SDLC
1) Planning
2) Analysis
3) Design
4) Coding
5) Testing
6) maintenance

Waterfall (Classic)
Feasibility study
Requirement Analysis
Design
Coding
Integration
Maintenance

Pros:
1) Base model
2) Simple
3) Small project

Cons:
1) No Feedback
2) No Parallelism
3) High Risk

VShape model:
1) Testing in every phase

Pros:
1) Testing every component
2) progress can tracted
3) Time saving

Cons:
1) Not good for oop
2) No Feedback
3) Risk Analysis

Prototype:
Pros:
1) Customer not clear with idea
2) Throw away model
3) Good for technical and requirement task

Cons:
1) Increase cost

Requirement Gathering → Quick design → Build Prototype → Customer Evaluation → Refine Suggestion → Acceptance → Design → Implementation → Test → maintenance

Incremental:
- Divide project into small parts
- Module by module working
- For large project
- Flexiable to change
- Customer interaction is max

Spiral:
Pros
1) Risk handling
2) Flexible
3) Large project

Cons
1) Complex
2) Time
3) Expensive

objective determination and identify Solution → Risk identification → Develop Prototype next version → Review → [no early lock on requirement]

Evolutionary model:
- Combination of iterative and incremental
- Clear Requirement
- For large projects.
- Requirement not locked

Iterative:
Same as waterfall but feedback from maintenance to others.

Pros:
1) Feedback
2) Small project

Cons:
1) Rigid 2) No parallelism
3) No immidiate change

Agile:
- Latest
- Iterative and incremental
- Emphasize Customer interaction

Scrum
Iterative and development with predefine rules

XP:
Extreme programming
Focus on principle such as simplicity.
Feedback, Coding Review, Testing, Incremental Development.

Sprint:
Scrum breakdown project.

Kanban:
visual management tool for workflow. Visualize work item moving through different stage (Jira)

Pros:
1) Frequent delivery
2) Face to Face interaction
3) Changes
4) Time

Cons:
1) less documentation
2) maintenance is less

Agile method (continue):
Extreme Programing | Scrum | up → Agile modeling

Rapid app development model:
Based on prototyping and iterative model with no or less) specific planning [for time bound projects]

Pros:
1) Feedback at initial stages
2) Process measure through stages
3) easier to accomodate changing requirement

Cons:
1) Highly skill people require to use
2) Customer involvement throughout
3) not every app can be used

Elicit requirements → Modularize requirements → Develop module → Integration of all module → Testing the final Product

SE:
Object identification: identify component within system those could be physical component or software module.

Activity identification: Recognizing various process task, activity within system

Boundary identification: Defining scope, boundary of systems

Waterfall
Basic, Rigid, Inflexible not for real project.

Iterative
Basic Problem is well understood

Prototype
user requirement not clear, costly, High use Reusability involvement

Incremental:
module by module, Easy to test, Requirement clear

Use case description:
Id, Summary, Actors, Goals, Stake holder, Pre Post condition, Basic Alternative flow

CPM:
Slack time = last finish − Early finish
last start = last finish − Duration.
∴ last finish is the late start of next activity.
∴ In case of last activity early finish is the late finish.

ES	D	EF
Activity		
LS	ST	LF

Processor → Activity

Gantt chart:

Sprint calculation:
- multiple done task to points
- then add every sprint point achieved.
- Divide to total number of sprint.

Avg = Story point complete / Total sprint

To check the user story that
user point = AVG / per story points

Function Requirement:
Define Specific requirement that System must meet. eg: user-authentication

NF:
Define the performance and quality attribute of software. eg: performance, Scability,

ER:
◇ : relationship
⬭ : derived
◎ : multivalued
▭ : entity

State:
- Start
- End
- State
- Transition

Activity:
- Start
- End
- Fork
- Decision
- Merge

Fence:
State 1
State 2
State 3
State 4

DFD:
- Process
- data flow
- Data store
- entity

Level:
0 level: Decompose process
1 level: Each process should have only one incoming and outgoing data flow

Work breakdown management:
- Clearly project scope.
- Task management
- Risk management
- Performance monitoring.

Example:
Level 0: Project title
Level 1: major deliverable
1) ——
2) ——
Level 2: Breakdown each major.
1) ——
(1.1) ——
(1.2) ——
Level 3: Further Breakdown of selective task
(1.1) ——
(1.1.1) ——
(1.1.2) ——

Class Diagram:
◇ aggregation
▷ inheritance : — association : → composition
+ public ; - private

Sequence:

CRC:

Class Name	
Responsibility	Collaboration

Decision Table

Conditions	Condition entries
Action	Action entries

Petri nets:
- ○ : nodes
- ▮ : transition
- → : input/output
- • : mark / tokens

Example:
1) ○ → ▮ ○
2) ○ → ▮ ○ ○

Inheritance: Is A
Composition: Has A

RE:
Complete list of everything met customer want.

