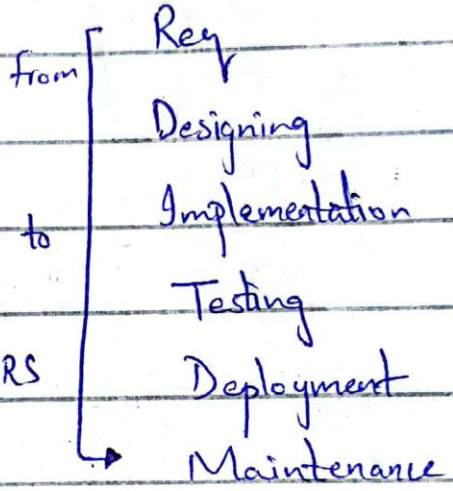
SCD-1 (07/07/23)

1) Software Engineering.

↳ 19% of softwares got rejected due to failure in req gathering.

↳ Understand domain



SCD-2 (08/07/23)

Domain:

↳ This involves req gathering.
System Req / Specification.

SDLC

↳ In this, firstly we find the domain, problem on which we are working.

Req:

User Req:

System Req:/Specification

Non-functional Req:

Waterfall Model:

↳ We follow steps, like firstly gather req then design.

+ point ↳ Each step is performed deeply.

- point ↳ Can't implement where requirements change frequently.

SCD-3 (12/09/23)

Software Engineering — Coding

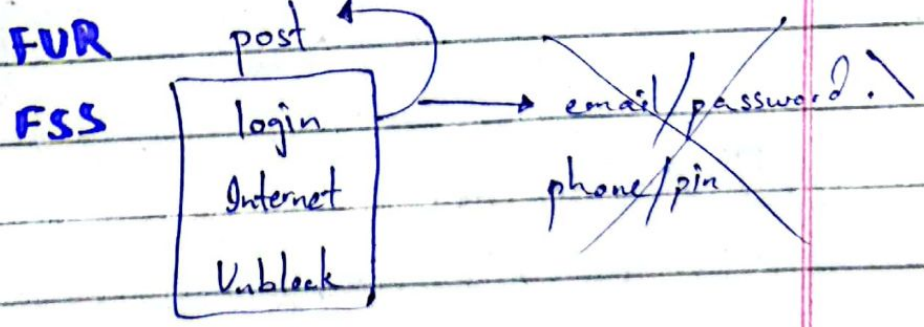
1. Understand Problem Domain.
2. Req Gathering
3. Architecture.
4. Design.

2. Req Gathering

- Functional Req.
  - a) User Req. (Only what is Req).
  - b) System Req. (How req are implemented).

Post or update ← a) User Req.

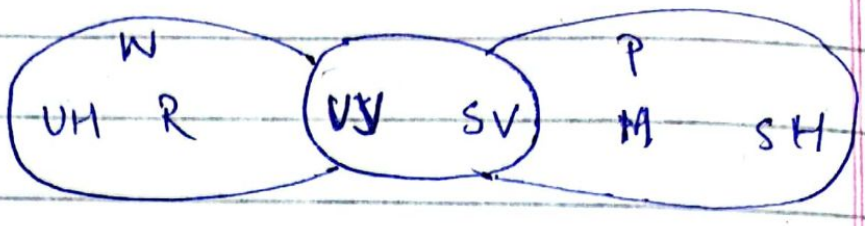
must login to post ← b) System Req.



- Non-Functional Req.
  - Product Req. (OS version, hardware version).
  - Organizational Req. (NADRA verification)

- External Req (External companies req)

WRSPM



U → User S → System.

W → World (Domain).

R → Requirement.

S → Specification.

P → Program.

M → Hardware Machine.

S → R Specification specifies req

P & M → S

③ Architecture.

↳ Decompose an enterprise system into independent sub-systems that have value in the system.

→ How these systems interact?

→ Principles and guidelines for the design evolution over time.

Bad Architecture ⇒ Bad Design.

4. Design.

SCD-4 (13/09/23)

- Component is build by modules.
- Module is not independent.

Persons or companies may own cars. The car owner ID is the ID either the person or the company that owns the car. A car may have only one owner (person or company). A car may have a loan or multiple loans. A bank provides a loan to a person or a company for the purchase of a car. Only the loan owner may

obtain a loan on the car. The car owner type and the loan customer type indicate whether the car owner/loan holder is a person or a company.

