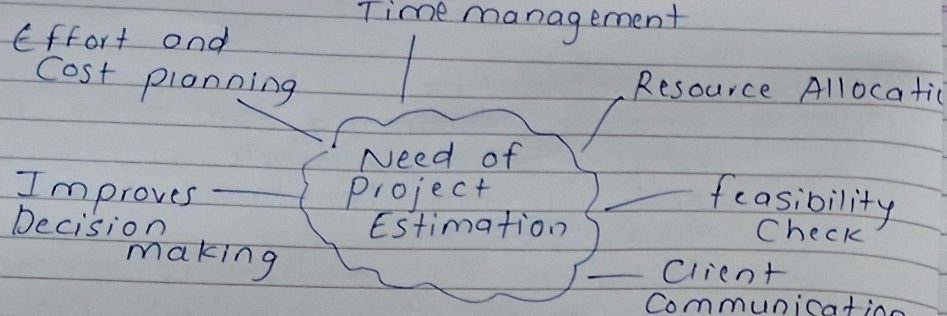
Software Engineering

Unit-3

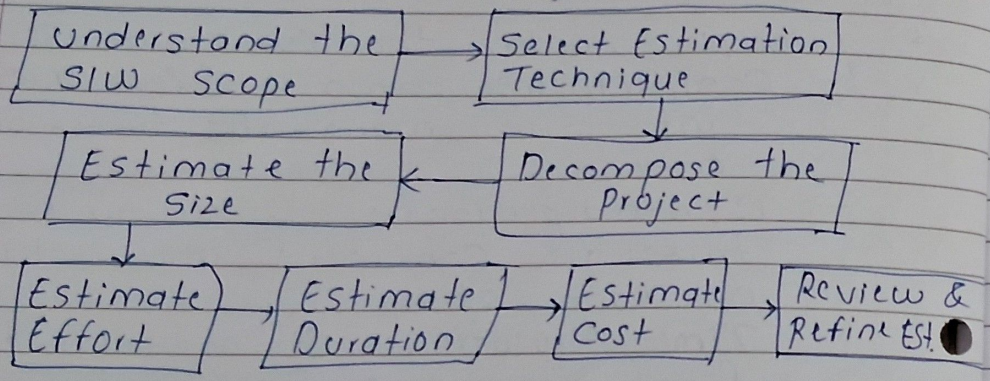
* project Scheduling :
  - involves separating total work in a project into separate activities and judging the time required by these activities
  - Activity that distributes estimated effort across the planned project duration by specific allocation of effort to specific task

* principles guiding the slw project Scheduling
  - Compartmentalization : Break project into small manageable tasks and actions
  - Interdependancy : Understand how tasks are connected and depend on each other
  - Time Allocation : Giving each task a fixed amount of time
  - Effort validation : make sure not assigning more people than available.
  - Defined Responsibilities : Assign each task to a specific team and person.
  - Defined outcomes : know what result is expected from each task.
  - Defined milestones : set to track progress.

*



* Steps while Estimating Software:



* LOC (Lines of code) Estimation:
  - No. of lines of Source Code in SLW
  - oldest and simplest techniques used to estimate the Size, effort and Cost.

Effort = LOC / Productivity
Cost = Effort x Cost per person-day
Time = Effort / No. of persons.

* FP (Function Point) Based Estimation:
  - measures SLW functionality from the user's perspective.
  - Independant of programming language
  - FP Calculated based on:

[External Inputs] + [External outputs] + [External Enquiry]

+ [Internal logical files] + [External Interface files] = UFP

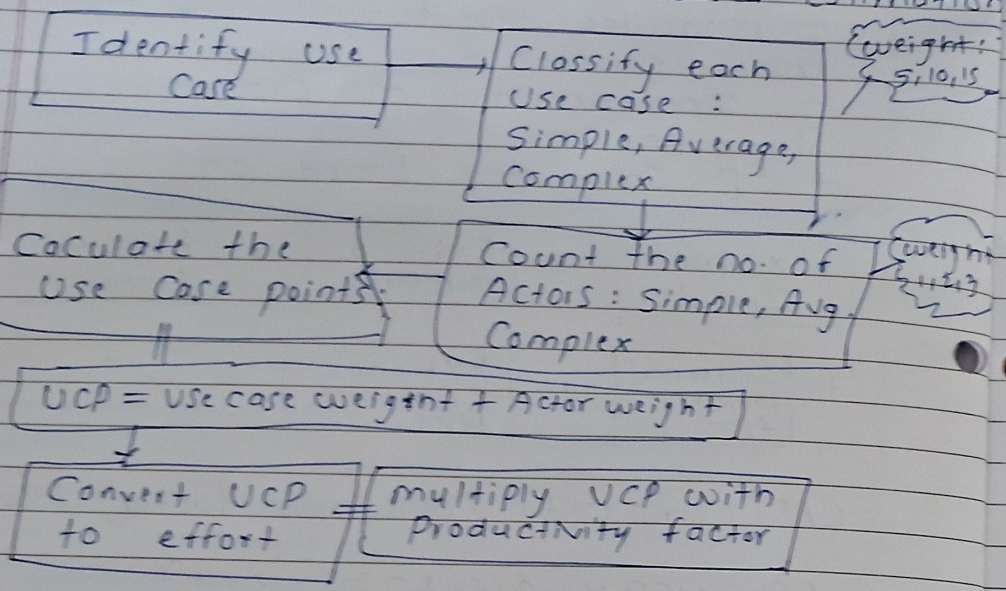
CAF (Complexity Adjustment factor) Assume (30)

FP = UFP x (0.65 + 0.01 x CAF)

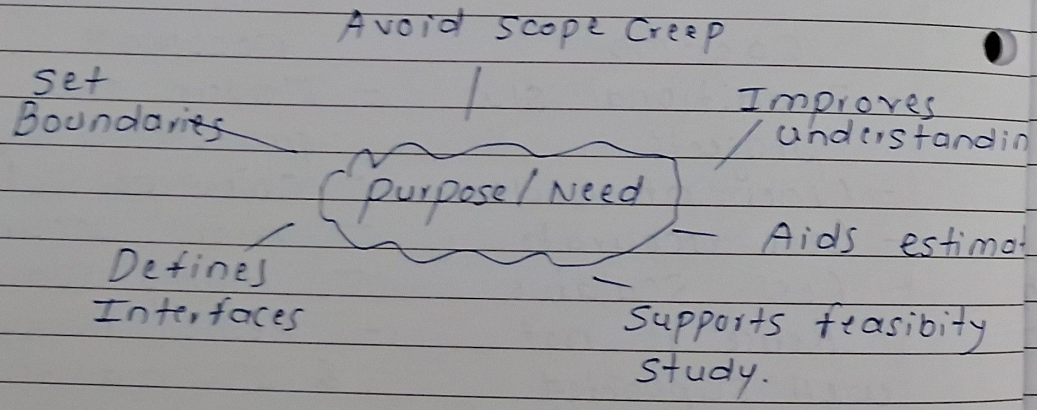
* COCOMO Vs. COCOMO II

COCOMO	COCOMO II
- Traditional SLW projects	- Modern SLW projects
- Lines of Code	- Object point, fun. point
- Less accurate	- more accurate
- Less Scalable	- Highly Scalable
- Reuse is not well supported	- Reuse, reengineering & SLW maintenance is included
- Organic, Semi-detached, Embedded projects	- Application Composition, Early design, post-Architecture projects.

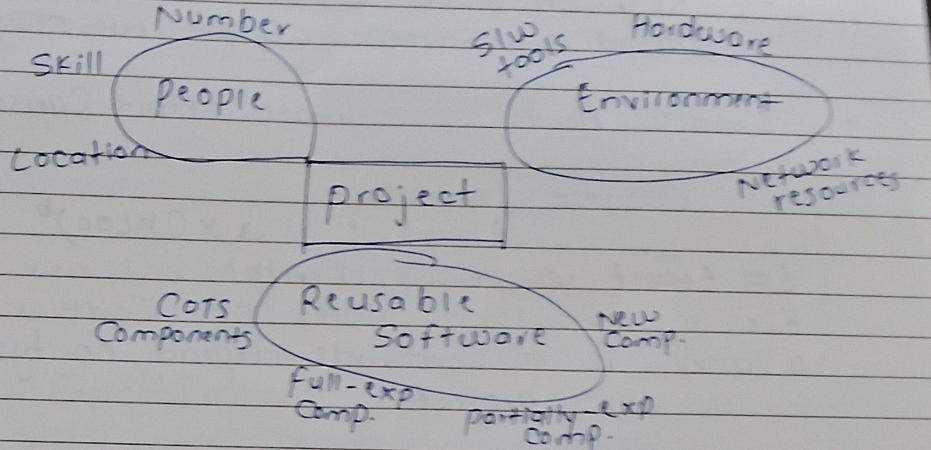
* Steps to perform use case based estimation



Need to define a Software Scope:



* Categories of SLW engineering Resources (Project Resources) :



* Tasks of project scheduling

1. Defining a task set
List all major Activities & Sub tasks

2. Task Sequencing
Arrange tasks in order which they must be performed

3. Time Allocation
Assign start & end date to each task

4. milestone Definition
Define imp. Checkpoints or milestones

5. Assigining Resources
Allocate Human, SLW, HLW resources for each task

6. Review & Assignment
monitor progress & adjust schedule if delays / changes occur.

* Empirical Estimation models
  - Uses real project data & formulas to estimate effort
  - Rely on Loc or FP
  - Based on samples of previous slw proje

* CocoMo model to estimate effort (person-mon

CocoMo formula :

Effort (E) = $a \times (KLoc)^b$

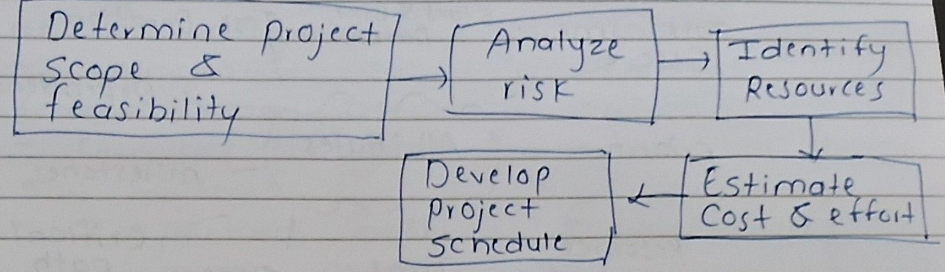
E = Effort in person months
KLoc = Thousand Lines of Code
a and b = Constands depend on type of project

project type	a	b
small	2.4	1.05
medium	3.0	1.12
Complex	3.6	1.20

A slw project is expected to be 25 kLoc and falls under small category

Effort = $2.4 \times (25)^{1.05}$
= 64.85 ≈ 65 person-month

* Activities during slw project planning:

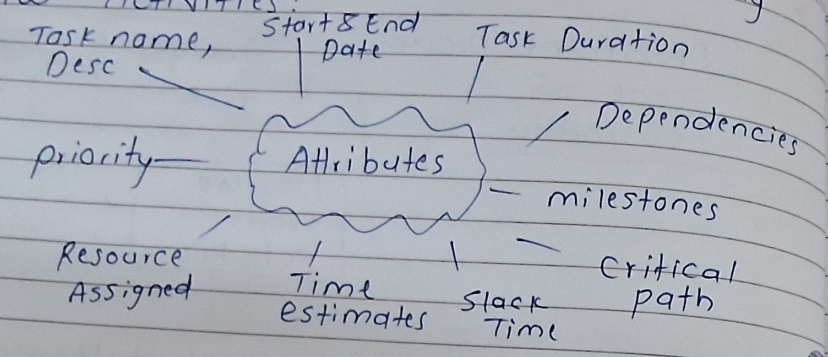


* Open source tools for scheduling slw activities:

Gantt Project
- Uses gantt charts to schedule tasks
- Allows for task dependencies and milestones, resource management
- Gantt chart for visualization of project schedules
- Resource Allocation and management
- Export to PNG, Pdf
- Use case: Idle for managing medium size slw projects

Open project
- open source project management slw
- Supports Agile and Traditional project management methods.
- provides scheduling, task tracking and team Collaboration features
- Gantt charts for planning & scheduling
- Scrum & Agile boards for task tracking
- Time tracking and budgeting tools
- Use case: well-suited for teams with Agile or traditional methodologies

* Attributes to Consider while scheduling slw Activities:



Unit - 4

* Design Concepts:

1) Abstraction:
- ability to cope up with Complexity
- slw design occurs at diff. level of a
- abstraction to apply for refining slw s
- Higher level: broad terms, lower level: detail
- procedural abstraction and data abstraction

2) Modularity:
- slw is divided into separately named and addressable Components called as modules.
- Divide and Conquer strategy.
- problem is divided into smaller subproblems and solutions to these is obtained.

3) Architecture
- Representation of overall system structure of an integrated system.
- Components called as elements interact and uses the data structures
- provides the basic framework for the slw system.
- Important activities conducted in systematic manner.

4) Refinement
- Explaining things in more detail
- start with general idea and break it down into smaller and detailed parts
- Abstraction hides low-level details, Refinement adds those details step by step.

5) Design pattern:
- Pattern is named & useful idea that describes a proven solution to a common problem
- Ready-made solution that can be used when a similar type of problem comes up in Software
- pattern can be reused in other projects

6) Information hiding:
- important property of effective modular design
- modules designed such that information ine one module cannot be accessed by other modules.

