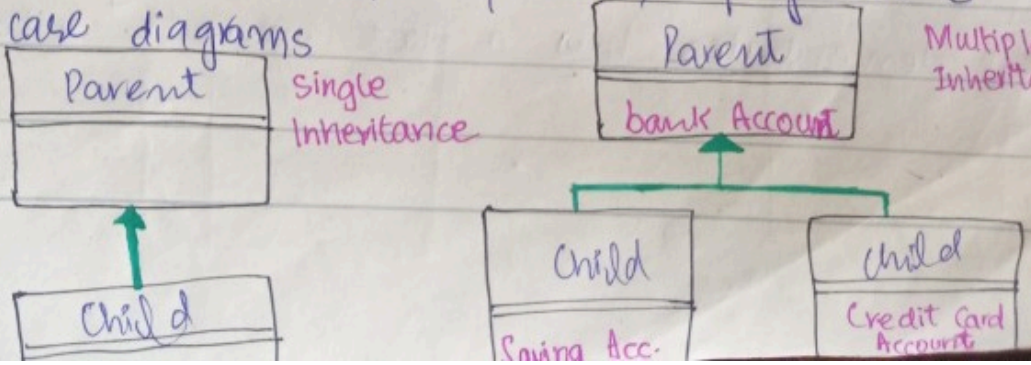
- It depicts how various things within a system are dependent on each other.
- In UML, a dependency relationship is a kind of relationship in which a client (one element) is dependent on the supplier (another element)
- It's used in class diagrams, component diagrams, deployment diagrams, & use-case diagrams.

i.e



UML - Generalization

- generalization relationship is a relationship that implements the concept of O-O - called inheritance.
- generalization relationship occurs between two entities or objects, such that one entity is the parent, & the other is the child.
- child inherits the functionality of its parent & can access as well as update it.
- it's utilized in class, component, deployment & use-case diagrams



UML Class Diagram

- depicts static view of an application.
- It represents the types of objects residing in system & relationship between them.
- visualize, document various aspects of system, & also constructs executable SW code.
- It constitutes class names, attributes, & functions.
- termed as structural Diagram.

Vital Components

Upper Section – name of class.

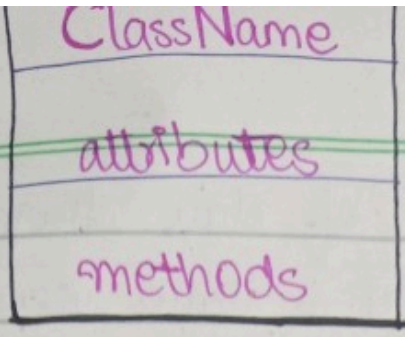
- Capitalize the initial letter of class name.
- class name must be in bold letters.
- name of abstract class should written in italic format.

Middle Section

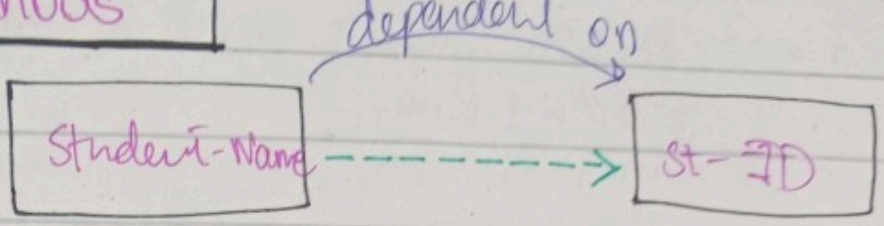
- constitutes of attributes, which is quality of class.
- The attributes are written along with its visibility factors, which are public (+), private (−), protected (#), & package (~).

Lower Section

- it contains methods or operations.
- methods are represented in form of a list, where each method is written in a single line
- It demonstrates how a class interacts with data.

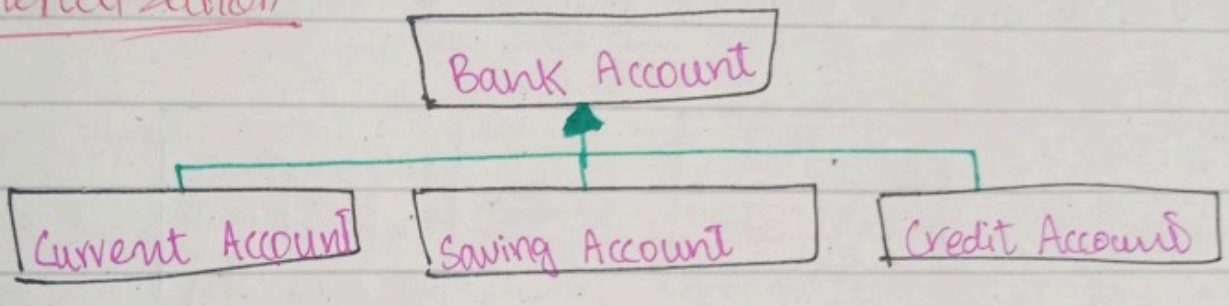


Dependency

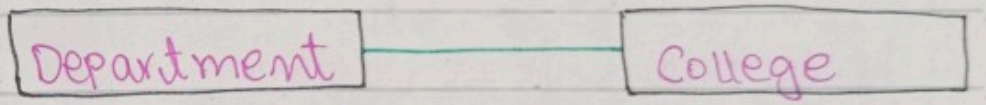


- it's weaker relationship.

Generalization

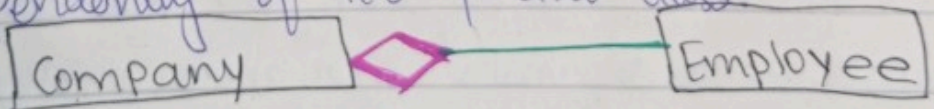


Association



- Department is associated with college.

Aggregation - aggregation is a subset of association, which represents has a relationship.
- It defines a part-whole or part-of-relationship.
- In this relationship, the child class can exist independently of it's parent class.

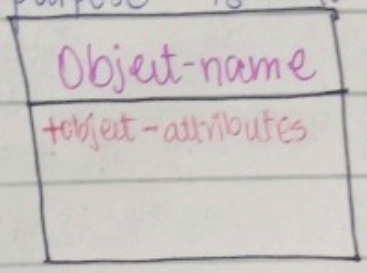


- company has employees, if one employee resigns, the company still exists.

UML Object Diagram

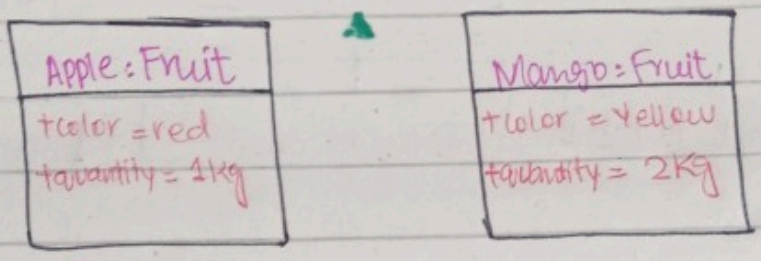
- They are dependent on class diagram as they are derived from them.

- It represents an instance of class diagram.
- main purpose is to depict a static view of system.



- It's used to perform forward & reverse engineering.

Example



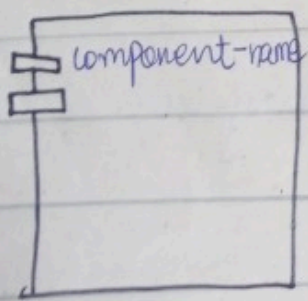
Applications of Object Diagrams

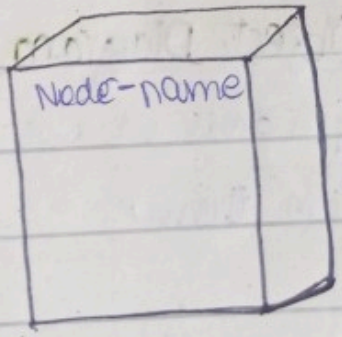
→ To build a prototype of a system.
→ To model complex data structures.
→ Reverse engineering.

UML Component Diagram – used to break down a large object-oriented system into the smaller components, so as to make them more manageable.

– It models the physical view of a system such as executables, files, libraries .etc.

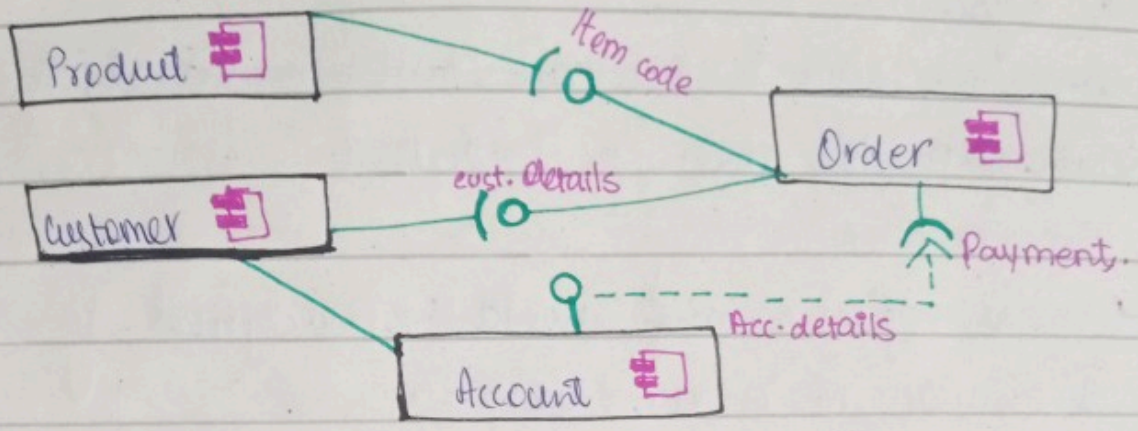
Notation.





a) Component

Example – for online shopping system



UML - Use-Case Diagram

- use-case diagram models the behaviour of a system.
- used to illustrate the functional requirements of the system & it's interaction with external agents
- it gives us the high level view of the system without going into implementation details.

Purpose

- It gathers the system's needs.
- depicts external view of system.
- represents the interaction between the actors.

How to draw

it's essential to analyze the whole system before starting. And once every single functionality is identified, they are used as use-cases in use-case diagram.