RE specification:
Requirement as specification How system will work.

RE Task
Inception, Elicitation, Elaboration, Negotiation, specification, validation, management.

RE characteristic:
Correct, Consistent, Unambiguous, Complete, Feasible, Relevent, Testable, Traceable.

Design Concept:
Abstraction: Hiding unnecessary details

Modularity: Breaking System into smaller module

Component based: Build system using reusable and interchangable Software Components.

Decomposition & view: Decompose in small parts reduce complexity Eg web app decompose to front, back end and DB.

Architecture Design: Define overall structure and behaviour of System.

→ **Client-server:** Client interacts with centralized Server for access.

→ **layered architecture:** Component divide in layers each performing its own role.

→ **Microservices:** App build as small independent service which communicate over network

Architechture:
Process of defining hardware and software component and their interfaces to establish the framework for development of computer system.

Type:

Peer to Peer: Each component acts as both client and Server eg Bittorrent, Skype.

PIPE Filter: Stream data for I/O transformation of data (Filter)
→ Pipe
: Data source
→ : pipe
: filter
: Data target

Client - Server: Server offer client uses. (two tier)
○ : client
▭ : server
→ : request / reply

layer System: different architecture style can be used on diff layer.
- UI → App layer
- Interaction
- not easy to use components

Repository:
→ Central Data Store
→ All operation perform on
: Knowledge share
: Blackboard
: Read / write

Publish / subscribe:
- Subscribe to content on network.
- You are notified if event take place

Call & return:
Main program
Controlled

Decomposition & view:
Top level
First level of decomposition
Second level

TYPES:
Functional : Partition function
Feature oriented
Event "
object "

Example:
Student Sys
Data entry valid
Transaction
Report generate
data update
Student data verify
Course data verify
Transaction check
Error
Add

Achieving Quality ATTRIBUTE
Modifiability: Design ease to change

Performance: Response time, Throughput, Load.

Security: Immunity, Resilence

Robustness: mechanism to recover form problem.

Usability: ease in which user is able to use the Software

Bussiness goal: Attributes the system is expected to exhibit.

Realibilty: Fault (human error).

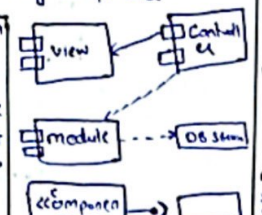
Trade off Analysis:
multiply every priority entry with every other column entry, and do sum. Highest priority will be chosen

Component base Design:
- sub system decomposition:
The breakdown of system into small parts.

logical Component:
Bussiness component at the logic layer without direct physical runtime equivalent.

Physical components:
refer to database server that have explicit runtime equivalent.

Component Diagram:
visualize and show relation among components.



Design methodology:
overall approach to solve customer problem including software architecture design.

Design Principles:
modularity, Interface, Info hiding, Incremental dev, Abstraction, Generality.

modularity:
single responsibility, easy to change, Cohesion + coupling.

Cohesion:
dependency within module
* Coincidental (lowest) → all part of module unrelated
* logical → if/else, calsum/caldiff (ProcessData/ProcessData)
* temporal → By time
* procedural → (mix → Knead → Rise dough)
* Communicational → same data, used by diff. function
* Functional → performs only function which it is designed for any (openfile, readfile)
* Sequential → input of one is output of other (low preffered)
* Informational → grouped to provide some info such as whether present

Coupling:
module depending on each other (low preffered) * Content (lowest) → one module is used in another (changing data)
* Common → data accessible for common data share * Control → one module passes parameter to control behaviour of other module * Stamp → when module complex data structures.
* Data → only data is passed among modules. (Simple data)

Interfaces:
what this unit requires of it environment and what it provide to it.

Info hiding: encapsulation of design hidding data rep → informationally cohesive, hiding algorithm → functionally cohesive.

Incremental Dev:
map out units <<uses>> relation, use graph, Fan in incoming arrows
Fanout → outgoing arrows.

Abstraction:
omit some details to focus on others.

Generality:
universal application software.

Umbrella:
Quality assurance, config management, technical reviews, Project tracking, Risk management.

Integration testing:
Driver:
a write that call a particular component and passes a test case to it. (Bottom up)
Stub:
a special purpose program to simulate the activity of the missing component.
Types:
Big Bang:
all module are combined at once and tested.