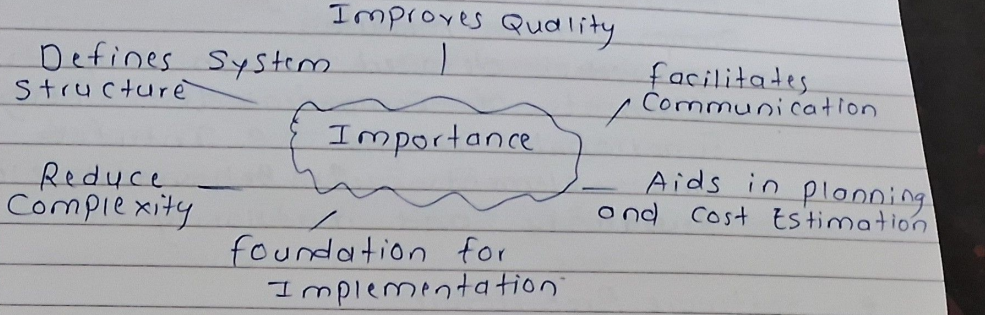
7) Functional Independance:
- Achieved by developing the functional modules with single-minded approach.
- Using func. Independance functions are Compartmentalized and interface are simplified
- Independant modules are easier to main-
- Functional independence is asse using two qualitative Criteria - Cohesion, Couplin

* Cohesion:
- means how closely related the tasks inside a single module are
- performs one task with minimum interaction with other modules
- supports information hiding
- Types: Cogical, Temporal, Procedural

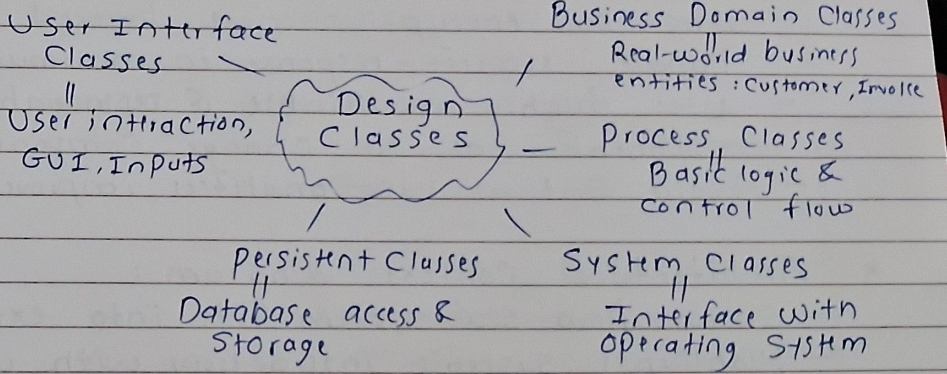
* Coupling:
- means how strongly one module is connected to other
- measure of interdependance betn modules
- Lower coupling = better design.
- Types: Data, Control, Comman, Content

8) Refactoring:
- Simplifying the design without changing the function or behaviour
- process of changing SIW System in such a way that external behavior not changes however internal structure is improved.

* Importance of SIW design:



* Types of Design Classes:

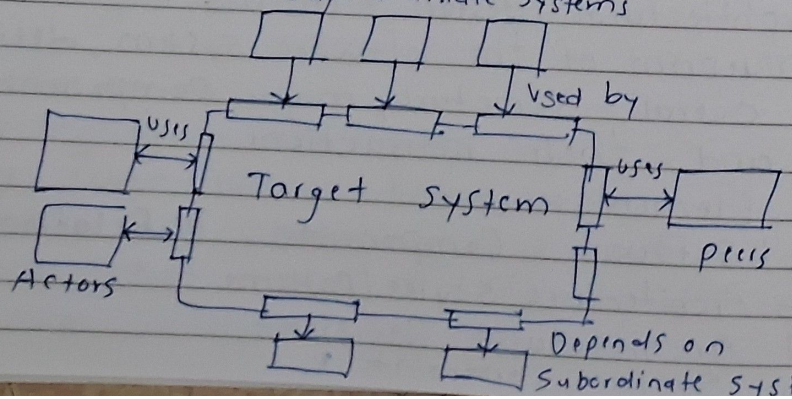


* Architectural design elements
- blueprint of the software system, defines overall structure, major components and their interactions
- Elements:
  1. Software Components
  2. Relationships
  3. Architecture styles / patterns
  4. Architectural Decisions
  5. System Constraints

* Component Level Design Elements
- focuses on internal logic for each component or module
- provides detailed design to guide dev
- Elements of Component Level Design:
1. Functional elements 2. Interface elem
3. Data elements 4. Behavioral element
5. Reusability and modularity.

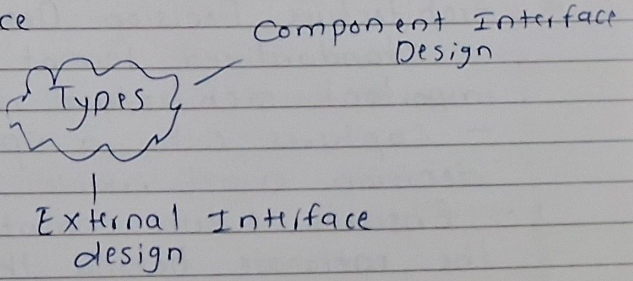
* Software Architecture:
- Structure of program modules where they interact with other.
- System is secured against malicious users by encryption and layered archit
- Handle request-response in minimum time
- uses highly modifiable & replaceable components, easy to change Components
- Avoid Critical functionalities, improve Comm

* Architectural Context diagram:
- Shows how SW system fits into external e
- Highlights systems interactions with users, other systems, database, devices etc. external int

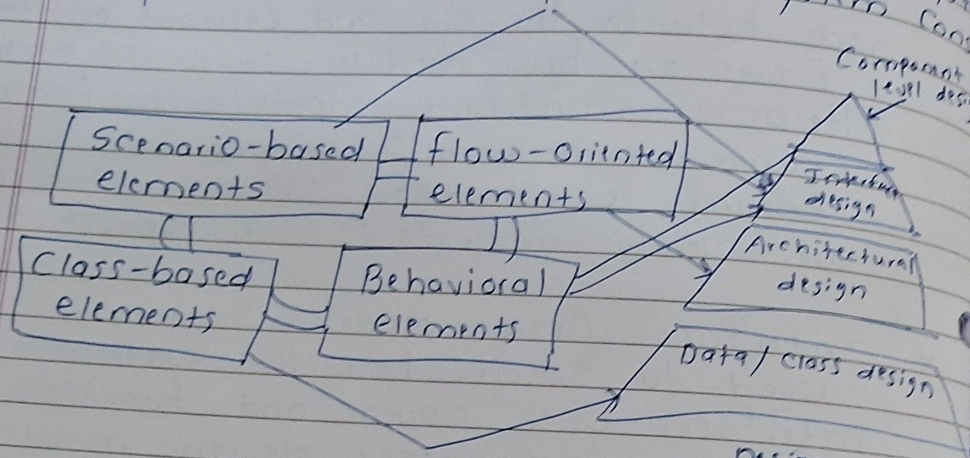


* Interface Analysis:
- process of identifying and defining the interactions between Software Components and between the SW and external entities
- Focus on how modules comm. w/ each other
- Helps in identifying inputs/outpus/data exchanged between Components
- Ensures user/hardware/software interface are clearly understood and specified.

* Interface Design Models:
- used to specify how the systems interfaces will appear and function.
- Types of interface design models:

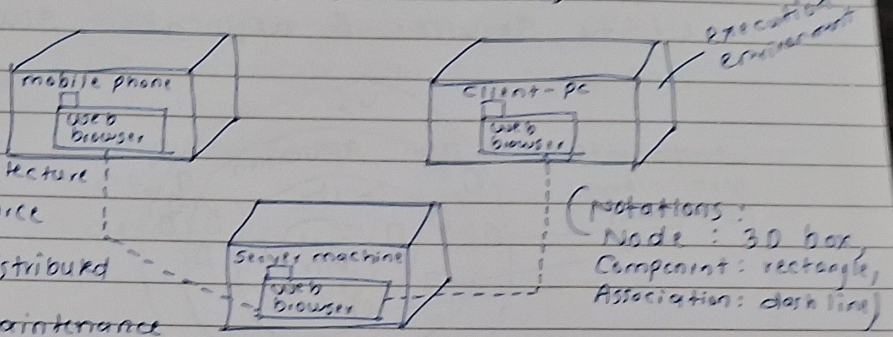


* Translate requirements model into design mo
- Involves Converting user needs and Syst
requirements into blueprint for system Cons

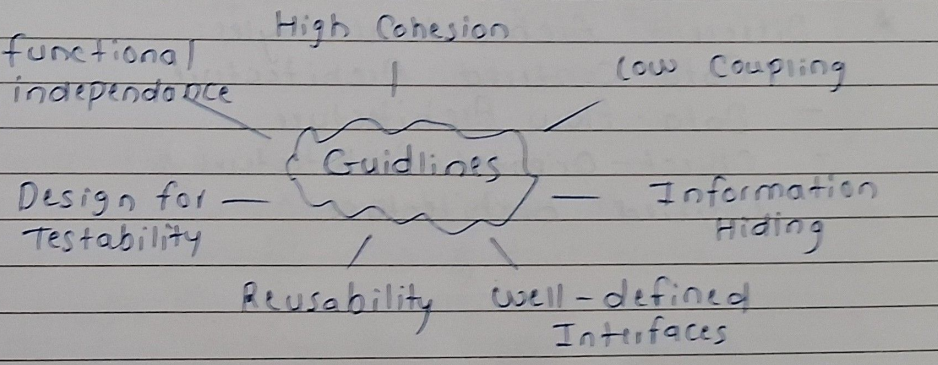


* Architecture Decision Description Template
- standardized format used to document important architectural decisions.
- Captures why a specific Architectural decision was made
- Ensures future maintainers understand the rationale behind the system's structure
- Typical fields in the Template:
Title, Context, Decision, Rationale (why the decision was made), Implications (effect on other components), Alternatives, Related decision
Author/Date

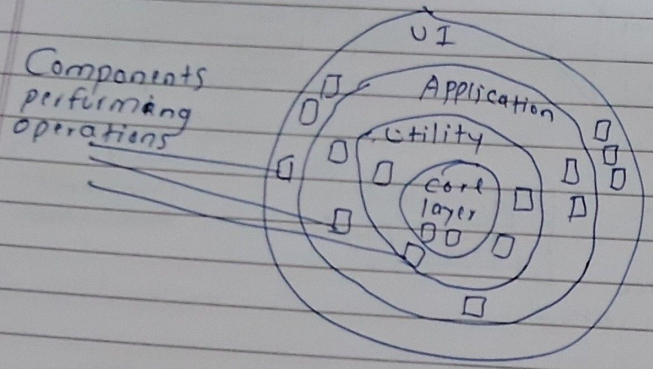
* Deployment level design elements:
- Indicate how sw functions and sw subsystems are assigned to the physical computing environment
- for example web browsers may work in mobile phoned or they may run on client PC or Server machines



* Guidelines for Component level design

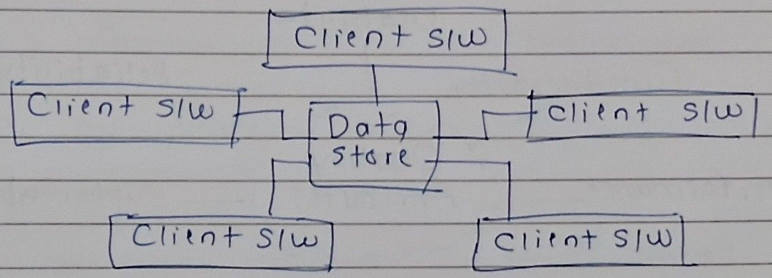


* Layered System Architecture:
- Composed of different layers, Each layer is intended to perform specific operations
- Outer layer for performing user interface operations, inner layer perform operating system interfaces
- Components in intermediate layers perform utility services & application s/w functions



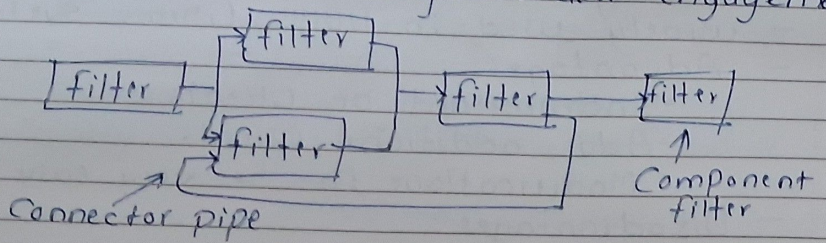
* Different Architectural Styles:
- Data-Centered Architecture
- Data-Flow Architecture
- Object-Oriented Architecture
- Layered architecture.

* Data-Centered Architecture:
- Data is stored at a centralized location & can be accessed by everyone frequently
- widely used in DBMS, Library systems etc.
- Advantages:
  - Independant of clients
  - Add additional clients
  - Modifications can be very easy
- Disadvantages:
  - Duplication or replication is possible
  - Changes in data structure highly affect the clients

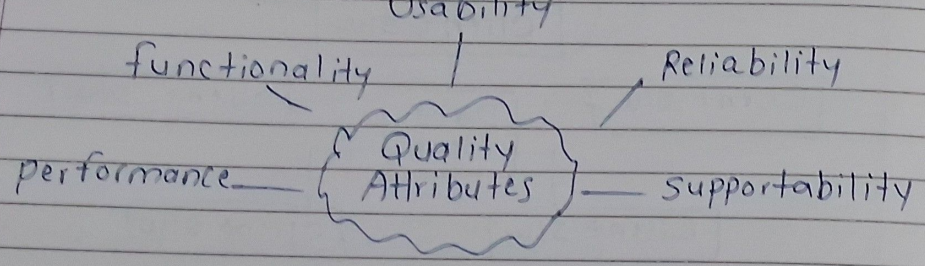


* Data-flow Architecture:
- Used when input data is needed to be transformed into output data
- pipe is a connector that passes the data directionally from one filter to another.
- Filter is a component that reads data from its input pipes & performs its functions.

- Advantages:
  - Concurrent execution is supported
- Disadvantages:
  - Does not allow greater user engagement

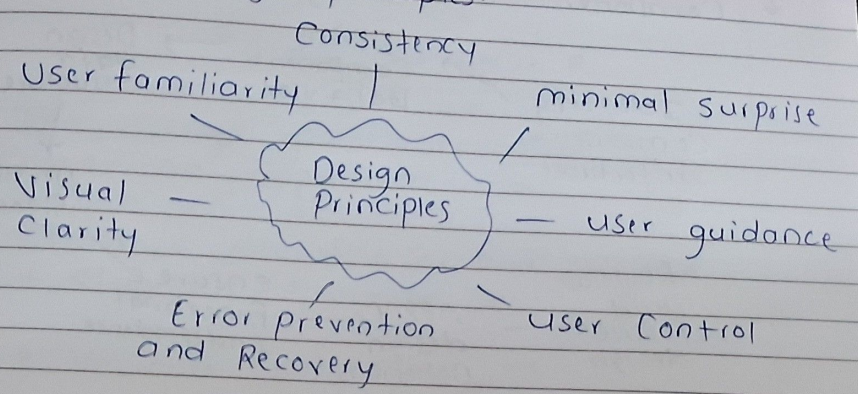


* Software design quality attributes:



* guidelines of slw design quality are:
  - Design using recognizable architectural style
  - Design should be implemented in evolution manner.
  - Representation of data, Architecture, interface and Components must be distinct
  - Components must show the independant functional characteristics
  - Design of slw must be modular.

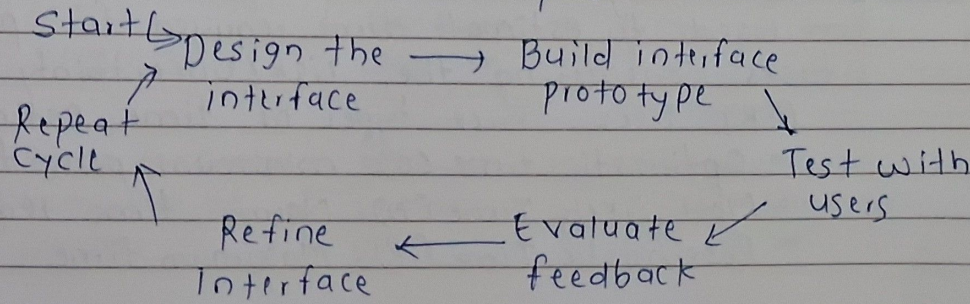
* Interface design principles:



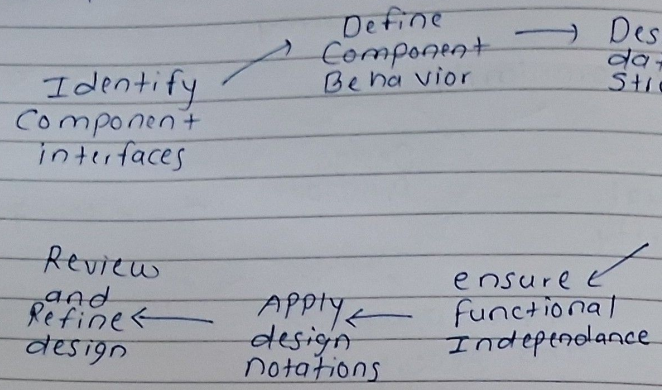
* Abstraction Vs. Refinement

Abstraction	Refinement
◦ Hide Complexity	◦ Reveal details
◦ Top-down approach	◦ stepwise elaboration
◦ Simplicity, general view	◦ precision, implementation level detail
◦ Used to understand the System	◦ Used to build the System.

* Interface evaluation Cycle:



* Component-Level Design steps:



Unit-5

* steps involved in risk planning

Risk Identification → Risk projection → Risk Refinement → Risk mitigation, Monitoring, managgement → Preparing RMMM plan

* PERT - Program Evaluation and Review technic
  - Used to estimate time required for project tasks Considering the risk/uncertainty
  - PERT uses three types of time estimates:
    - Optimistic time (O): minimum possi. time
    - Most likely Time (M): Normal time required
    - Pessimistic Time (P): Maximum time.

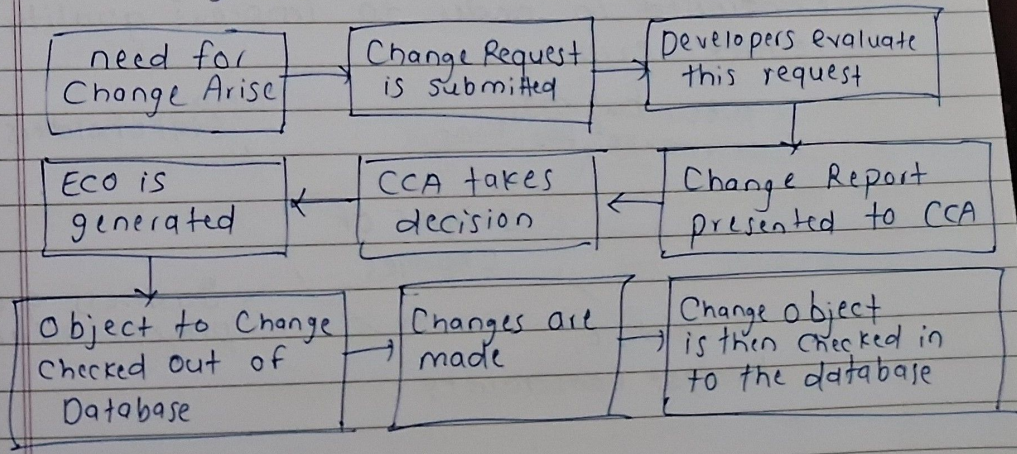
Expected time calculation:

$$TE = \frac{O + 4M + P}{6}$$

This average gives a weighted time estimate based on impact of risk

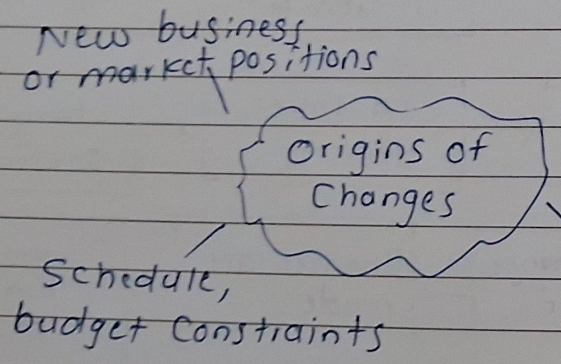
Standard deviation (σ) and variance
- Standard Deviation (σ): $\frac{P-O}{6}$
- Variance (σ^2): $\left(\frac{P-O}{6}\right)^2$

* Change Control:
  - Changes in SW project is vital
  - introducing small changes in the system may lead to big problems in product.
  - Similarly, introducing changes may enhance Capabilities of the System.
  - Change Control process:

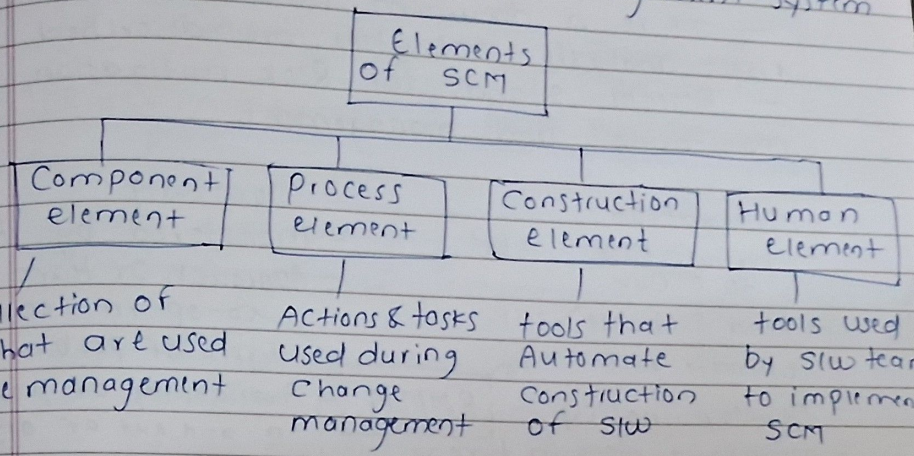


* Version Control:
  - Version is an instance of system which is functionally distinct in some way from other system instances
  - Help manage different versions of Configuration items during development p
  - Attributes associated with slw version: 'date', 'Creator', 'Customer', 'Status'
  - Each version needs associated name for easy reference.
  - Each version of slw system is a collec of slw configuration items.

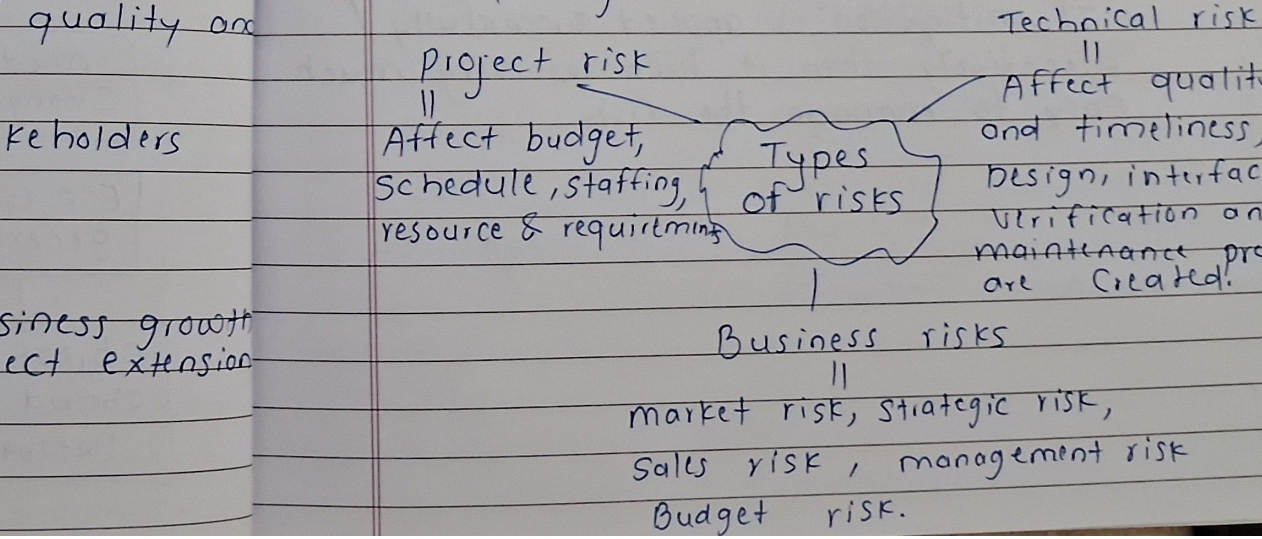
* Software Configuration management (SCM)
  - It is set of activities carried out for identifying, organizing and Controlling Char throughout the lifecycle of Computer Softwa
  - SCM is a QA activity that is applie throughout the Software process
  - slw Change must be managed & Controlled in order to improve quality an reduce error.



* Elements of Configuration management System



* Software Risk and types:
  - Refers to a Condition or event that may adversely affect a slw project.
  - Risk is potential problem - it might happen or it might not.
  - It impact project Cost, schedule, performance or quality.



* RMMM Plan :
  - It is a document in which all the risk analysis activities are described
  - RMMM stands for Risk mitigation, monitoring and management.

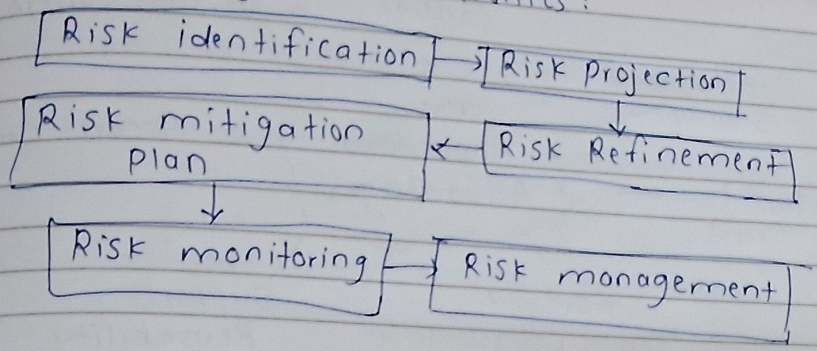
Mitigation:
- preventing risks to occur
- find Probable risk
- elimate risk causes
- Develop policy
- maintain docs in timely manner
- conduct timely reviews

RMMM

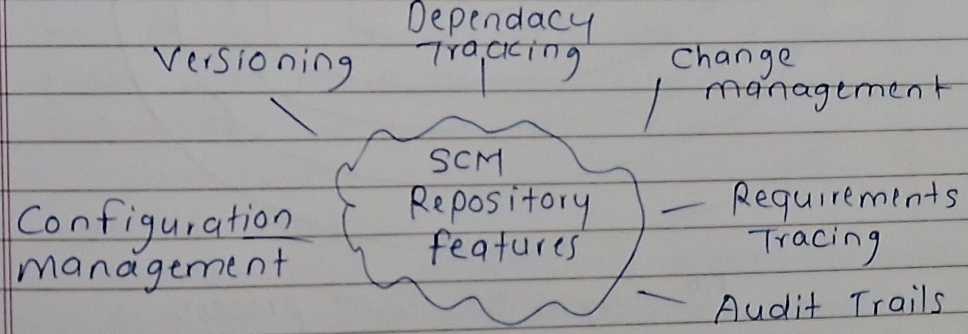
Monitorig:
- monitored by PM
- Approach of team
- co-op in team
- types of problems
- Availability of jobs in and out of organiz
- Check risks really occurs or
- ensure mitigation steps are applied Correctly
- gather info to analyze r

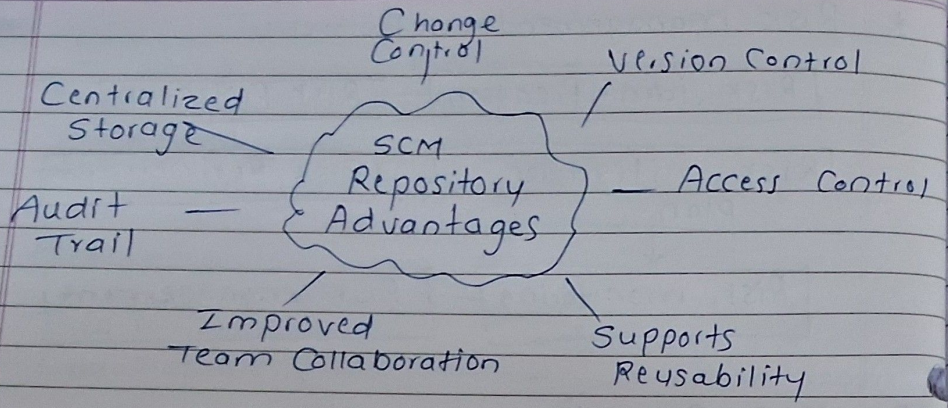
management
- PM performs this task when risk becomes reality
- If mitigation is applied successfully / effectively then it becomes very much easy to manage the risk

* Risk management activities :



* SCM Repository :
  - SCIs are maintained in a project repo.
  - It is basically a database that acts as Center for both accumulation and Storage for slw information
  - Handled using modern DBMS functions.
  - It maintains properties like: data integrity, sharing and integration.
  - must maintain uniform structure & format





* SCM process:

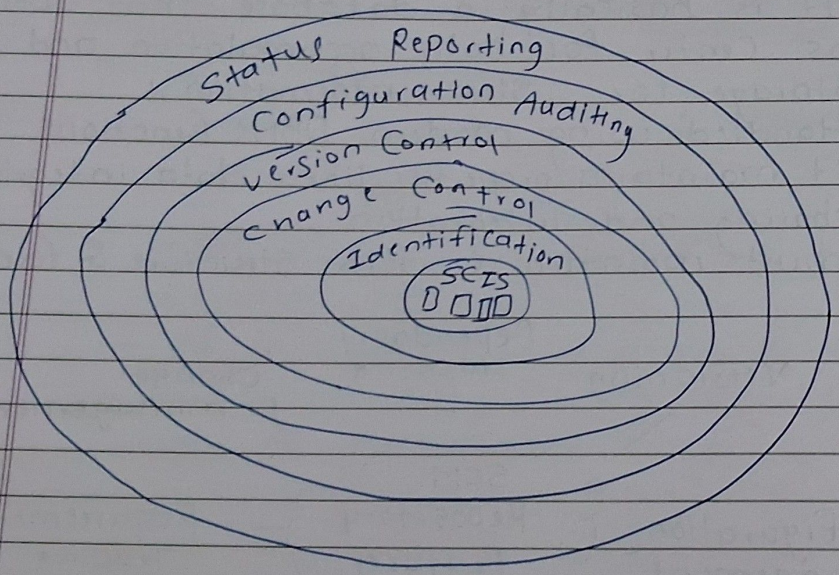


fig. Layers of scm process.

Benefits:
- prevents unauthorized changes
- facilitates team collaboration
- Supports Traceability and rollback
- Improve product quality and reliability.

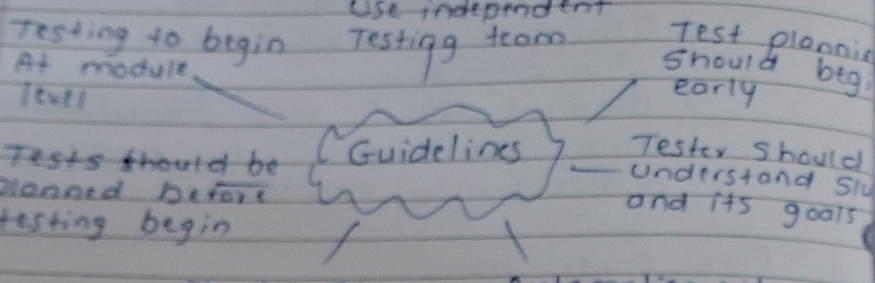
* Risk Identification:
  - first step in Risk management process
  - goal is to create list of potential problems
  - Based on two approaches
    - Generic risk identification: threat to slw project
    - product-specific risk identification: threat identification by people, technology and working environment.

* Risk projection:
  - Also called as Risk estimation
  - Rates Risk in two ways:
    - probablity of risk occurance
    - Consequence of risk occured
  - Risk table for managers to use for risk projection (Table fields: Risks, category, probablity, impact, RMMM)

* Risk Refinement:
  - process of breaking down general risk into more specific and manageable sub-risks
  - Used when risk is too complex to address directly.
  - make risk assesment more accurate
  - Allows creation of precise mitigation plans.

Unit - 6

* Guidelines for successful testing:



* Integration testing:
  - Individual modules are combined and tested as a group
  - goal is to identify interface defects betⁿ m
  - ensures diff. part of the system work together correctly

* Top-down Integration Testing:
  - Modules integrated from top (main modul to bottom (Sub-module)
  - Main Control is tested first
  - Stubs are used for sub-modules not yet developed
  - Early verification of top-level logic & Contro
  - major design flaws detected early
  - Requires many Stubs
  - Low-level modules tested late

* Bottom-up Integration Testing
  - Integration starts from lowest-level module and moves upwards
  - Drivers are used to simulate higher module
  - test independant Sub modules first
  - Lower-level modules tested early
  - NO Stubs are needed
  - High-level logic is tested late
  - Requires test drivers for each module se

* Verification Vs. Validation:

Verification	Validation
◦ Set of Activities that ensure SW Correctly implements specific func.	◦ Set of Activities that ensure SW is build to Customer requirement
◦ Starts After Complete Specification	◦ begins as soon as project starts.
◦ It is prevention of errors	◦ It is detection of errors.
◦ Conducted using reviews, walkthroughs, inspection and audits	◦ Conducted using System testing, UI testing, Stress testing.
◦ Also called white box testing or static testing	◦ Also termed as black box testing or dynamic testing.
◦ finds about 50-60% of defects	◦ finds about 20-30% of defects

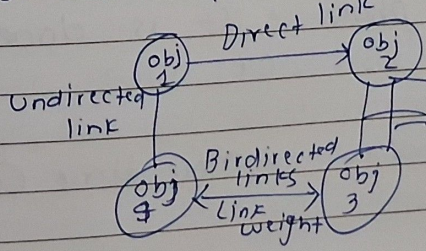
* Black-Box testing Vs White Box testing

Black Box testing	White Box testing
° Based on inputs & outputs,	° Based on code and structure
° Internal Code is not visible	° Internal Code is visible
° verifies functionality against requirments	° verifies internal logic, loops, paths.
° No knowledge of internal code required	° Requires full knowledge of source code
° performed by Testers or QA team	° performed by developers.
° Functional testing	° structural testing
° Tools used: Selenium, QTP, etc.	° Tools used: JUnit, NUnit, etc.

* Conventional testing Vs. object oriented testing

Conventional testing	object-oriented testing
° Focuses on functions, procedures & data flow	° Focuses on classes, objects and their interactions.
° Testing levels: Unit → integration → System → Acceptance	° Testing levels: Class → Cluster → System
° Testing units: individual procedure & modules	° Testing units: Classes and methods
° Inheritance & polymor. Not applicable	° must test inherited methods, overridden behavior.
° Relatively simpler, procedural logic	° more complex due to inter-object interactions and state behavior.

* Graph based functional testing techniques:
- technique in which a graph of objects present in System is Created

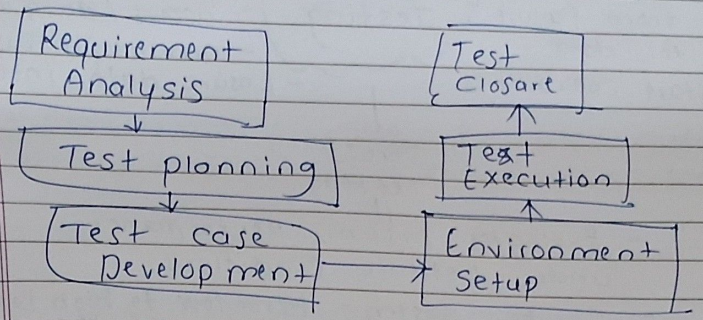


* User Acceptance testing:
  - Testing to ensure SIW works Correctly in user's environment
  - Final phase of SIW testing process
  - performed by end users or client
  - Two types: Alpha & Beta testing.

* Alpha testing Vs. Beta testing

Alpha testing	Beta testing
° Performed at developers site	° Performed at end user's site
° Performed in Controlled environment	° Performed in uncontrolled environment
° Developer is present	° Developer is not present
° Less probablity of finding errors	° High probablity of finding errors
° Not Considered as live application	° It is Considered as live application.
° done during implementation phase of Software	° It is done pre-release of SIW
° Less time Consuming	° More time Consuming

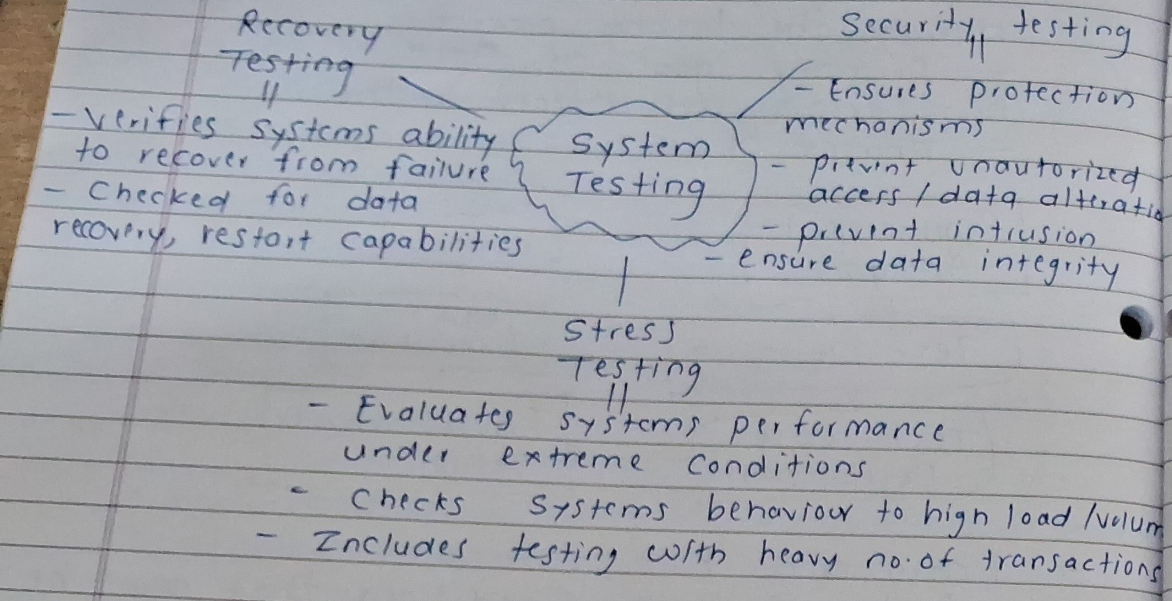
* Software testing Lifecycle:
  - Sequence of activities Conducted during the testing process
  - primary goal of STLC is to identify bugs, errors, and defects in the SIW
  - phases of STLC:



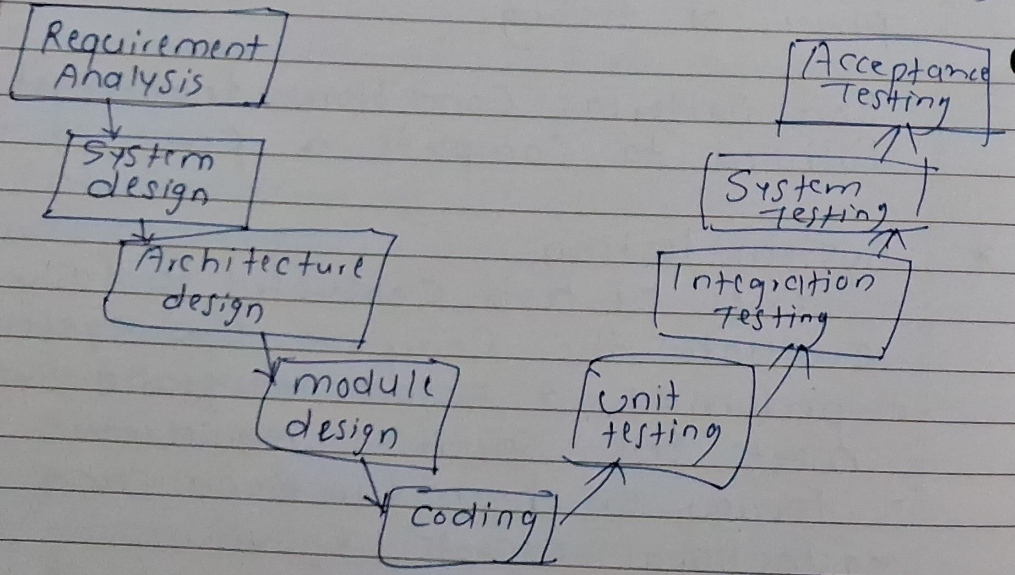
* Entry and Exit Criteria:
  ° Entry Criteria: Conditions that must be met before to start a particular phase of testing
  ° Exit Criteria: Conditions that must be fulfilled to Complete a phase of testing.

* System testing:
  - Series of tests Conducted to fully evaluate the Complete SIW system.
  - performed to ensure integrated System meets the specified requirements.
  - focuses on both functional and non-functional aspects of system.
  - validate Overall performance of System
  - Checks System's ability to work under various condition

* types of System testing :



* verification & validation model (V-model)



* Manual testing Vs. Automated testing

Manual testing	Automation testing
• Human executes Test cases without using tools or scripts	• Uses tools and scripts to automatically execute test cases.
• Executed step-by-step by the tester	• executed automatically based on predefined scripts
• Slow Speed	• Fast speed
• Lower initial cost but expensive for large projects	• Higher initial cost but cost effective over time
• Can be prone to human errors.	• more accurate
• more flexible	• Less flexible.
• Can handle limited tests by testers capacity & time	• Can handle large no. test cases across platf

All the Best!!

— KARAN SALUNKHE