- Then enlist the actors that will interact with the system. Actors – invokes the functionality of

a system.

- once the actor & use-cases are enlisted, the relationship b/t both is inspected.

Some rules

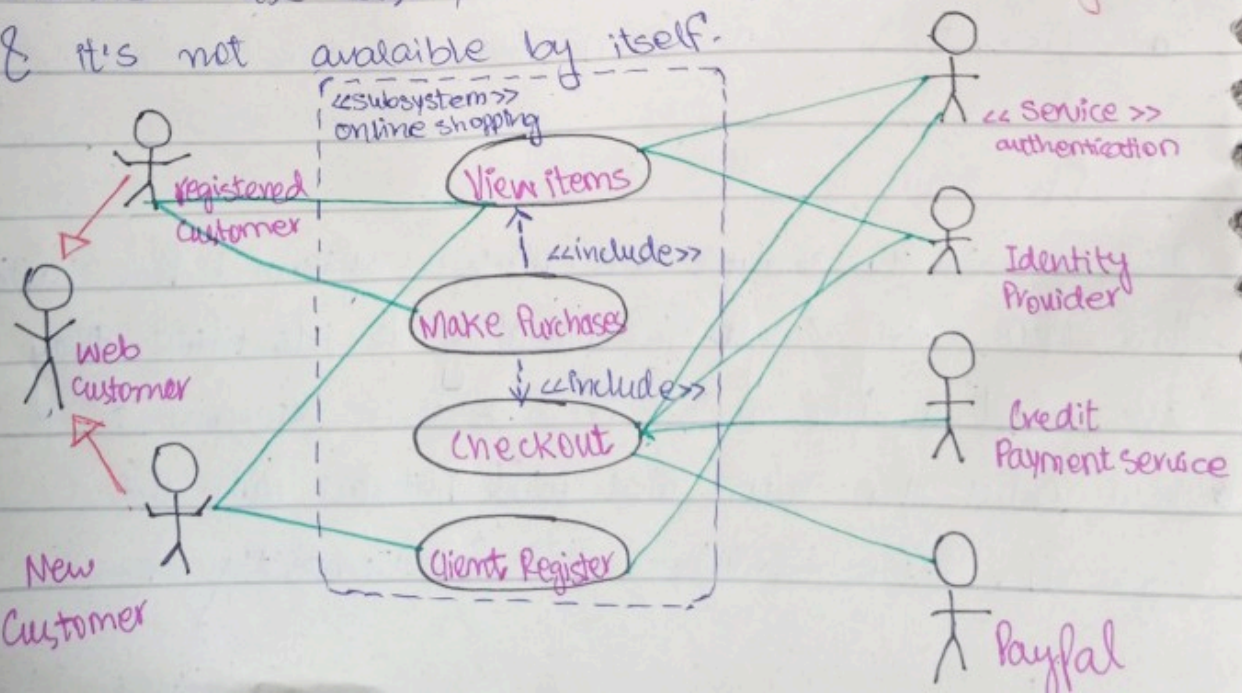
~ meaningful name should be assigned to the actor or a use case.

~ specified notations to be used as & when required.

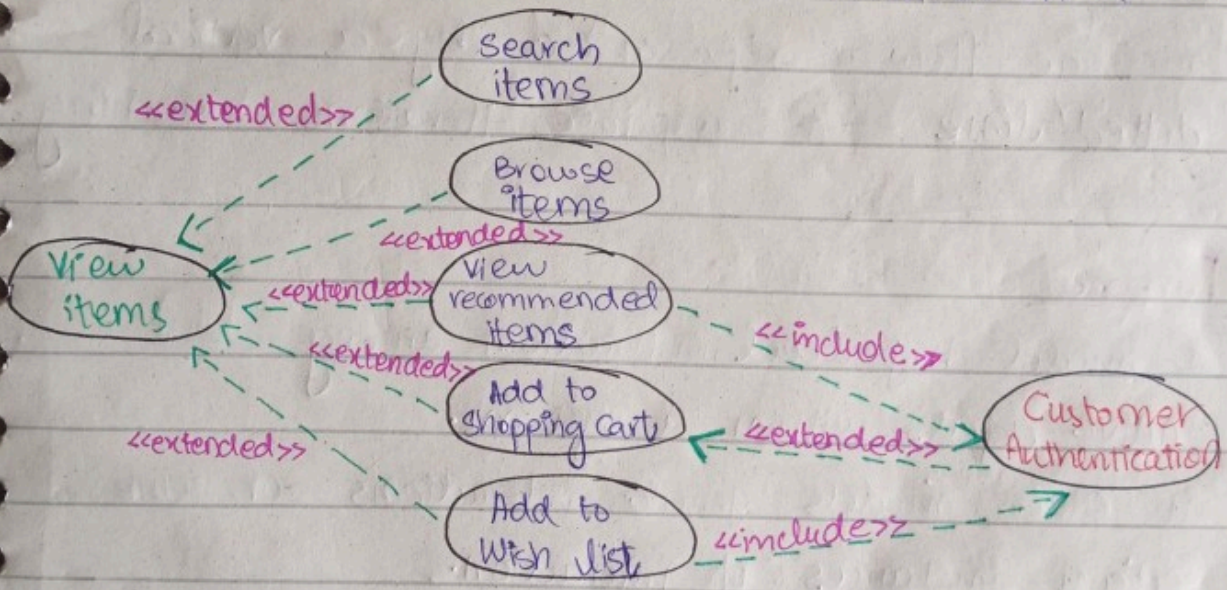
Example

—for Online shopping website.

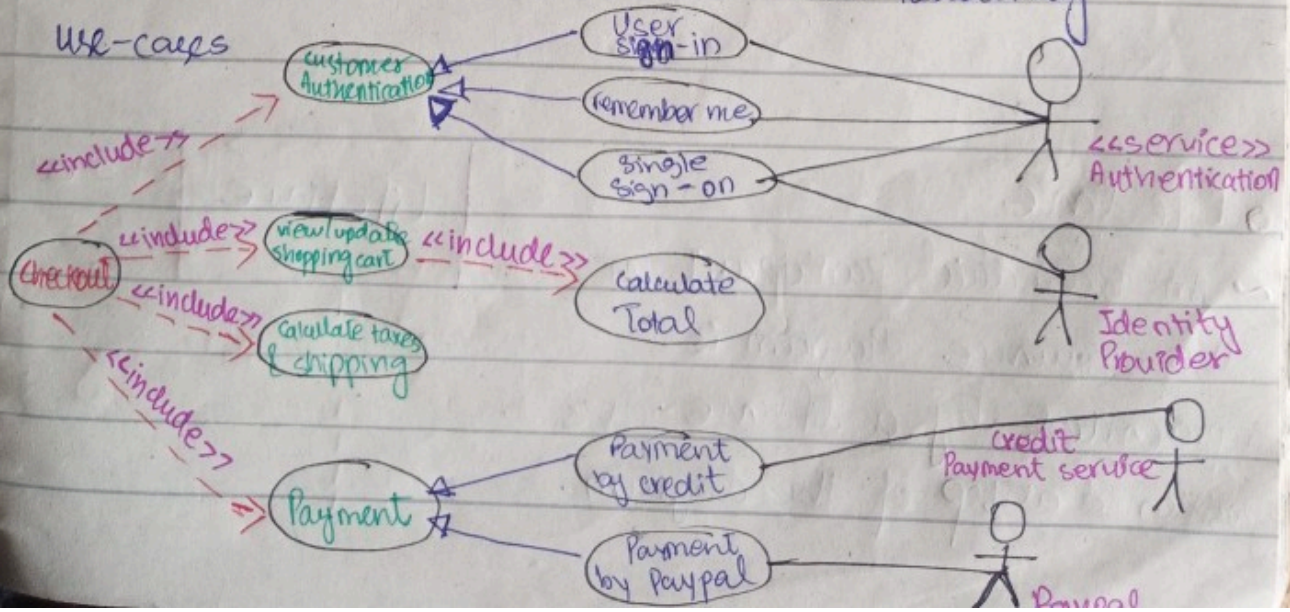
The top-level uses are; View items, Make purchases / check-out, client-register. The View items use-case is utilized by the customer who searches & view products. The Client Register use-case allows the customer to register itself with the website for availing gift vouchers, etc. Checkout is an included use case, which is part of Making purchase & it's not available by itself.



- View items can further be extended by several use-cases such as; search items, Browse items, View Recommendation items, Add to shopping cart, Add to wish list,
- And these use-cases are functions to customers, like, search for an item, and View items extended to search items, Browse items, View recommended items, Add to shopping cart, Add to wish list.
- both View recommended items & Add to wish list include customer Authentication.



→ Checkout use-case also includes the following use-cases



Sequence Diagram

- it represents the flow of messages in the system & is also termed as an event diagram.
- helps in envisioning several dynamic scenarios.
- It potrays the communication b/t any two lifelines as a time-ordered sequence of events.
- In UML, the lifeline is represented by a vertical bar,
- message flow is represented by a vertical dotted line — it incorporates iterations, branching.

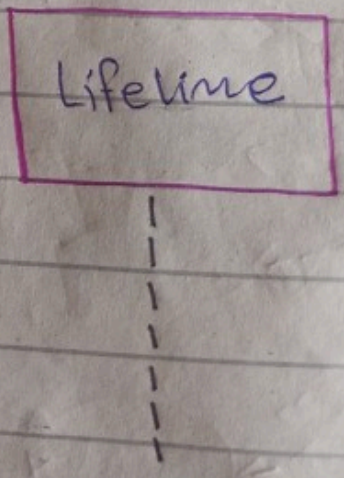
Purpose

- To model high-level interaction among active objects within a system.
- It either models generic interactions or some certain instances of interaction.

Notations

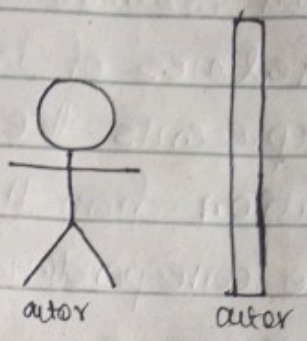
- Lifeline

~ An individual participant in the sequence diagram is represented by a lifeline. It's positioned at top of the diagram.



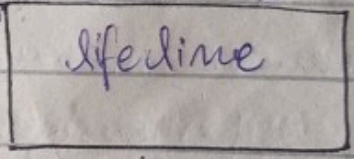
Actor

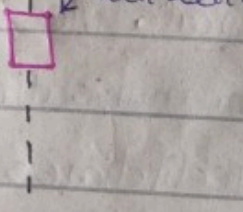
- a role played by an entity that interacts with subject — called actor.
- It's out of scope of system.
- It — involves human users, external hardware or subjects.



Activation

- It's represented by a thin rectangle on the lifeline.
- It describes the time-period in which an operation is performed by an element
- Such that, the top & the bottom is associated with the initiation & the completion time respectively.

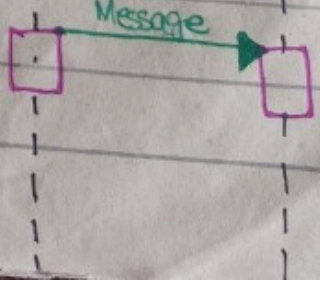




Messages

- it depicts the interaction between the objects & are represented by arrows.
- They are in sequential order on the lifeline
- The of seaverce diagram is formed by messages & lifelines.

→ Call Message — defines particular relationship b/t lifelines of an interaction, which represents that target lifeline has invoked an operation.

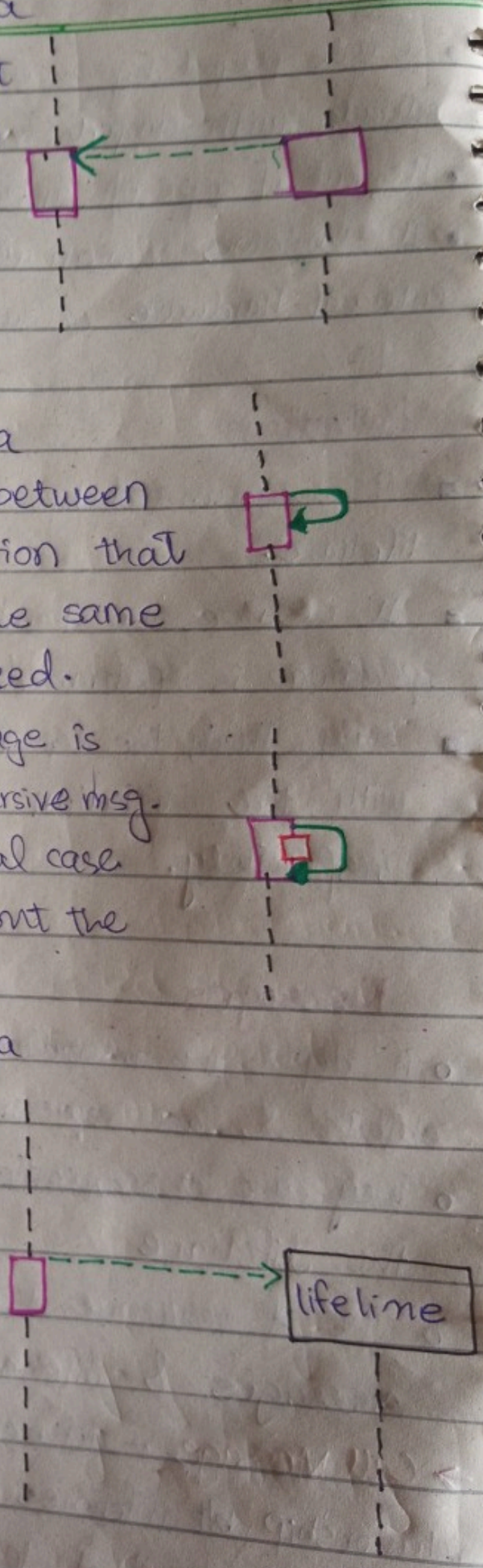


→ Return Message - defines a particular communication b/t the lifelines of interation that represents the flow of information from the reciever of the corresponding caller message.

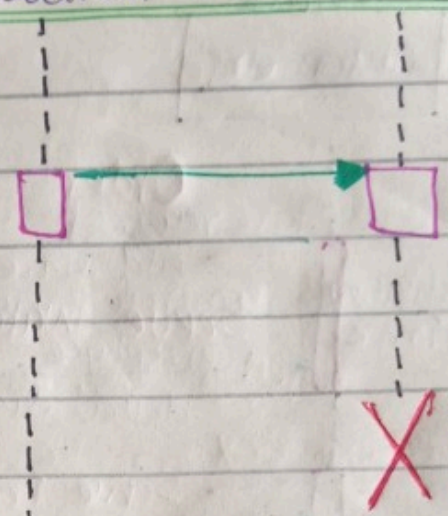
→ Self Message - describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.

→ Recursive Message - a self-message is sent for recursive purpose — recursive msg. - recursive message is a special case of self-message that represent the recursive calls.

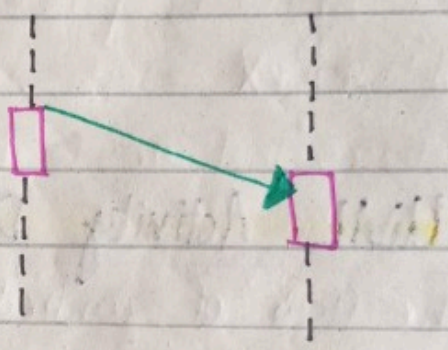
→ Create Message - it describes a communication, particularly b/t the lifelines of an interaction describing that the target (lifeline) has been instantiated.



→ Destroy Message – a communication particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.



→ Duration Message – a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.



~~Sequence Fragments~~

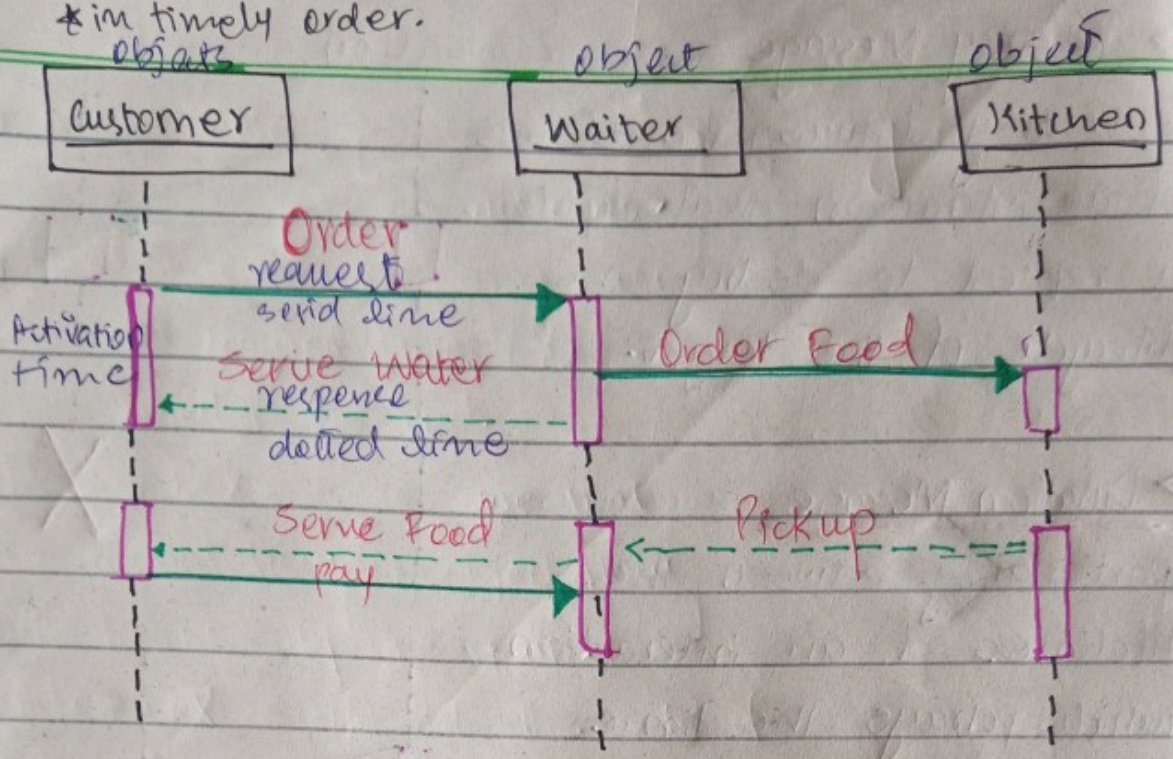
Benefits of Sequence Diagram

- It explores the real-time application.
- Implement in both forward & reverse engineering.
- easily updateable as change in system.

Drawbacks

- In case of too many lifelines, can get complex.
- may produce incorrect result, if order of msg flow change

*in timely order.



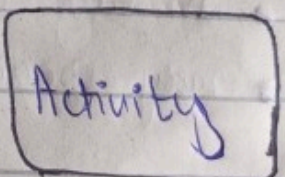
UML Activity Diagram – used to demonstrate the flow of control within the system rather than the implementation.

- It models the concurrent & sequential activities.
- also termed as an object-oriented flowchart. It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram.

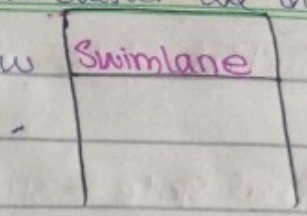
Components

- **Activities** – The categorization of behaviour into one or more actions is termed as activity.

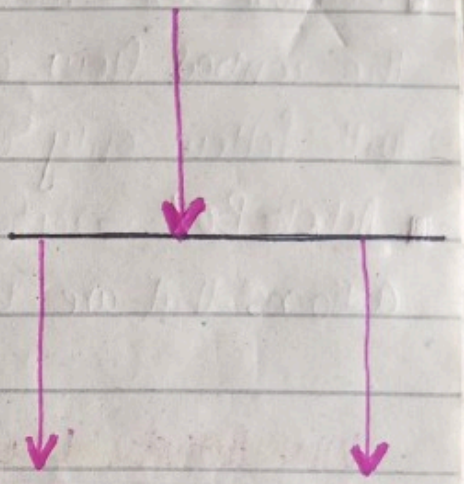
- Activity is network of nodes, connected by edges.
- Edges depict the flow of execution.
- Activities are initiated at the initial node & are terminated at final node.



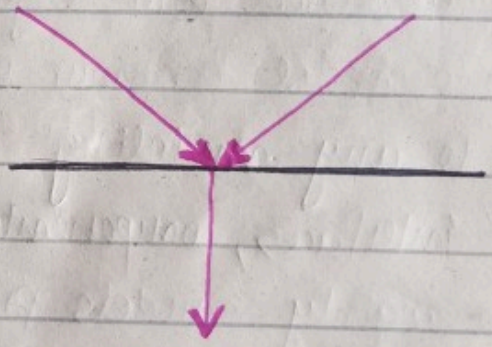
• Activity Partition/Swimlane – It's used to cluster all the related activities in one column or row
– It can be either vertical or horizontal.
– not necessary in diagram.



• Forks – Forks & join nodes generate the concurrent flow inside the activity.
– A fork consists of one inward edge & several outward edges.
– When data is recieved at an inward edge, it gets copied & split crossways various outward edges.
– It splits single inward flow into multiple ~~outward~~ parallel flows.



• Join-nodes
– opposite of fork nodes.
– logical AND operation is performed on all of inward edges as it synchronize the flow of input across one single output (outward) edge.



• Pins
– Small rectangle, which is attached to the action rectangle.
– It clears out all the messy & complicated thing to manage the execution flow of activities.
– It's an obj. node, represents one input to or output from the action.

Notation

- **Initial State** – Initial state or beginning of set of actions.
- **Final State** – stage where all all control flows & object flows end.
- **Decision-Box** – makes sure that the control flow or object flow will follow only one path.
- **Action Box** – represents the sets of actions that are to be performed.

Action 1

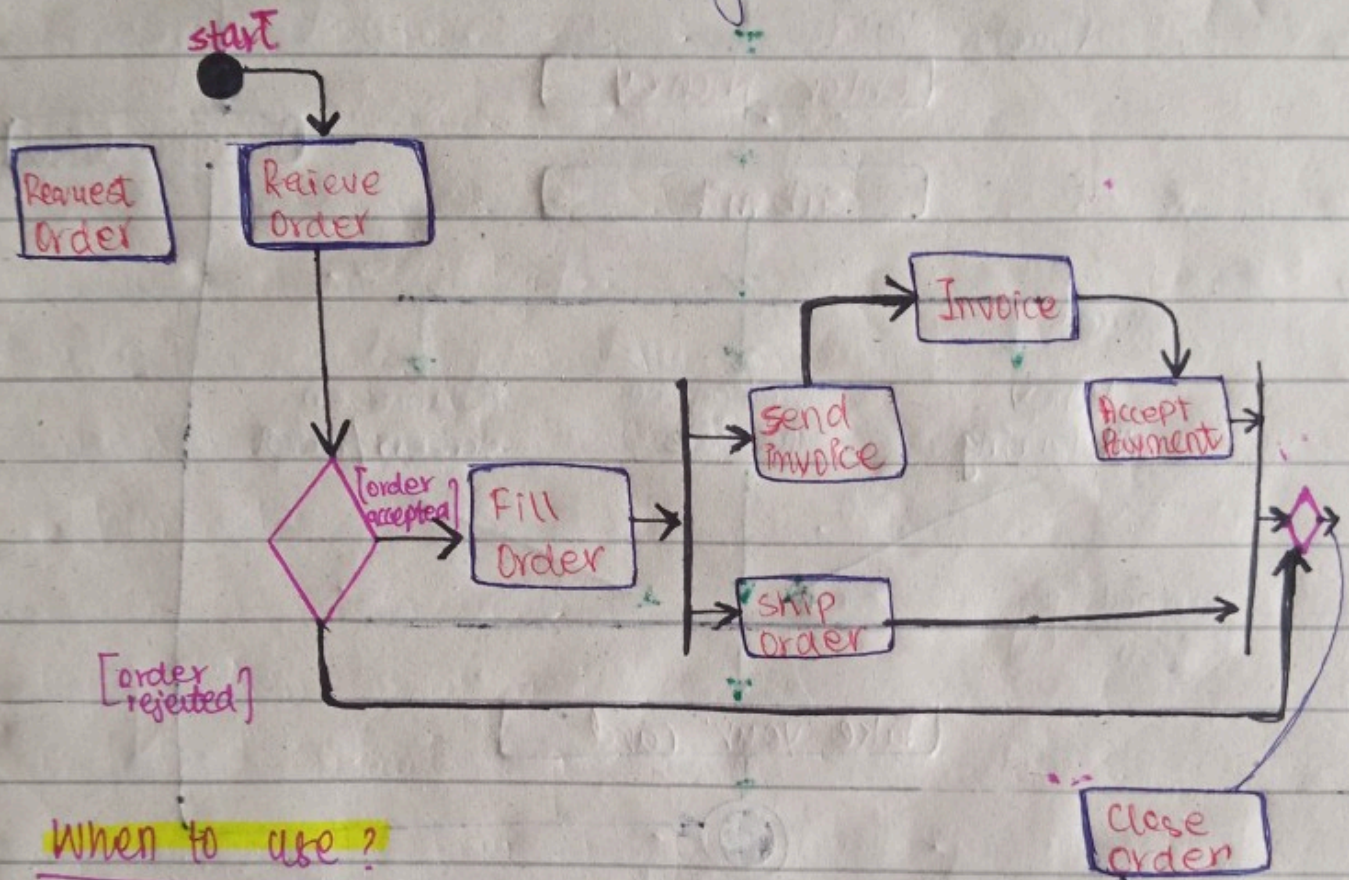
Why Activity Diagram

- An activity diagram encompassing a group of nodes associated with edges.
- To model behaviour of activities, they can be attached to any modeling element, use-case, classes, interfaces, components, collaborations.
- mainly models processes & workflows.
- envision dynamic behaviour of system as well as constructs a runnable system that incorporates forward & reverse engineering.
- message flow is'nt included in activity diagram.
- depicts flow b/t several activities.

Example

example — business flow activity of order processing

→ Input parameter is the Requested order, & once the order is accepted, all of the required info is then filled, Payment is also accepted, & then the order is shipped. It permits order shipment before an invoice is sent or payment is completed.



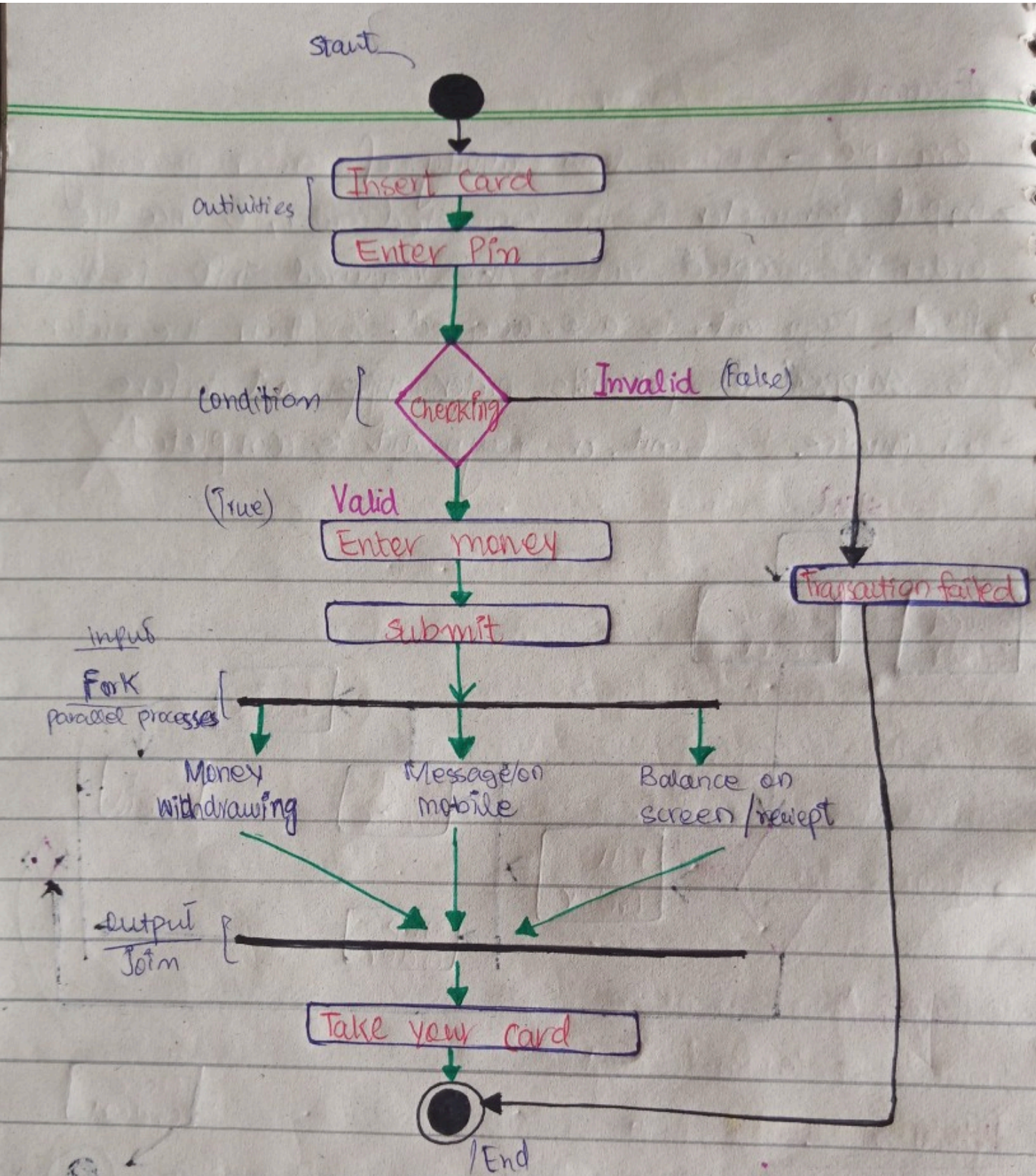
When to use ?

→ To model the execution flow among several activities.

→ To model comprehensive information of a function or an algorithm employed within the system.

→ To envision the dynamic aspect of a system.

→ To represent a high-level view of a distributed or an object-oriented system.


Start
Insert Card
Activities
Enter Pin
Condition
Checking
Invalid (False)
(True)
Valid
Enter money
Transaction failed
Submit
Input
Fork
parallel processes
Money withdrawing
Message on mobile
Balance on screen / receipt
Output
Join
Take your card
End