- **Make a list of Noun**

Class.

1) Person — name, CNIC, address, ph #
2) Company — name, address.
3) Car — Type ID, owner ID, (P, C). 1 — purchase
4) Loan
5) Bank.

Q: Why the client needs that software?
Q: What problems he/she wants to resolve?

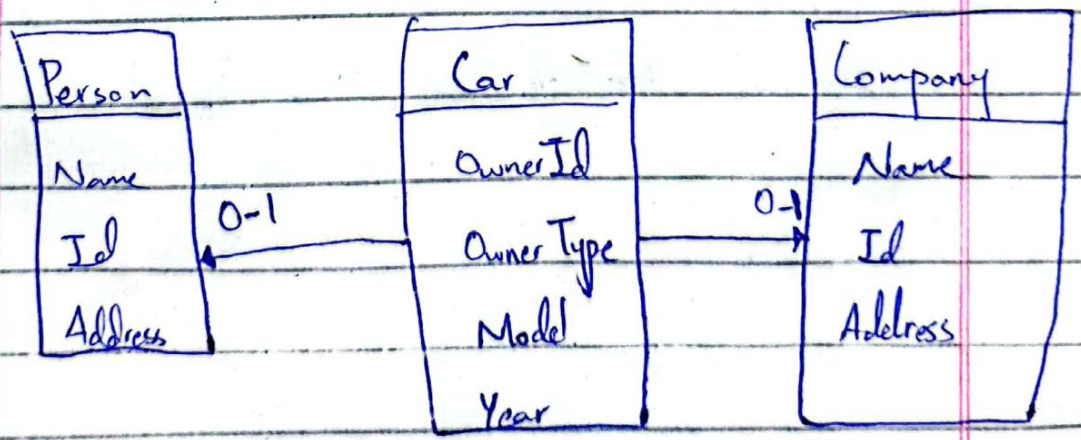
SCD-5 (19/09/23)

Scenario:

(Previous)

→ Nouns are used for classes.

Person	Company	Car	Loan	Bank.
Name	Name	OwnerId	CustomerType	Name
Id	Id	OwnerType	Id	Address
Address	Address	Model	Amount	BankId
		Year	Date	
			AccountId.	



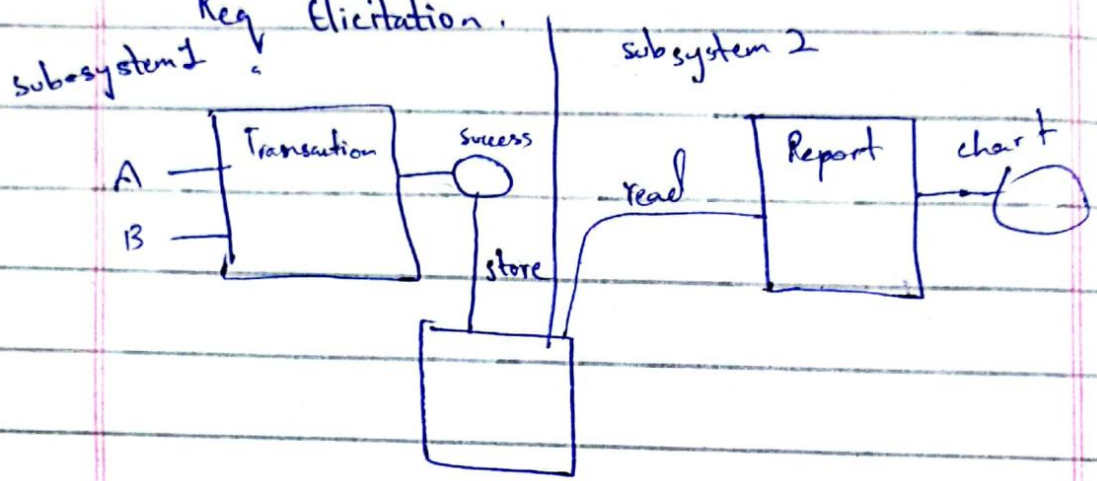
SCD-6 (20/07/23)

Domain Modeling:

Understand the problem

U & E

WRSPM



Software Architecture Models.

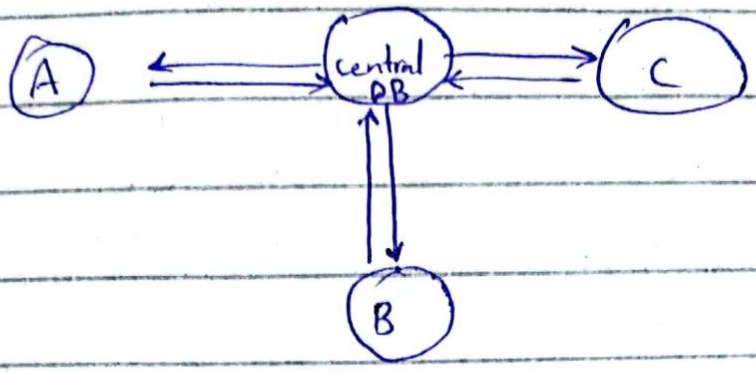
1) Pipe & filter
2) Blackboard.
3) Layered
4) Client server.
5) Event Based.

1) Pipe & filter.

Same input & same output everywhere. Eg compiler.

2) Blackboard.

- Central System.



3) Layered Architecture.

⇒ view, controller.

4) Client - Server

Req → Response.

5) Event - based

Auto response on event.

⇒ similar to blackboard.

Concerns:

System Structure (Decompose into components).

Intraction of components.

Modular Decomposition. (software Design - SD)

Modularity:

Breaking down of a component and reassembling / interaction.

Modules:

1) Coupling (low coupled).
2) Cohesion (highly cohesive)
3) Information Hiding.
4) Data Encapsulation.

SCD-7 (26/07/23)

- Data Encapsulation.
- Information Hiding.
- Coupling. — If change in one class, it effects 2nd class ⇒ tightly coupling.
- Cohesion.

(Lose Coupling is recommend)

Tight Coupling:

Contents Coupling — 1) Module A can directly access Module B data members.

Common Coupling — 2) Module A and B are relied on same global data.

External Coupling — 3) Module reling on externally imposed formed (protocol/interface).

Medium Coupled: (Acceptable)

Control Coupling — 1) Module A controls the logical flow of module B by passing implementation or by using flags.

data structure coupling — 2) Module A & B rely on same ~~components~~ composite data structures. Changing data structure directly affects the other module.

Loose Coupling:

Data Coupling: 1) Module A only pass parameters for requesting functionality of Module B.

Message Coupling: 2) Module A sends messages to Module B.

3) (No Coupling)

Weak Cohesion:

Coincidental: 1) Different parts of module are together just because they are in a single file.

Temporal: 2) Different parts/code/functions are activated at the same time (move to one file).

Procedural: 3) One part follow the other in time.

4) Similar parts/function are grouped. They are similar but perform different things.

Medium Cohesion:

1) All elements operate on similar inputs produces same output.

2) One part output serves as input to other part.

Strong Cohesion

Object
1) Each operation in module can manipulate object attributes.

functional
2) Every part of the function is necessary for execution of single well-defined behaviour

Inheritance
- Domain
- Requirement Elicitation.
- Software Architecture.

Construction
- ④ Software Design.
- ⑤ Coding & Debugging.
- ⑥ Unit Testing.
- ⑦ Integration Testing.
- ⑧ Integration.
- System Testing.

Architecture Maintenance

Construction Phase → 4, 5, 6, 7, 8.

→ Building the software.

→ It takes 30–80% of the time.

Black box Testing

- Done by QA
- whole code is tested.

White Box Testing

We test it on modular approach.

V & V process

Validation & verification

Validation:

We check that in req-elicitation we check the req we gathered have completed all requirements?

Verification.

We hand over the project

to client, we ensure the client is agree on this once got agree it is verified.

Deployment:

- Physical environment.
- Hardware : also he can rollback too.
- Documentation.
- Training.
- DB related concerns.
- 3rd Party software.
- Software.

Failure/Maintenance

Gold Backup.
WARM Standby.
HOT FAILURE

Why construction is formal.

→ All paper work is useless until you don't provide the code in working app, which is done on this phase.

⇒ 30 to 80%.

- Your working code is your actual documentation. So you can tackle most of the by looking at code

⇒ Does your developer know the language.
⇒ What that language provides you.
⇒ low level language ⇒ less performance.

- Key Construction Decision.
  - Choice of programming language.
  - Programming Convention.
  - Current technology wave.

  Selection of Major Construction Practice.

⇒ Your system should be such that if new things are added or changed then the rest should not be disturbed.

⇒ Your code should be reusable.
⇒ What problem is resolved by the software
⇒ Constraints / Business Rule

SCD-8 (27/09/23)

SCD-9 (02/10/23)

Classification while creating software:

⇒ Predictive & Adaptive:

- Predictive : Clear knowledge of what client requires and their is a clear one way path. (Impossible, never done).
- Adaptive : No clear idea what client really wants. (Requirements not clear).

Predictive = Waterfall Model.

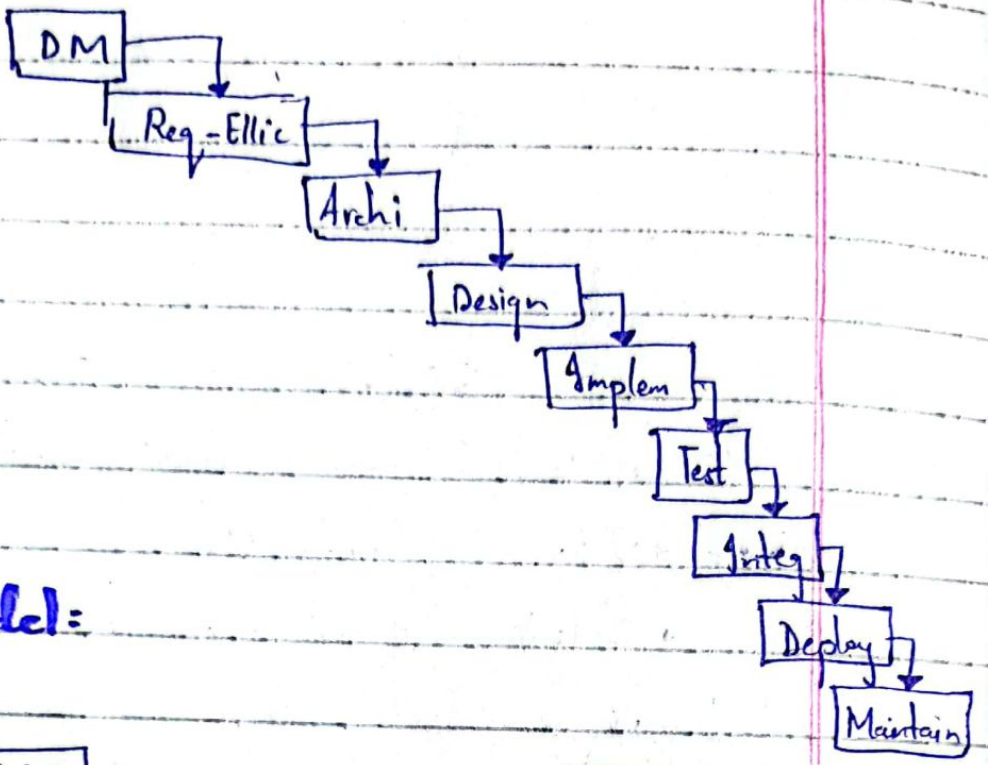
⇒ Iterative & Incremental:

⇒ Incremental : layer-by-layer, module-by-module if feedback then adaptive otherwise predictive.

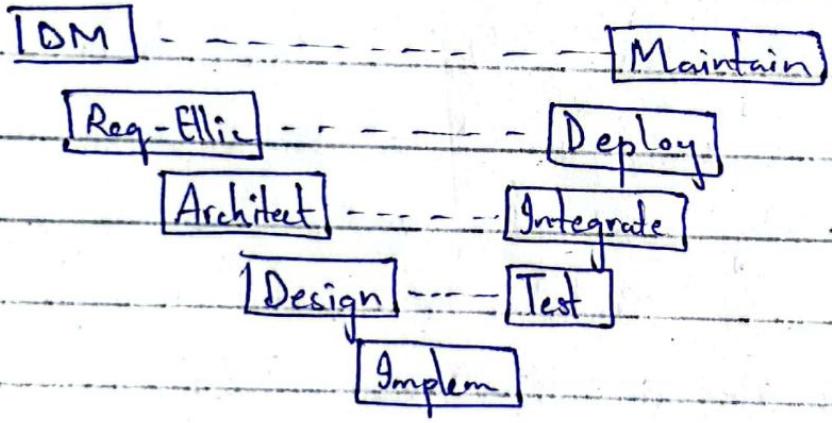
Big things divided into smaller parts.
e.g Bike → Tyre, Engine, Body

⇒ Iterative: Changes with respect to time.