Top-Down:


- Only A is tested by itself.
- Stubs of B, C are used at first level
- N-1 stubs required.
- #(non leave) + edge = session

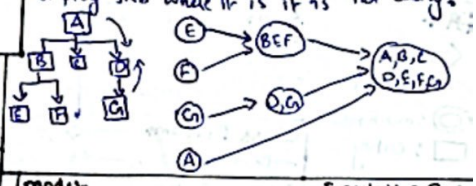
Environment:
S1: A, Stub(B), Stub(C)
S2: A, B, Stub(C)
S3: A, Stub(B), C
S4: A, B, C, Stub(E), Stub(F)
S5: A, B, C, E, Stub(F)
S6: A, B, C, Stub(E), F
S7: A, B, C, E, F

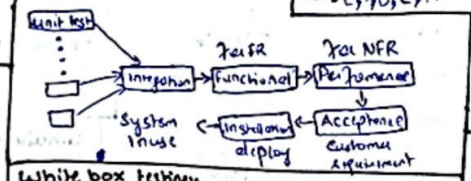
Bottom-up:
same diagram of Top-Down
- Driver are used to call the child function.
- Driver are relatively intelligent
- N-1 leaves driver

- Number of drive = (non leave)
- Number of session = (non-leave) + edge.

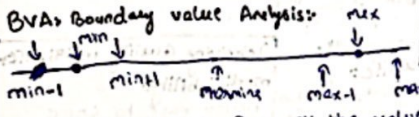
Environment:
S1: E, driver(B)
S2: F, driver(B)
S3: E, F, driver(B)
S4: Ef, B, drive(A)
S5: C, driver(A)
S6: E, F, B, C, A

Sandwich:
- Viewed system in three layer
- Employ BU where using drivers is not costy.
- Employ stub where it is it is not costy.

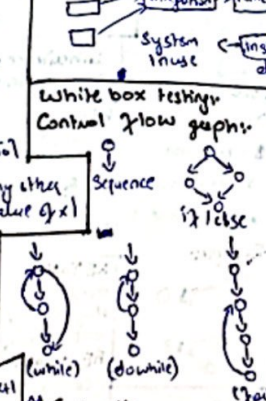




Blackbox testing:
Input range 1-10 Eq: {1...10} {x<1} {x>10}
↳ Equivalence class Partitioning.
Input range 250 Eq: {250} {x<250} {x>250}
Input set: {-2.5, 7.3} Eq: {-2.5, 7.3}, {any other value of x}
Input: if input is Bool, Input true: Eq {true} {false}

BVA: Boundary value Analysis:


All the above graph is for all the value for invalid and valid
6n+1 formula.

White box testing:
Control flow graph:


Cyclomatic complexity:
Edge - Node + 2 if P=1
Edge - Node + 2P if P>1
∴P is connected components

Statement coverage:
The technique involve execution of all statement of source code at least once.
(Execution line / Total line) x 100

Branch Coverage:
(number of branches covered / Total number of edges) x 100
with control flow graph

model:
Requirements complete and frozen (BEP) Expectation of waterfall and
- Iterative value prototyping, systematic aspect of waterfall, evolutionary, risk identification (Spiral
- Product backlog, user stories, sprints (Scrum)
- Visualize window, limiting the amount of work in progress (WIP) (Kanban)
- A phase in software lifecycle (Maintenance)
- Delivery software in timely manner (RAD)
- Quick plan, quick design, quick review lead.
- focus on internal quality such as maintainance (XP)
- Four phase work on almost all discipline in each phase, use of UML and mini project (UP)
- Core product first, Scope largely known (Prototype)
- user stories, test-driven dev, pair programming (Recess (XP)
- Human resource hungry, short cycle time, parallel dev of module (Kanban)
- Respond to change focus more on working software (Agile)
- Reduce cost of change during software dev (Incremental)
- A phase in software lifecycle: (Umbrella Activity)
- user avalible for daily change requirement (Scrum) review of work, frequently
- long wait before running software can be seen (Expectation of waterfall)

Unified Process: (UP)
object - oriented:
4 phase consist of iteration → mini project
1) Inception: approximate vision + rough estimate.
2) Elaboration: Refine vision + basic estimate, implement core architecture
3) Construction: implementation of remaining element,
4) Transition - beta testing deployment.

Functional point Analysis:

Component	Low	Average	High
EI	3	4	6
EIOs	4	5	7
Inquiries (EQ)	3	4	6
(EIFs)	7	10	15
(EOs)	5	7	10

FP = count [0.65 + 0.01 * sum[Fi]]

Estimation loc:
optimistic loc + 4 * most likely + pessimistic / 6

if line of code are given FP * line of code

Estimation Effort:
Effort = line of code / Productivity
= FP Estimation LOC / Productivity

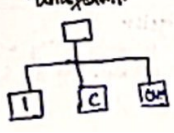
Decision Table:

Component based:
requires → provides
<<component>> part → <<realize>>
→ provides, ⟶ <<uses>>
→ requires.
Decomposition reviews:
functional decompose.

Golden rule of UI:
make interface consistent
- Ensure that similar element behave in similar way
- maintain uniformity in visual design, terminology and layout across the interface.

Place the user in control:
Provide clear Navigation, Allow undo and redo, offer Hotkey, Ensure Direct navigation
Provide immediate feedback.
Reduce memory load:
minimize information overload

User Recognition overall Recall, Provide consistent layout, Simplify Task, offer clear instruction

modeling DFD to Architecture:
Transform:


Transaction: