Software Engineering:-

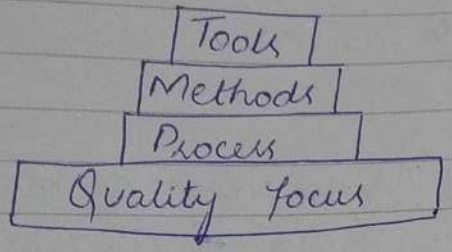
Application of systematic, disciplined, quantificable approach to deve operation & maintainance of software.

OR Study of application of methedologies to develop quality software that fulfil customer needs.

Common Issues:-

- Software doesnot fulfil customer needs.
- Hard to improve & extend.
- Bad documentation.
- Bad quality
- More cost and time than expected.

SE as a Layered Technology:-



Generic View:-

1- Definition phase focus on what.
2- Development phase focus on how.
3- Support phase focus on changes associated with error correction.

1- System or info engineering.

- Software project plan
- Req analysis

2- Software Design
- Code generation
- Software Testing

3- Correction

- Adaptation
- Enhancement
- Prevention

(Process Models)

Prespective models.

- Linear (Step by step)
- Evolutionary (Cyclic repititions allowed) Spiral
- Incremental. (Divide in chunks)

Protolype

SDLC (Software Development Life Cycle)

1- Plan & Required Analysis
2- Define : Implementation.
3- Design
4- Build / Coding
5- Test
6- Deploy.

Process : Different steps
Process Model : Whole Project

I/P → Process → O/P
only execution

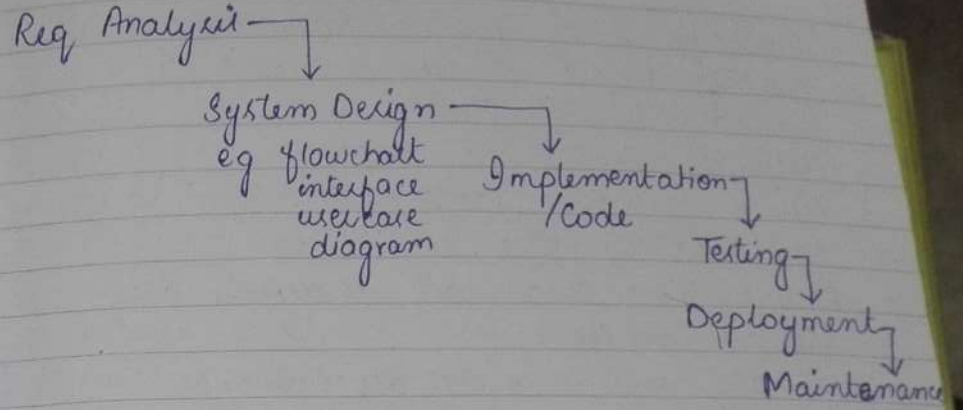
Waterfall Model :-

→ Linear sequential life cycle model.
→ simple to understant & use
→ Each phase must be completed before moving to next phase.
→ No Overlapping.
→

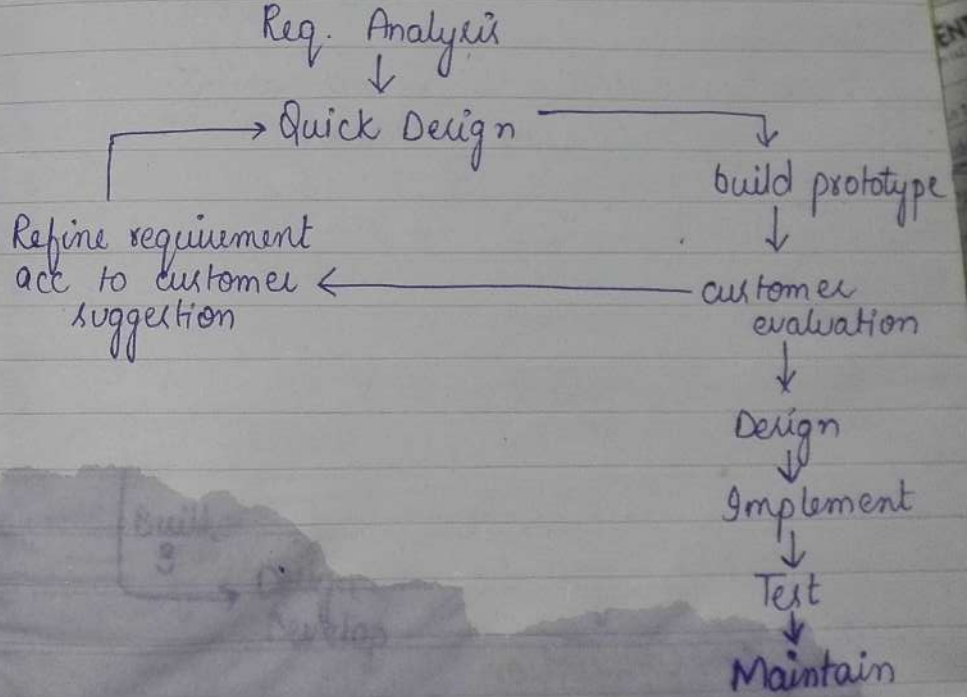
Application :-

→ Requirements are well documented clear & fixed.
→ Product definition is stable
→ Tech is understood & not dynamic.
→ Ample resources with required expertise are available.
→ The project is short.

Software Req Specification : SRS
Req - Filter - go (Output of req Analysis)



Prototype Model :-

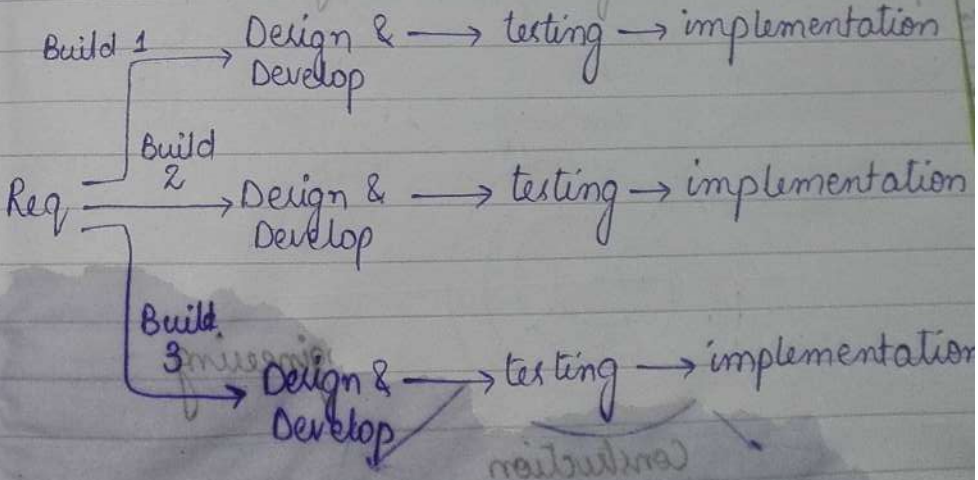


→ Prototype is toy implementation of system.
→ Working model of software with limited functionalities
→ Helps to reduce risks.
→ Quick feedback, scope may extend.
→ can lead to false expectations.
→ Reduce time and cost as defects are deleted earlier
→ Primary goal is rapid development.
→ Prototype can be
- horizontal: Display user interface, broadens view without internal functions.
- vertical: technical, to get details of exact functions.

Basic idea behind this is to develop a system through repeated cycles & in smaller portion of time.

Iterative Model:-

→ Starts will small set of requirement & iteratively enhance evolving version until complete system.
→ At each iteration, modifications are made.
→ Results obtained early & periodically.
→ Parallel development can be planned.
→ Less costly to change scope/requirement.
→ Testing smaller part is easy.
→ Suited for large & mission critical projects.
→ More resources required.
→ End of project may not be known.
→ Each phase of iteration is rigid with no overlaps

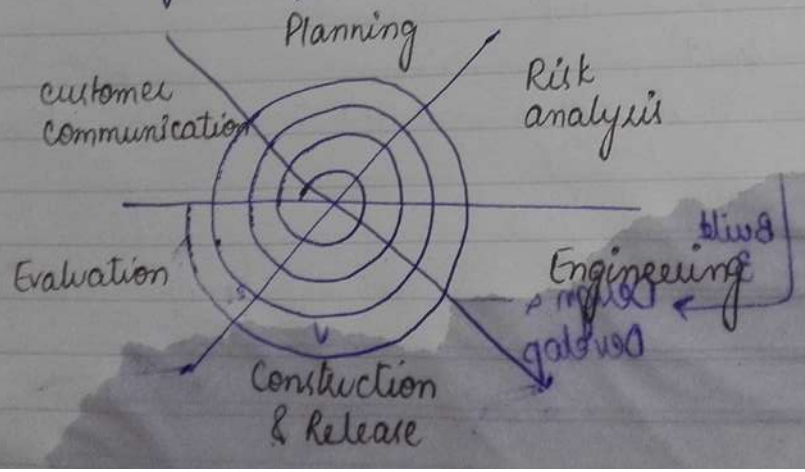


Spiral Model:-

- Risk driven process Model.
- Combination of waterfall & iterative.
- Introduce by Barry Boehm in 1986.
- Team adds functionality for additional requirement in every increasing spiral.

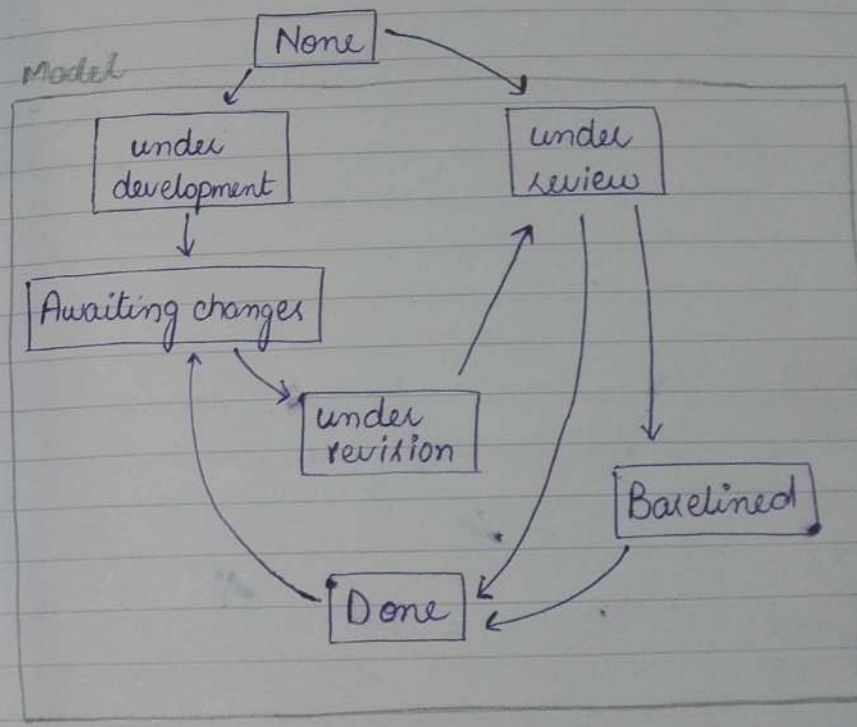
Use:-

- When project is large.
- Requires frequent releases.
- When risk & cost evaluation is important.
- Unclear & complex requirements.
- Changes requirement at any time.



Concurrent Development Model:-

- Concurrent Engineering.



AGILE DEVELOPMENT

Manifesto for Agile Development:-

- Individuals & Interaction over processes & tools.
- Working software over comprehensive documentation.
- Customer Collaboration over contract negotiation.
- Responding to change over following a plan.

Agile Methodology:-

- Group of software development methodologies based on iterative developm
- Where requirements & solutions evolve through collaboration of self organizing cross functional teams.
- Any development process that is aligned with concepts of agile manifesto.

Principles of Agile Methodology:-

1. Customer satisfaction by early & continuous delivery of valuable software.
2. Welcome changing requirements even in late developments.
3. Working software is delivered frequently (weeks rather than months).
4. Close daily cooperation between business people & developers.
5. Projects are built around motivated individuals
6. Face to face conversation is best form of communication. (co-location)
7. Working software is principle measure of progress.

8. Sustainable development able to maintain constant pace.

9. Continuous attention to technical excellence & good design.

10. Simplicity, art of maximizing ammount of work not done is essential.

11. Best architecture, requirements & design emerge from self-organizing team.

12. Regularly team reflects on how to become more effective & adjust accordingly.

EXTREME PROGRAMMING:-

- Improve quality & responsiveness to change.
- frequent release in short development cycles to improve productivity & introduce checkpoints.

4 activities:-

- Code
- Test
- Listen
- Design

12 practices, group in 4 areas:-

1- Fine scale feedback.
   - pair programming.
   - Planning game
   - Test driven depelopment.
   - whole team

2- Continuous process.
   - continuous integration

Extreme Programming:- (XP)

Extreme programming is a new methodology designed to help small development teams deliver value despite constantly changing requirements.

1- Fine Scale Feedback.

1- Pair Programming:- XP programmers work in pairs. All code is developed by two programmers who work together at a single machine. High quality code at same or less cost is produced.

2- Planning Game:- It is main planning process in XP. It includes release planning & iteration planning.

3- Test Driven Development:- Each building block is tested prior to release. This allows for clean application in the long term by flushing out problems before they get lost in large application.

4- Whole Team:- The whole team is available to make sure the customer is integral part of development effort.

Release Planning:- Requirements are gathered & commitments are done regarding project.

Iteration Planning:- The developers were involved to plan the activities & tasks for iteration.

→ **Continuous Process:-**

5. Continuous integration: Intagrating the chunks & keeping application up to date with most recent changes.

6- Design improvement: Constantly improves design of the software.

7- Small releases:- With XP, you deliver & develop application series of small frequently updated versions.

→ **Shared Understanding:-**

8- Coding Standards: All the code is written in same way so it is understandable to all.

9- Collective Code Ownership:- No person owns or is responsible for indivival code segments.

10- Simple Design:- The best design is the easiest one that works.

11- System Metaphor: XP systems development means sticking to a set of standards for items i.e. vailable names etc.

→ **Programmer Welfare:-**

12- Sustainable Pace:- For XP practices to be effective, the developers must be on top of their game & able to maintain sustainable pace without exhausting themselves.

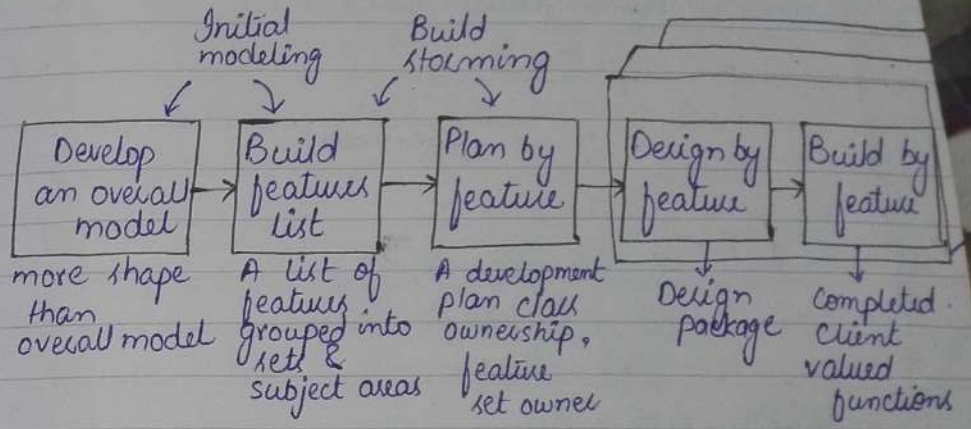
- design improvement
- Small release

3- Shared Understanding.
- coding standard.
- collective code ownership.
- simple design
- system metaphor.

4- Programmer welfare
- sustainable pace

Feature Driven Development:-

- Client centric, Architecture centric, pragmatic software process.



Primary Roles:- Project Manager, chief architect, Developer manager, chief programmer, class owner, domain expert.

Lean Software Development:-

Software developers face
- muda → waste created by employees by extra effort that is non-valuable for customer.
- mura → waste created by unevenness (resulting in lack of interest of employee)
- muri → waste created by overburden
depend on human nature (controllable)

3

Lean principles:-

1- Eliminate waste (i.e. muda)
2- Amplify learning
3- Decide as late as possible
4- Deliver as fast as possible
5- Empower the team
6- Build integrity

Perceived integrity: customer need to have overall exp of system

Conceptual integrities: components work well together

7- See the whole
(Think big, act small, fail fast, learn rapidly)

Lean Principles:-

01- Eliminate Waste:-
Unnecessary code, features and functionalities are eliminated.

02- Amplify Learning:-
Create solutions to unique customer problems means new work should be done every time.

03- Decide as Late as Possible:-
If project requirements are not clear and understood, then decisions should be taken as late as possible i.e. once every thing becomes understand-able.

04- Deliver as fast as possible:-
The project once completed should be deliver as fast as possible because rapid delivery means less time for customer to change their time.

Advantages:-

1- Elimination of waste leads to overall efficiency of development process. Reducing project time & cost.

2- Delivering the product early is a definite advantage.

3- Empowerment of development team helps in developing the decision making ability of the team, which results in creating a more motivated team.

Disadvantages:-

1- Project is highly dependant on cohesiveness of team & indivival commitments of team members.

2- Succes in the project depends on how disciplined the team members are and how exeptional are their technical skills.

3- If you don't have a person with right business analyst skills, Then you quickly find this become a cause of scope creep.

4- In lean you allow SRS to evolve, flexibility is great but too much of it will lead to a development which looses sight of its original objectives.

5- Empower the team:-
Let the working team design their own working procedures.

6- Build Integrity:-

Integrations should be made of 2 types.

i)- Perceived Integrity:-
short iteration should be used & feedback should be acquired.

ii- Conceptual Integrity:-
Understand the problem & solve it at the same time.

07- See the Whole:-
(Think big, act small, fail fast & learn quickly)

You must have a complete picture of your project.

SCRUM

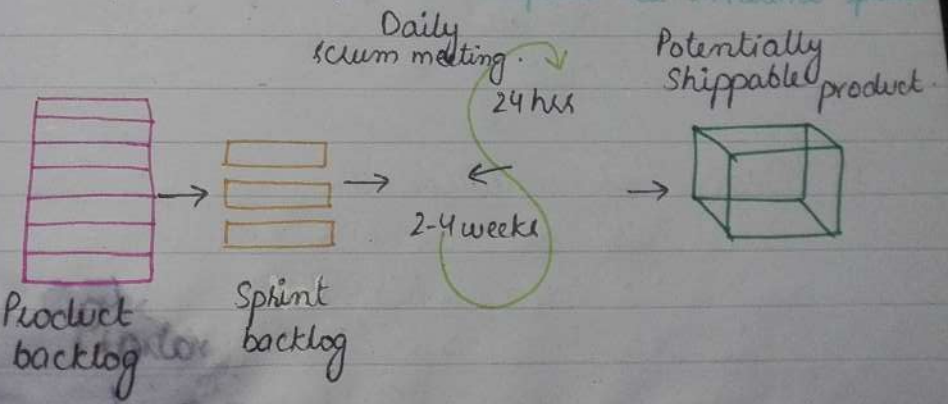
Follows all agile principles (12 Points) & its manifesto

- Aggressive deadlines, complex requirement, degree of uniqueness. to predict a model of a software that never been used before
- Project moves foward via series of iteration called sprints. sprint → duration of 2-4 week. You have to complete a specific task cycle in a sprint.

- Scrum team — 5-9 persons having all qualities of agile team.
- Product owner — key stake holder (any person which have an impact (+ve/-ve) on project.
- Scum master — team lead (handles, lead, manages & divide work)

we deal with direct stake holder (client)

Sprint backlog
- Product backlog — all requirements, features of project
- Sprint planning meeting — how to manage work of project backlog.
- Daily scrum — meetings on daily basis (close daily cooperation)
- Sprint Review meeting — at end of each sprint.
- Sprint Retrospective — also at end of sprint to see all lackness in project & how to improve that in further sprints



Scrum

Scrum is most important and most used technique of agile methodology as it follows completely agile manifesto and its principles.

Scrum is used in following cases.

1) Aggressive Deadlines:-
When you have to deliver the project on the exact time, scrum will be the best choice.

2) Complex Requirement:-
When the project is very large and all the requirements are not clearly known.

3) Degree of uniqueness:-
If you are working on a new innovative idea which does not exist already and you have to build a new strategy and ideology, then scrum will be best choice.

The project moves forward in a series of iterations called sprints (2-4 weeks).

Scrum Team:-

Scrum teams consists of 5-9 persons having all qualities of agile team. It also consists of.

Product Owner:-
Here we are saying that product owner is a person who is direct key stake holder of the project (i.e client) and creates a prioritized wish list called a product backlog.

Scrum Master:-
Scrum Master leads the team & keeps the team focused on its goals.

Scrum Events:-

Product backlog:-
Product backlog is the prioritized wish list created by the owner.

Sprint backlog:-
The team pulls a small chunk from the top of that wish list (product backlog), a sprint backlog, and decides how to implement those pieces.

Sprint planning meeting:-
Scrum team decides how the work will be done in each iteration and a proper plan.

Daily Scrum:-
The team has a certain ammount of time, a sprint (2-4 weeks). Each day the project's progress is assessed.

Sprint Review Meeting:-
At the end of the sprint, the work should be ready to hand to a customer, put on a store shelf, or show to a stakeholder but before that the sprint is reviewed in a meeting.

Sprint Retrospective:-
The sprint ends with a sprint review and retrospective.
As the next sprint begins, the team chooses another chunk of sprint backlog and begins working again.

Advantages:-

- Due to short sprints & constant feature, it becomes easier to cope with changes.
- Scrum is iterative in nature. It requires continuous feedback from user.
- Agile scrum helps the company in saving the money.
- Daily meetings make it possible to measure individual productivity.
- Improve the productivity of each of the team members.

Why Use Scum?

1- Scum offers freedom of implementation within its set of rules.

2- Scum is easy to learn and use.

3- Through inspection & adaption Scum lets embrace the change & it has less impact for the customer & the product development.

4- Scum reduces the risk by building the product by increments.

5- Scum optimizes the team's efficiency.

6- Scum lets the customer to use the product earlier.

7- Scum is continuous improvement.

- It is easier to deliver a quality product in a scheduled time.

Disadvantages:-

- If the team members are not committed, the project will either never complete or fail.
- If a task is not well defined, it can be spread over several ~~units~~ sprints.
- It is not good for long ongoing projects.
- Scrum often leads to scope creep due to lack of a definite end-date.
- The framework can be successful only with experienced team members.
- Quality is hard to implement, until the team goes through aggressive testing procedure.

SOFTWARE REQUIREMENT ENGINEERING

Requirements:-

The software requirements are description of features & functionalities of target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirement Engineering:-

The process to gather the software requirements from client, analyze & document them is known as software requirement engineering.

Tasks:-

It is a four step process, which includes.

i- Feasibility study OR Inception.
ii- Requirement Gathering OR { Elicitation, Elaboration, Negotiation

iii- SRS OR Specification.
(Software Requirement Specification)

iv- Software Requirement Validation OR { Validation Requirement

1- Feasibility Study:-

- A feasibility study is carried out to select the best system that meets performance requirements.
- The main aim of feasibility study activity is to determine whether it would be financially & technically feasible to develop the product.
- Feasibity study activity involves the analysis of the problem & collection of all relevant information related to the product. such as different data items which would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system as well as various constraints on the behaviour of the system.

2- Requirement Gathering:-

- If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user.
- Analysts & engineers communicate with the client & end-users to know their ideas on what the software should provide & what features they want the software to include.

3- Software Requirement Specification:-

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

The requirements recieved from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language

Features of SRS:-

SRS should come up with following features.

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo Code.
- Formats of forms & GUI prints.
- Conditional & mathematical notations for DFDs etc.

so that they can be comprehended and useful by the software development team.

4- Software Requirement Validation :-

- After requirement specification are developed, the requirements mentioned in this document are validated.

User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly, this results in huge increase in cost if not nipped in bud.

Requirements can be checked against following conditions.

- if they can be practically implemented.
- if they are valid & as per functionality & domain of software.
- If there are any ambiguities.
- If they are complete.
- If they can be demonstrated.

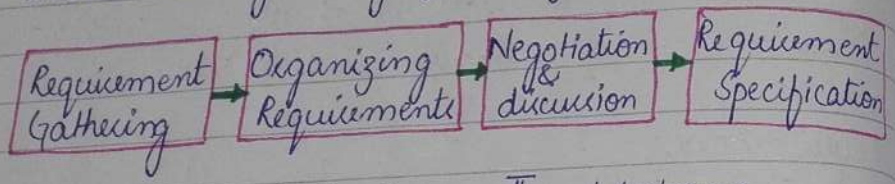
Elicitation :-

Requirement gathering
Organizing Negotiation
Requirement Specification.

- Interview:
  - structual (closed) ← fixed number of questions
  - Non-structual (open) ← open discussion or random questions.
  - oral, written, one-to-one, group.
- Survey ← preffered when we dont have specific customer
- Questionaries ← targets stakeholders, but do not conduct directly, instead questions are send.
- Task analysis
- Domain analysis
- Brain storming
- Prototyping
- Observation.

ELICITATION:-

Requirement elicitation process can be understood by using following diagram



Requirement Gathering ⇒ The developers discuss with the client & end users and know their expectations from software.

Organizing Requirements ⇒ The developers prioritize & arrange the requirements in order of importance, urgency & importance.

Negotiation & Discussion ⇒ To remove the ambiguity & conflicts, they are discussed for clarity & correctness.

Documentation :- All formal & informal, functional & non-functional requirements are documented & made available for next phase processing.

Methods / Techniques :-

1- Interviews :-

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews as :-

1- Structured (closed) interviews in which fixed question are made in advance about every single information to gather.

2- Non-Structured (open) interviews which are more flexible & less biased.

3- Interviews can either be oral or written, they can be one-to-one or group interviews.

2- Surveys :-

Organization may conduct surveys among various stake-holders by querying about their expectation & requirement from the up-coming system.

3- Questionnaries:-

A document with pre-defined set of questions & respective options is handed over to all stake-holders to answer, which are collected & compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

4- Task Analysis:-

Team of engineers and developers may analyze the operation for which the new system is required.

5- Domain Analysis:-

Every software falls into some domain category. The expert people in the domain can be a great help to analyse general & specific requirements.

6- Prototyping:-

Prototyping is building user interface without adding detail functionality, for user to interpret the features of intended software product. It helps giving better idea of requirements.

The client's feedback for prototype serves as an input for requirement gathering.

7- Observation:-

Team of experts visits the client's organization or workplace. They observe the actual working of the existing installed systems.

They observe the workflow at client's end and how execution problems are dealt.

The team itself draws some conclusions which help to form requirements expected from software.

TYPES OF SOFTWARE REQUIREMENTS:-

Explicit Requirements:-

All those requirements which the customer tells itself ~~are~~ to the software developing team are categorized into explicit requirements.

Implicit Requirements:-

These are the requirements which are not being told by the customers, but the developing team show know these requirements.
for example, if a software is to be made for both android & widows, the developing team must know which technology would be used for that.

User-Interface Requirements:-

- User Interface is an important part of any software. User acceptance majorly depends on how the user can use the system. A
- A well performing software system must also be equipped with attractive, clear, consistent user interface
- Otherwise, functionalities of software system cannot be used in a convenient way.
- User interface requirements include content presentation, easy navigation, responsiveness, default settings etc.

SYSTEM REQUIREMENTS:-

System Requirements are further classified into 3 types.

i- functional Requirement.
ii- Non-functional Requirement
iii- Domain Requirement

1- Functional Requirements:-

- Requirements which are related to functional aspect of software fall into this category.

- They define functions and functionality within and from the software system.

Examples:-

i- Search option given to user to search from various invoices.

ii- User should be able to mail any report to management. etc.

2. Non-functional Requirements:-

- Requirements, which are not related to functional aspect of software, fall to this category.
- They are implicit or expected characteristics of software, which users make assumptions of.
- Examples include security, logging, storage, performance etc.

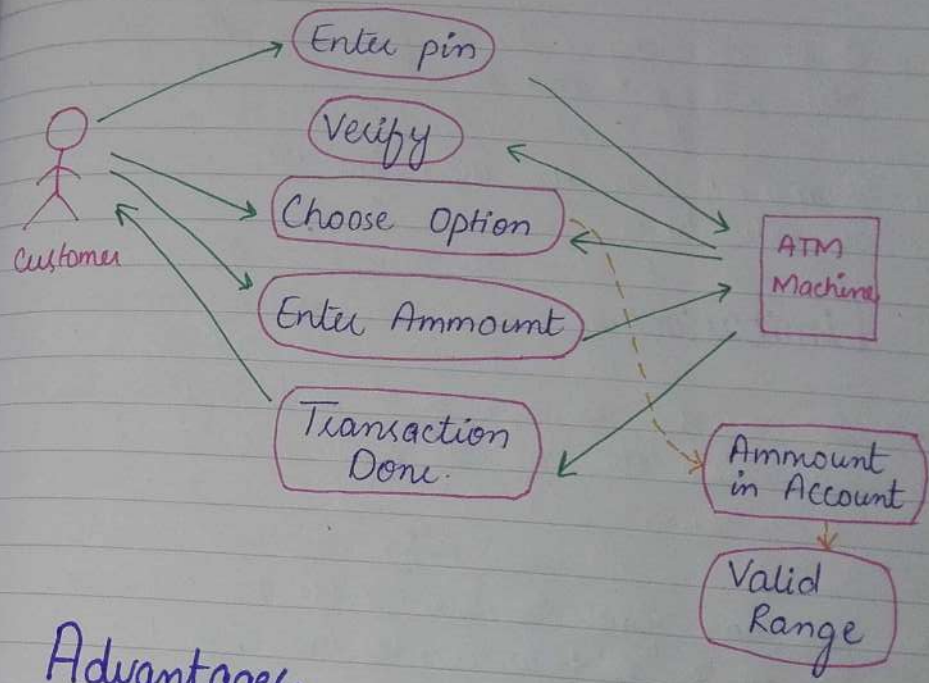
Functional Requirements	Non-functional Requirements
• Describe what software system should do.	Describe how sof

Use Case Diagram & Analysis:-

List of actions or events typically defining interactions between a role (actor) & system to achieve goal.

Elements of Usecase:-

i- Name
ii- Brief Desciption.
iii- Actor
iv- Pre-Conditions
v- Basic flow.
vi- Alternate flow
vii- Exeption flow.
viii- Post Condition.



Advantages:-

- List of goal names provide summary of what the system will offer.
- Also provide project planning skeleton.
- Helps to avoid documenting the same information in more than 1 detailed use case.

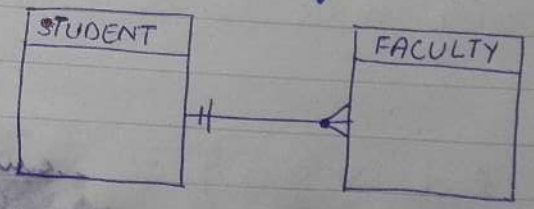
- Helps identify common functionality for later coding decisions.
- Scenarios can be used for test script or validation.

Limitations:-

- Use cases are not well suited to capturing non-interaction based requirements of a system.
- For some projects, use cases are complex to write and to understand, for both end users & developers.
- As there are no fully standard definitions of used cases, each project must form its own interpetation.

- Provides first technical representation of system.
- 3 primary objects.
  - to describe what customer requires
  - to establish basis for creation of software design.
  - to define set of requirement that can be validate once software is built.
- Shows 3 aspects of software
  1- Data Modeling (Description of data & its flow)
  2- Functional Modeling (functional requirements)
  3- Behavioral Modeling.

Analysis Rules of Thumb:-



overall shows the picture.

Relationship between student & faculty.

1- The model should focus requirements that are visible within problem or business domain (level of abstraction should be high)

2- Each element of analysis model should add an overall understanding of software requirement and provides inside into information domain function and behaviour of system.

3- Model should delay consideration of infrastructure & other non-functional models until design phase.

4- The model should minimize coupling throughout the system.

5- Should provide value to all stake holders.

6- Should be kept as simple as possible.

Identification specification and analysis of common, reuseable capabilities within a specific application : domain

Source of domain Knowledge

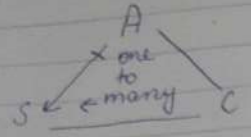
→ Technical Literature
→ Existing application
→ Customer survey & expert advice
→ Current / future requirement

Outcome of Domain Analysis

→ Class taxonomies
→ Reuse Standards
→ Functional & Behavioral Models
→ Domain Knowledge.

Data Modeling.

→ Data objects (entities)
→ Data Attributes
→ Relationship
→ Cardinality / Modality



DATA MODELING:-

Data modeling is the analysis of data objects and their relationship to other data objects.

Data modeling involves a progression from conceptual model to logical model to physical schema.

1- Data object:- object can be anything, person, or non-living thing.

2- Data attribute:- They are the properties of object. For e.g. name, gender of person or object.

3- Relationship:- Relationship is between two entities - object relationship are bidirectional.

4- Cardinality:- No. of occurances or relationship. It is used in ERD.

5- Modality:- Existance of data objects, whether it is mandatory or not. For eg to study, whether a student or book both are mandatory or just the student is mandatory. It will be decided using modality

1 = Mandatory, 0 = None.

ANALYSIS MODELING APPROACHES:-

1 Structured Analysis:-

- Consider data and the processes that transform the data as seperate entities
- Data is modeled in terms of only attributes and relationships (but no operations)
- Processes are modeled to show the
  1- input data.
  2- the transformation that occurs on that data.
  3- the resulting output data

2- Object Oriented Analysis:-

- Focuses on the definition of classes and the manner in which they collaborate with one another to fulfil customer requirements.

Elements of Analysis Model:-

Object-Oriented Approach.

Scenario Based Elements
Use-cases text
Use-cases diagrams
Activity diagrams
Swim lane diagrams

Structured Analysis Approach.

Flow-oriented Elements
Data flow diagrams
Control flow diagrams
Processing narratives

Class-based Elements
Class diagrams
Analysis packages
CRC Models
Collaboration diagrams

Behavioral Elements
State diagrams
Sequence diagrams

Flow Oriented:-
(Structured Analysis)

- Provides indication of how data objects are transformed by set of processing function.

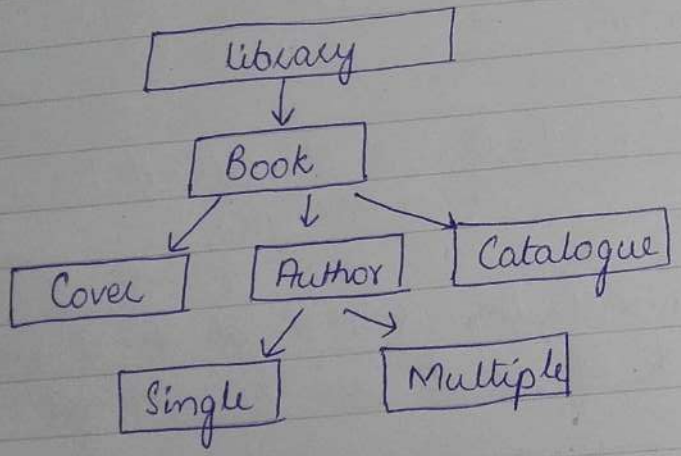
Object Oriented:-

- Scenario based: Representation system from user point of view. e.g. usecase diagram.
- Class based: Define objects, attributes and relationships.
- Behavioral: Depicts state of classes and impact of events on these classes.

* Analysis outputs are verified.
* Design outputs are validated.

DSD:- (Data Structure diagrams)

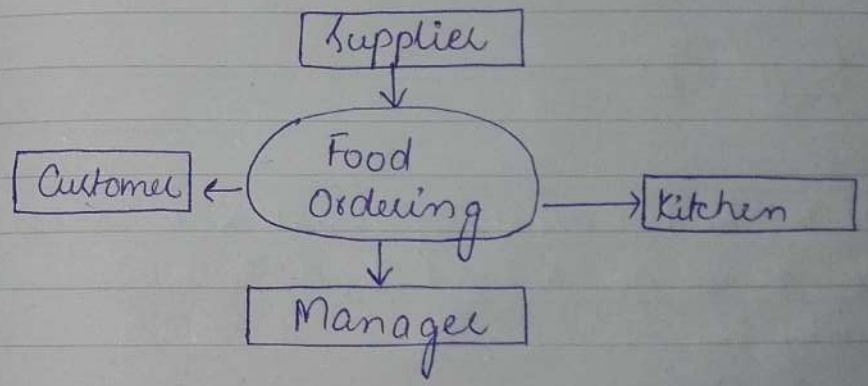
Data Structure diagram is a diagram of the conceptual data model which documents the entities and their relationships, as well as the constraints that connect to them. Data structure diagrams are most useful for documenting complex data entities.



DFD:-

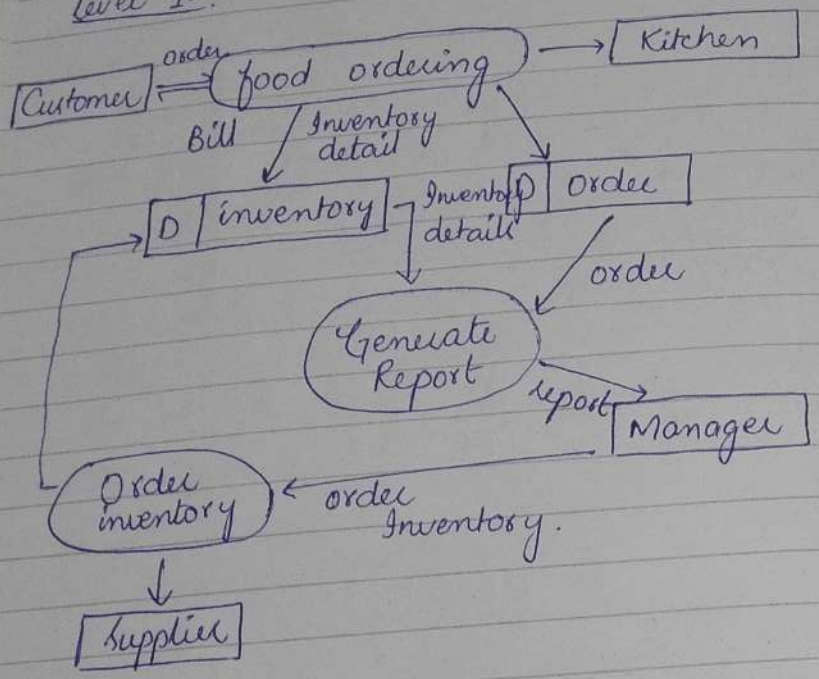
A data flow diagram (DFD) is the graphical representation of "flow" of data through an information system, modelling its process aspects.

A DFD is often used as a preliminary step to create an overview of system without going into great detail, which can later be elaborated.

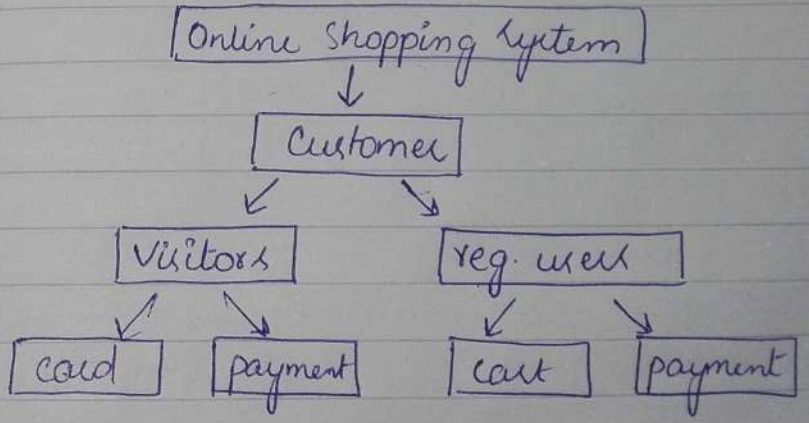


level-0

Level 1.

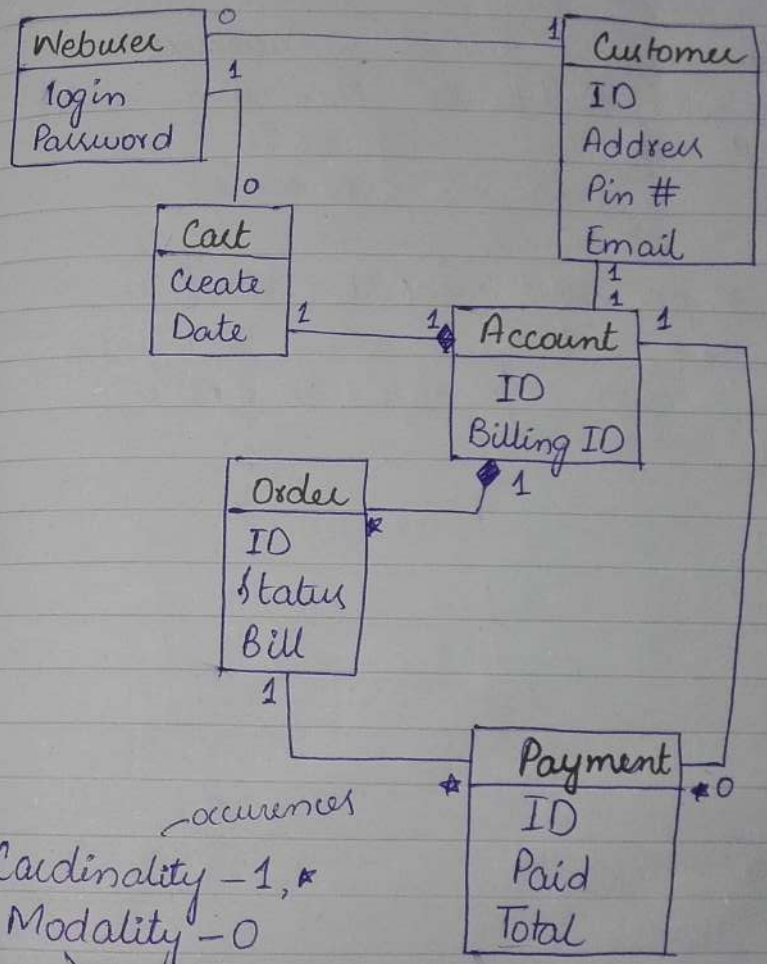


* As the level increases, elaboration increases.
* We do not go beyond level-2 while analyzing.



◆ → show composition

CLASS DIAGRAMS:-



occurences

Cardinality – 1, *

Modality – 0

optional

* – for many

0 – for optional.

CRC → Class Responsibility Collaborator.

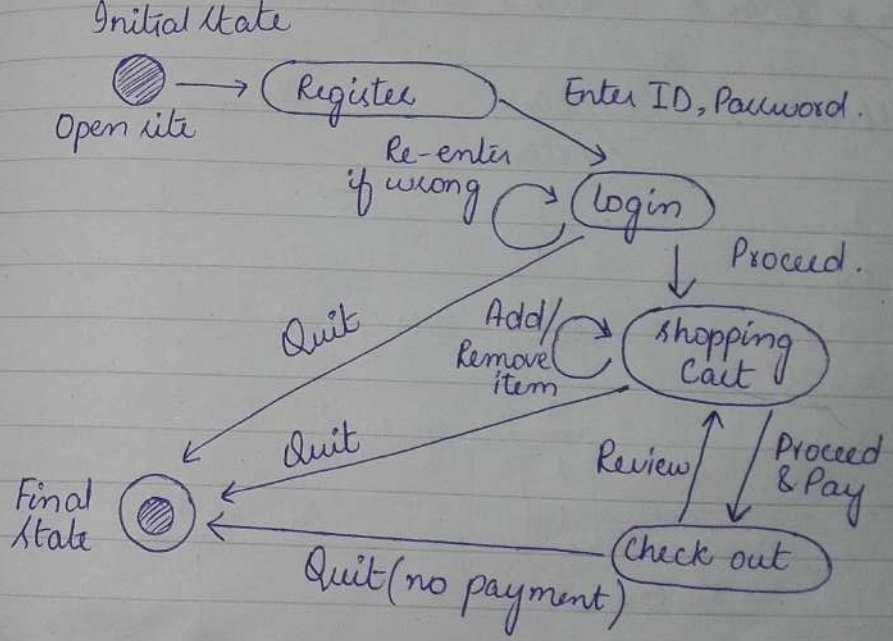
Class Name	
Responsibilities	Collaboration

Customer.	
Place Order Give ID Give Address	Order

Class responsibility collaboration (CRC) cards are a brainstorming tool used in the design of object oriented software.

STATE DIAGRAMS:-

A state diagram is used to describe the behaviour of systems.
State diagram require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction.



SOFTWARE DESIGN:-

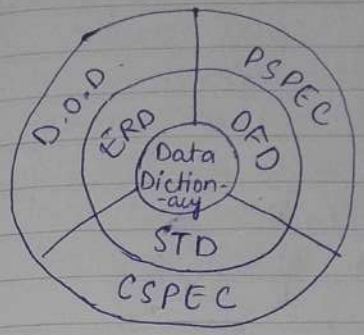
An iterative process of transforming requirements to a blue-print for constructing software.

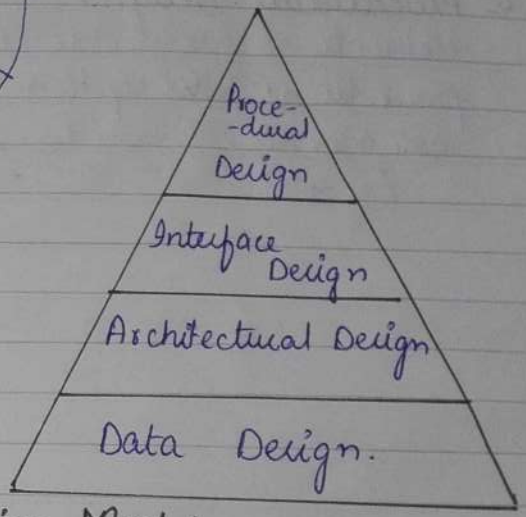
Goals:-

1- The design must implement all of explicit requirements contained in analysis model and it must accomodate all of implicit requirements desired by customers.

2- The design must be reliable, understandable guide for those who generate code and for those who test and subsequently support software.

3- The design should address the data, functional and behavioral domains for an implementation prespective.





Components of Design Model:-

1- Data Design:- Transforms information domain model to data structures required to implement software.

2- Architectual Design:- Defines relationship among major structural elements of software.

3- Interface Design:- Describe how software communicates within itself and humans that interact with it.

4- Procedural Design:- Transforms structural elements of software Architecture to procedural description of software components. (Algorithm).

Design Principles:-

1- The design process should not suffer from tunnel vision.

2- The design should be traceable to analysis model.

3- The design should not re-invent the wheel.

4- The design should minimize the intellectual distance between software and problem as it exist in real world.

5- The design should exhibit uniformity and integration.

6- The design should be structured to accomodate change.

7- The design should be structured to degrade gently even when abnormal (false) data, events, or operating conditions are encountered.

8- Design is not coding, coding is not design.

9- The design should be assessed for quality as it is being created, not after the fact.

10- The design should be reviewed to minimize conceptual (semantic) errors
↓
logical.

Process Models:-

To solve actual problem in an industry. SE must incorporate a development strategy that encompasses the process, methods, tools of layered technology and phases of Generic view. This strategy is reffered as a process model. It represents ways of executing process that depends on developer

SDLC:-

The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. The life cycle defines a methodology for improving the quality of software and the overall development process.

SDLC consists of following activities.

1- Planning:-
The most important part of software development is requirement gathering or requirement analysis, after it is done a scope document is created containg scope of the project.
This document is called SRS (Software Requirement Specification)

2- Design
This part contains designing of software using diagrams, ERDs etc.

3- Implementation:-
The software engineers start writing code according to client's requirements.

4- Testing:-
This is the process of finding defects or bugs in the created software.

5- Deployment:-
The software is deployed after it has been approved for release.

6- Maintenance:-
Software maintenance is done for future reference.

DESIGN CONCEPTS :-

1- Abstraction
   - i- Procedural Abstraction
   - ii- Data Abstraction
   - iii- Control Abstraction

2- Refinement

3- Modularity
   - i- Modular decomposibility
   - ii- Modular composibility
   - iii- Modular understandibility
   - iv- Modular Continuity
   - v- Modular Protection

4- Software Architecture
   - i- Structural Model
   - ii- Framework.
   - iii- Dynamic / Behavioral
   - iv- Process
   - v- Functional

5- Control Hierarchy (Program structure)

6- Structural Partitioning (Horizontal / Vertical)

7- Data Structure
8- Software Procedure
9- Information Hiding.

DESIGN CONCEPTS:-

There are 9 design concepts as follows

01- Abstraction:- (Med-high)

Clearly the software architecture and component functionalities which satisfy functional and non-functional requirements while avoiding unnecessary implementation constraints.

i- Procedural Abstraction:-

A named sequence of instructions that has a specific and limited function.

- Examples of such function
  - Display menu
  - Get user option

ii- Data Abstraction:-

A data abstraction is a simplified view of an object that

- includes features only that one is interested in
- while hides away the unnecessary details.

In OOP, it is implemented as 'class'.

abstraction & refinement ... mentary concepts

↓ Suppress low level details

↓ Reveal low-level details.

iii- Control Abstraction:-
Procedures must have a well-defined call mechanism. It implies program control mechanism without specifying internal operations.

2- Refinement:-

- It is the process of elaboration.
- Starts with the statement of function defined at the abstract level, decompose the statement of function in a stepwise fashion until programming language statements are reached.

3- Modularity:-
Decomposing large software into a number of smaller as independent as possible components, usually with the goal of placing different functionalities or responsibilities in different components.

i- Modular Decomposibility:-
It provides a systematic approach for decomposing the problem into sub-problems.

ii- Modular Composibility:-
It enables existing (reuseable) design components to be assembled into a new system.

iii- Modular Continuity:-
It is defined as small changes to the system requirements result in changes to individual modules.

iv- Modular Understandibility:-
Module can be understood as a standalone unit. (no need to refer to other modules)

v- Modular Protection:-
If unexpected condition occurs within a module & its effects are constrained within that module.

4- Software Architecture:-

Defines relationship among the major structural elements of a software.
It is basically the overall structure of the software and the ways in which that structure provides conceptual integrity for a system.

i- Structural Models:-

- Defines the components of a system (e.g. modules, objects, filters). and
- How the components are packaged and interact with each other.

ii- Framework Model:-

- increase level of abstraction.
- the strategies procedure that we have to follow and use (definition of activities).

iii- Dynamic / Behavioral Model:-

- Detect and supply methods for object model.
- Predicts behavioral & reliability aspects.

iv- Process SDLC's process models

v- Functional Model:-

- Depicts functional Hierarchy.
- e.g a design to implement particular function.

5- Control Heirarchy:-

- Also called program structure.
- Represents the organization of program components.
- Does not represent procedual aspects of software such as decisions, repetitions etc
- In OOP project, how many classes & relationship between them etc.

divide the program structure so that it can be easily defined.

6- Structural Partitioning:-

If the architectural style of a system is hierarchical, program structure can be partitioned as

i- Horizontal Partitioning:- (independant)

separate 3 branches can be defined for each major function

1- Input
2- Data Transformation
3- Output.

ii- Vertical Partitioning:-

- Control & Work modules are distributed top down.
- top level modules perform control function
- lower level modules perform operations (input, processing & output)

7- Data Structure:-

8-Feb-2018.

SOFTWARE TESTING:-

. Process of executing program with intention of finding errors.

Purpose:-

1- To demonstrate quality or proper behaviour.

2- To detect & fix problems.

Direct Objectives:-

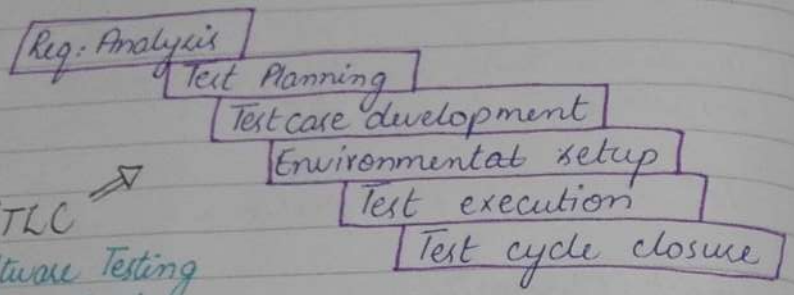
1- To identify and reveal as many errors as possible.

2- To bring tested Software after correction of identified errors & retesting, to an acceptable level of quality.

3- To perform requirement test efficiently & effectively, within budgetary & scheduling limitation.

Indirect Objective:-

- To compile a record of software errors for use in error prevention.



Req Analysis:-

Test Planning ⇒ which test needs to be performed & on what stage, who will conduct and how?, & why that test is used.

Test Case Development ⇒ Scenario on which you are testing (code or anything).

Environmental setup. ⇒ requirements & tools used for a particular testing, gathering all re.

Test execution ⇒ Start executing testing by a dummy robot etc. & generate report

Test cycle closure ⇒ Formed report.

stopping criteria => when to stop testing.

last friday

TESTING STRATEGIES :-

How you test your system? Methods etc.

- Big Bang Testing
  - simple but complex.
  - test whole project at a time not in modules.
  - It is complex because testing whole project at a time is difficult.
  - used in small projects.

i- Black Box Testing (functional test) & White Box testing (glass box testing)

Black Box Testing	White Box testing
- You need to find out outputs only while testing a project. (not concern with methods)	- internal mechanism is tested.
- functionality is tested only how it is produced (not tested)	- each line of code is tested.
	- sometimes we don't need only output, we need the coding perfectly done also.

ii- Usage based & Coverage based Testing.

only important things and tested.

stopping criteria

- → Resources Based: ⇒ if you have to test 500 tc. then 500 test cases should be test until your resources made. are finished (time, cost) ⇐ (irresponsible method)
- → Activity Based. the activity you have planned, test will be conducted according to that.

Incremental :-

Unit - test the project in increments, modules & then at last test whole project so that error in a single module can escalate first rather than having impact on whole project.

Integration :-

Validation - all the work which is integrated is validated or not.

System - what impact this testing of error has on whole system.

- Time taking as compared to big bang Testing.

pareto principle } 20% work should be checked to find 80% of errors.

Testing Principles:-

1- All test should be traceable to customer requirements.

2- Test should be plan long before testing begins.

3- The pesito pareto principle applies on software testing.

4- Testing should begin in the small and progress towards testing in large.

5- Exhaustive testing is not possible.

6- To be more effective, testing should be conducted by independent third party.

Test Case → how to test a thing (code, etc). what i/p & what conditions & with what combination they are made to test a code or to syst

Black Box Techniques:-
(functionality testing).

1- Decision Table.

- A table is made in which i/p & o/p conditions are written and o/p conditions are judged by i/p conditions.

- From rows & columns of this table test-cases are made. (i.e. how much i/p should be provided & what # of test cases should be made).

- Multiple inputs are provided to judge a single output.

- If i/ps are various, then its combination will also be much more (2^n) that it impossible to test, so from reduction techniques, they are reduced to minimize the number of test cases.

2- Equivalence Partitioning:-

- When range of data is given (ie. input is provided from 50-100), then equivalence partitioning is used.

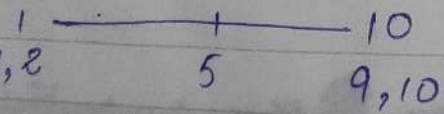
- Equal size of partitions are made & then lower bound is previous number, upper bound is after number and a middle number are tested.

L.B & U.B are also included.

3- Boundary Value Analysis:-

- Special case of Equivalence Partitioning.

- In B.V.A equal partions are not made, but it depends on tester that how much partitions and what size he made.



* Not mandatory to use all testing techniques.

- In common data Boundary value Analysis is used mostly.

White Box Testing:-

1- ~~Control flow.~~

1- Data:- Tes

- The variables which are used, what values are assignned to them and when are they used, it is changed or not after that, all these things are included here.

- To see whether unnecessary variables are made.

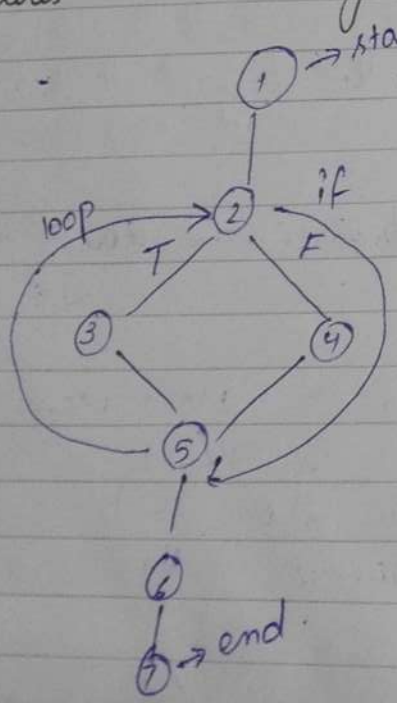
-

2- Statement Tes Coverage:-

- Statements are tested (each one of them) present in a code.
- Terminators, Brackets, ~~com~~ are included here for testing.
- Only when we are not sure where is error, then it is used.

3- Basis Path Testing:-

-

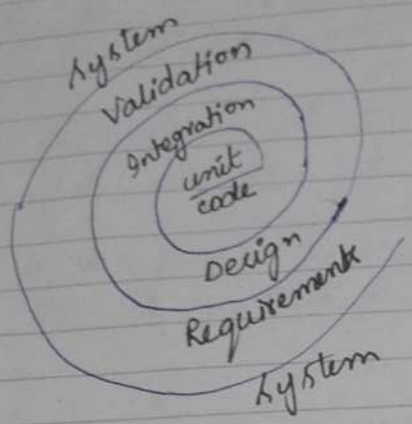


flow-graph.

- It works on flow-graph.
- In code, from initial point to final point all paths are tested.
- After knowing the paths, it is easy to test.
- used in complex code, where there are loops & multiple paths.

4- Control Flow:-

- Control Structures → if-else, switch, loops
- Control structures are tested here only, no need to test whole code.
- Also works on flow-graph.
- Main focus is on control structures not on normal



- Unit Testing → Code is tested → White box testing.

- Integration Testing → Design →

- Validation Testing → Based on Requirements. whether you did satisfy all the requirements.

- System.

INCREMENTAL TESTING:-

1- Unit Testing:-

- It can be a small table, some code, interface, data-base, it can be anything.
- You have to decide what unit is in your testing then we go for testing.
- Easy to test anything in smaller units rather than whole projects testing for best accuracy & convinience.
- Units should not be too many or too less.
- We can use either any of black box & white box testing in each unit.
- Mostly white box testing is used because most of the time black-box testing could not be applied because whole system

is not made to test its functionality only. so white box testing is used to test internal mechanism.

2- Integration:-

- Combining all units all together & check whether they all work well together.

• Top down:-
Start with larger unit (complete one) then we move downwards towards smaller units

• Bottom up:-
start with smaller ones and then move upwards towards larger ones.

- If any unit is not made at a time & the system needs them to run then stubs & drivers are used as dummy units.
- mostly used in early stages

- Driver → Bottom up
- Stub → Top down.

3- Regression:-

- To test if any change in a unit affects rest of the system, this is known as regression testing.
- helps to check the affects of the changes in integration.
- Used when there are many changes in a project (Agile, spiral etc)
- If changes are not allowed, regression testing is not used there.

3- Validation Testing:-

- To check whether all requirements are being covered in the system.

- Commonly used techniques are.

alpha
α-techniques
Beta β-techniques.

i- Beta Testing:-
- product is send to selected users (customers) to test it, where all requirements are validates.
- used only when changes affect user.
- held in a control environment.

ii Alpha Testing:-

- Test is conducted by team members rather than customers.

- When logical testing needs to be conducted, this technique is used.

4- System Testing:-

- Whole system is tested together as it specifies & satisfies a specific

i- Recovery Testing:-

- if project fails, how much time will it take to recover. through formulas & methods.

- System is being failed knowingly & then observed how it recovers with what methods & in how much time.

ii- Security Testing:-

- Used when security is a major concern in project.

iii-

- To know response time, output generated, etc, then performance

testing is conducted.

iv- Stress Testing:-

- To find out the load.

- More than its required data is given to system & then observe its behaviour.

- To know when will the system crash down.

RISK MANAGEMENT:-

↓
Doubt of failure

It can not be eliminated but it can be reduced.
In SW most common risk is customer satisfaction

To reduce risk impact factor & risk occurences are calculated.

Occurencies:-
When risk can occur,

Impact Factor:-
When risk occur, how much loss you suffer, more the loss high the impact factor.
To control & reduce risk.

- Proactive Strategy:-

- Plan before the risk occurs.
- Make alternate plans, if the software fails.

- Only those risk with higher occurrency & or highest y impact factors, proactive strategy is used.
- For all risks, proactive strategy is not used which will eventually result in accomodation of time, cost, resources etc.

Reactive Strategies:-
(fire fightening mode)

- Only when risk occurs, you do something to control it.
- Main disadvantage is that panic may occur effecting whole project.
- Those risks having moderate occurences & impact factor ~~risk facto~~ reactive strategies are used.

TYPES OF RISKS:-

1- Known Risk

- Common risk which everyone knows will occur
In s/w customer satisfaction.

2- Predictable Risk:-

- Through some calculation, we can find out that risk can occur.

During risk calculation, we only talk about known & predictable risk.

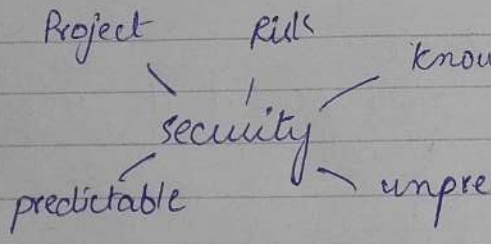
3- Unpredictable

- We can't predict it through any calculation.
- e.g. change in technology platform few days before sw launch.

Project Risk.
Any risk associated with project.
e.g. project

Technical Risk:-
Technology, law & ordi changed.
etc.

Business Risk:-
Any risk affecting your business budget, sales, profit & loss risk



Any risk can associate with many categories at a time.
(justification matters of that category)