Waterfall.



V-Model:



SASHMI

⇒ At a time multiple phases are running like during modelling it also makes other dept work on them.

⇒ Predictive.

⇒ Next module starts working on

⇒ Client can't wait.

the updated data from previous module as soon as it arrives.

⇒ Don't let the team to be free.

Unified Process.

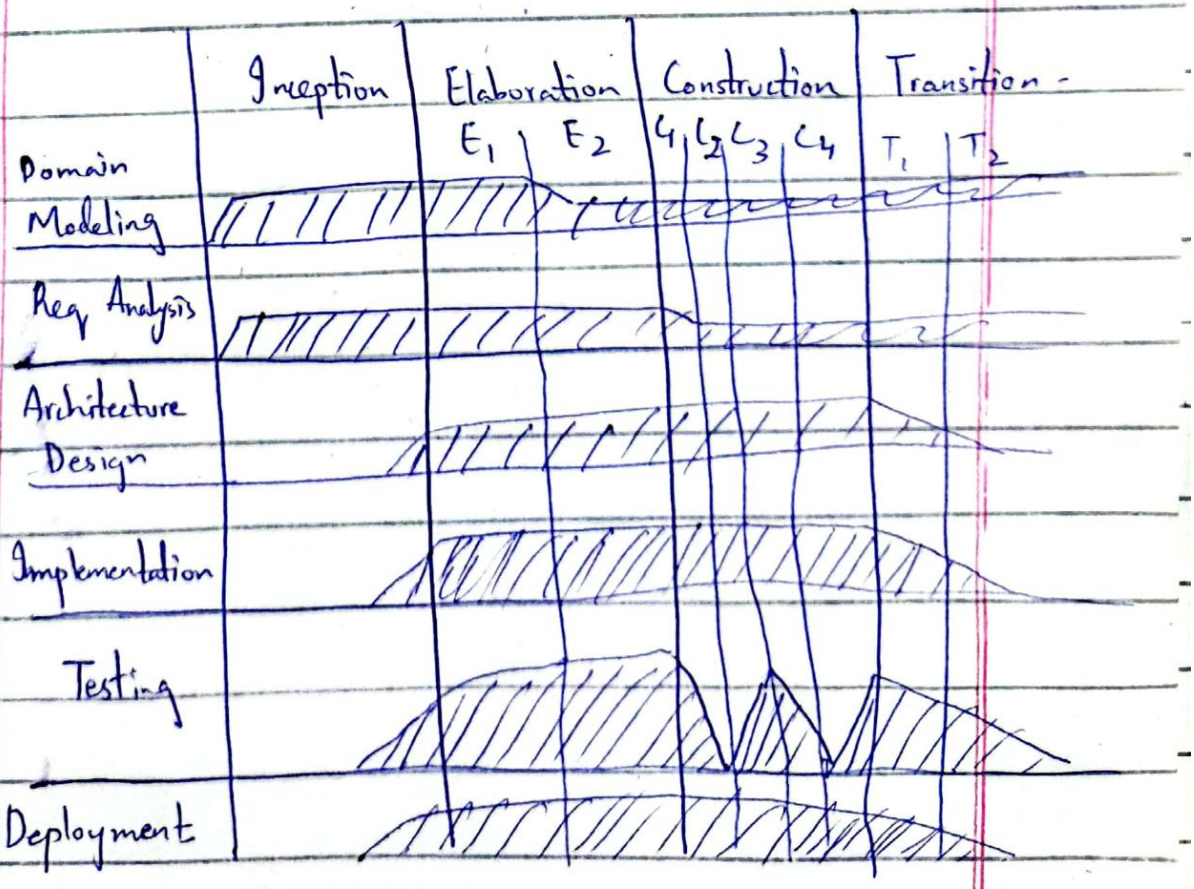
⇒ Rational UP

⇒ Enterprise UP.

SCD-10 (04/10/23)

Unified Process (Architecture Based).

↳ not completely predictive, not completely adaptive (mid).



⇒ How the phases.

⇒ framework.

Cons ⇒ Specifies Architecture.

⇒ Not a model, or iterative etc.

Cons ⇒ Implementation is difficult.

Spiral Model:

Define (define Objective)	Identify
Planning.	Implement/ Test.

⇒ for risk Analysis.

GATE:

⇒ Check after each phase/stage.

1) Should we go to next step?
2) Profitable?
3) Should we stop development?

(Is the model we are developing available in market at cheap cost?)

Cons:

Development team gets demotivated.

SCD-11 (10/10/23)

Agile Methodology:

4 values

① Individuals & Interactions over Process & Tools.

② Working software over Comprehensive documentation.

③ Customer Collaboration over Contract Negotiation.

④ Respond to change

12 Agile Principles.

SCRUM model.

↳ 70% of agile method.

Models of Agile

DSDM

FDD

SCRUM

Crystal

XP

Custom

Lean.

12 Principles of SCRUM

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

2) Welcome changing requirements even late in development. Agile process harness change for the customer's competitive advantage.

3) Deliver working software from couple of weeks to couple of months with the preference to the shorter timescale.

4) Business people and developers must work together daily through the project.

5) Build the projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.

6) The most efficient and effective method of conveying information to and within a development team is face to face conversation.

7) Working software is the primary measure of progress.

8) Agile processes promote sustainable development. The sponsers, developers, and should be able to maintain a constant pace indefinitely.

9) Continuous attention to technical excellence and good design enhances agility cost of exploration.

10) Similarity, the art of maximizing the amout of work not done is essential.

11) The best architectures, requirements &

designs emerge from self-organizing teams.

12) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Personal Software Process

A self improvement approach for software engineers.

① Bottom up approach to practice and improve engineering practice.

② Starting point is by training individuals skill and tools for work.

③ The improvement principles are not just only for software industry but neutral for all industries.

IDEA

① Individuals should be able to plan, deliver, monitor or improve the quality and timelines of their own work.

② Use data to justify refute unreasonable demands -

say "Yes" with confidence.

say "No" with data & options.

③ Learn from experience: use data from one piece of work to improve the next.

Principles (Personal Software Principles).

Measure Staff

Size

Time.

Effort.

Defects (time included in finding solution and point of introduction)

Measure Consistencies

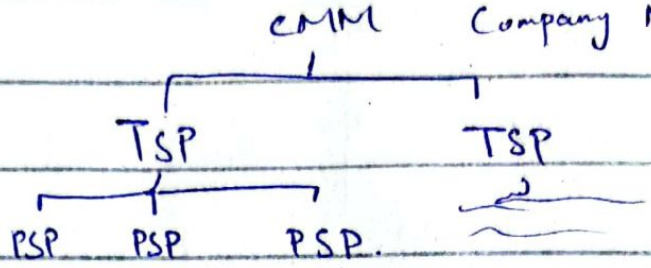
Use corelation to judge usefulness of time to predict future performance.

SCD-12 (11/10/23)

Team Software Process. (TSP)

Humphrey's

Personal Software Process (PSP)



Humphrey's

Idea

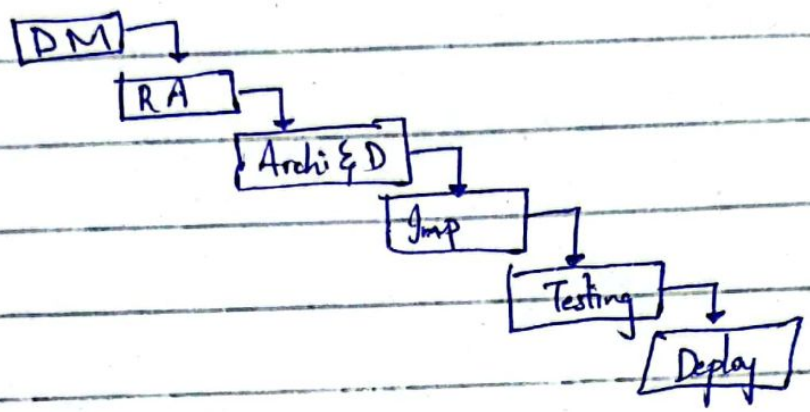
- The team should be self directed.
- Definite tasks assign to team individuals that plays role in achieving a single goal.
- Communication.
- Team members are dependent to achieve a common goal.

Principles

- Measure the task for each individual/analyze each member performance

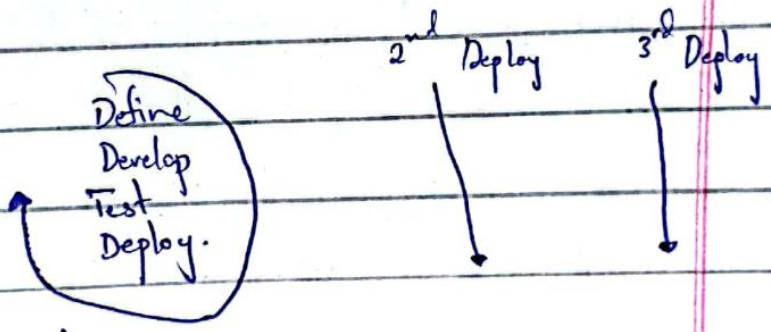
- Regular meetings.
- Rigourous planning. (Achievable goals).

Waterfall



SCRUM

Backlog.



1-4 weeks.

Product
1) Backlog
2) Sprint Planning Meeting.
3) Sprint Backlog.

Daily Meetings
- What previous Task
- Problems.
- Assignment.
- Time - Team Member.

TSP

↳ You have a script. If problem arise then you follow that script.

4) Finish Product.

5) Sprint Review.

6) Burn up/down clients.

SCD-13 (17/10/23)

Error: Illegal operations that results in abnormal malfunctioning of a program.

→ syntax error.

⇒ logical error

⇒ runtime error/unchecked error.

Non functional attributes.

→ Correctness

⇒ Robustness

Ability to handle many inputs. (correct/incorrect).

Technique Approaches

→ Descriptive / Corrective.

→ Active / Passive.

↙ frontend — before service call.

↘ backend — after service call.

Techniques

- Returning a neutral value
- Substitute the next valid value.
- logs & time stamps.
- Error codes / messages.
- shut down.

Exception: Undesirable situation that can arise in the program.

- Checked — compiler can recognise
- Unchecked. (Incorrect input) — Compiler can't recognise.

→ try catch

→ throw

→ throws.

try → check a set of instructions that can cause exception.

catch → handling routines/set of instructions for the exceptions if they occur.

throw → (throw new missMatchException(" ")).

↳ through the exception (it written in try catch, it can be handled).

throws → can't handle, itself, its caller handle.

↳ with class signature.

↳ throw multiple exception.

try
catch
final
↳ must execute.

Fault Tolerance

Collection of techniques that increase software reliability by detecting errors and then recovering from them if possible or containing their effects of recovery if possible.

1. Backup.
   - ↳ different from load balancing.
   - ↳ backup after 2, 3 days/week.
2. Retrying
   - ↳ Retry before reporting fault.
3. Auxilary Code.
   - ↳ (time tracking of code).
   - ↳ log files.
4. Voting Algorithm.
5. Replacing erroneous input with phony input. (fake)
6. Shutdown & restart.

SCD - 14 24-10

Design

- is a sloppy process.
- is a wicked problem.
- is about trade off, priorities and restrictions.
- is non-deterministic & it is a heuristic process.
- is unurgent.

Characteristics

- Minimal Complexity.
- Easily Maintainable.
- Loose Coupling.
- Extensibility.
- Reusability.
- High fan in: a class is used by many other classes.
- Low to medium fan-out: a classes uses low to medium number of other classes.
- Portability

- Leanness.
- Stratification (sorted, format).
- Keep levels of decomposition stratified so that you can view the system at any single level and get a consistent view.
- Standard Techniques.

Principles

- Abstraction: hiding implementation details
  - ↳ Abstract class, interface.
- Encapsulation: Attributes & functions are contained in a single object.
  - ↳ access ⇒ info hiding, from other classes.
- Polymorphism:
  - ↳ overloading / overriding.
- Inheritance:
- Modularity: Breaking down of a component & reassembling it.

Design Measures

- coupling
- cohesion.
- info hiding.
- separation of concern.

SOLID

S – Singe Responsibility.

O – Open Close

L – Liskov Substitution.

Every subclass should be substitutable for their parent class.

(child class uses 100% of parent class)

I – Interface Segregation.

An client should never be forced to implement an interface that it doesn't use.

D – Dependency Inversion

High level class should not depend upon low level classes. instead both use abstraction.