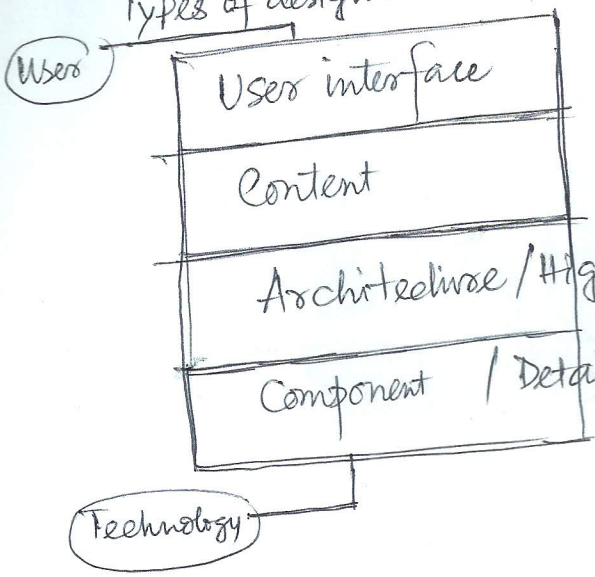
Software Design

- Good software design
- Cohesion & coupling
- Design review

Types of design:



Fisure

Major units of a system;
Component diagram,
Deployment diagram, class diagram with relations.

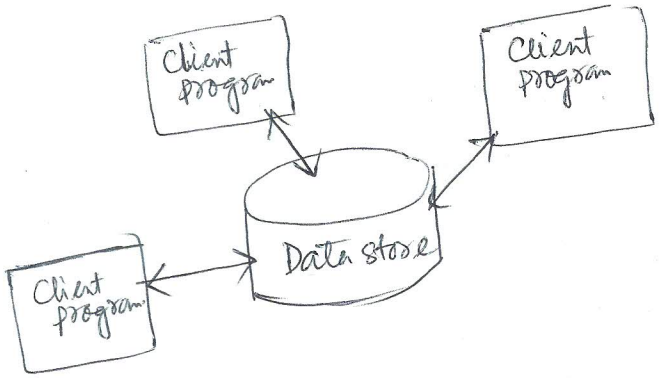
Modules & interactions among them.
eg. Structure chart; DFD
UML diagrams.

Data structure, Algorithm/Flowchart

Architectural design

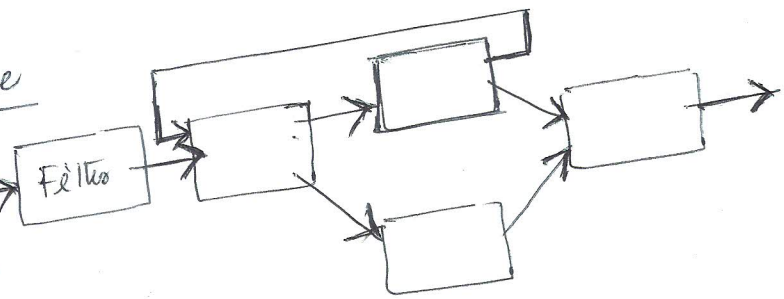
A. Data-centered architecture

A data store (eg. File, database) resides at the centers and accessed frequently by components of software

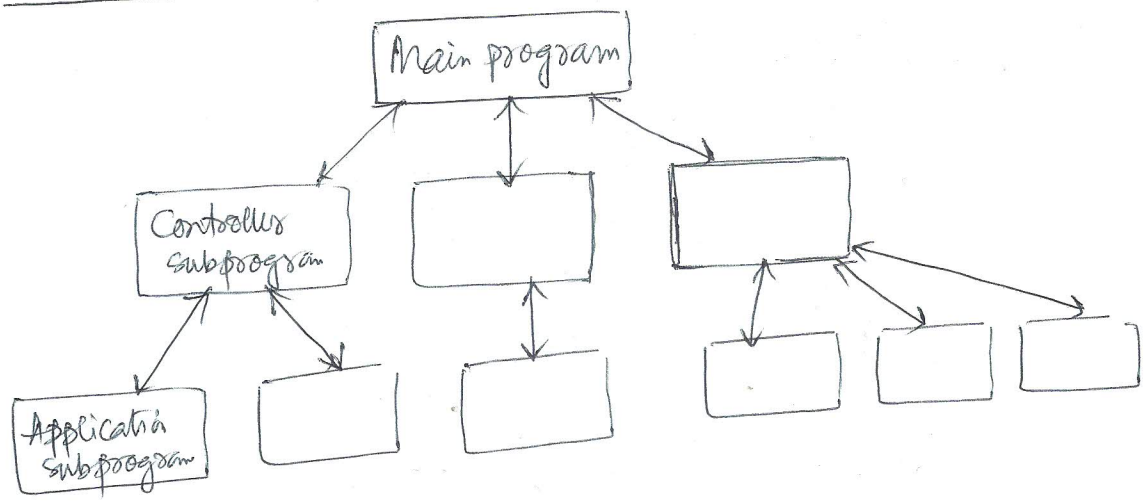


B. Data-flow architecture

Input data are passed through series of manipulative components into output data.

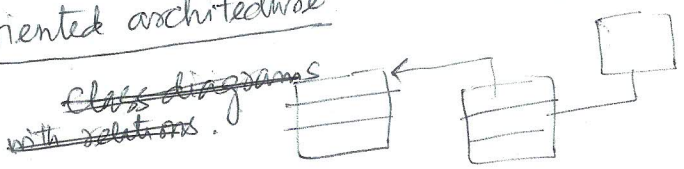


C. Call & return architecture



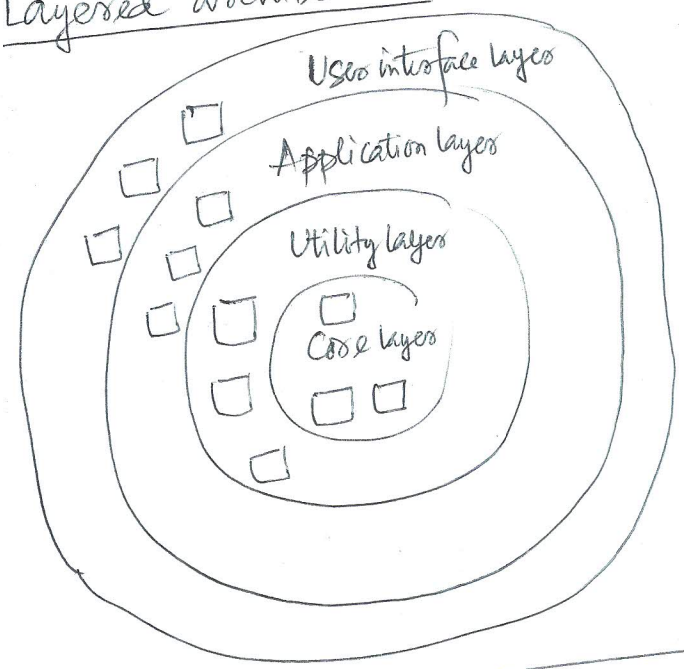
D. Object oriented architecture

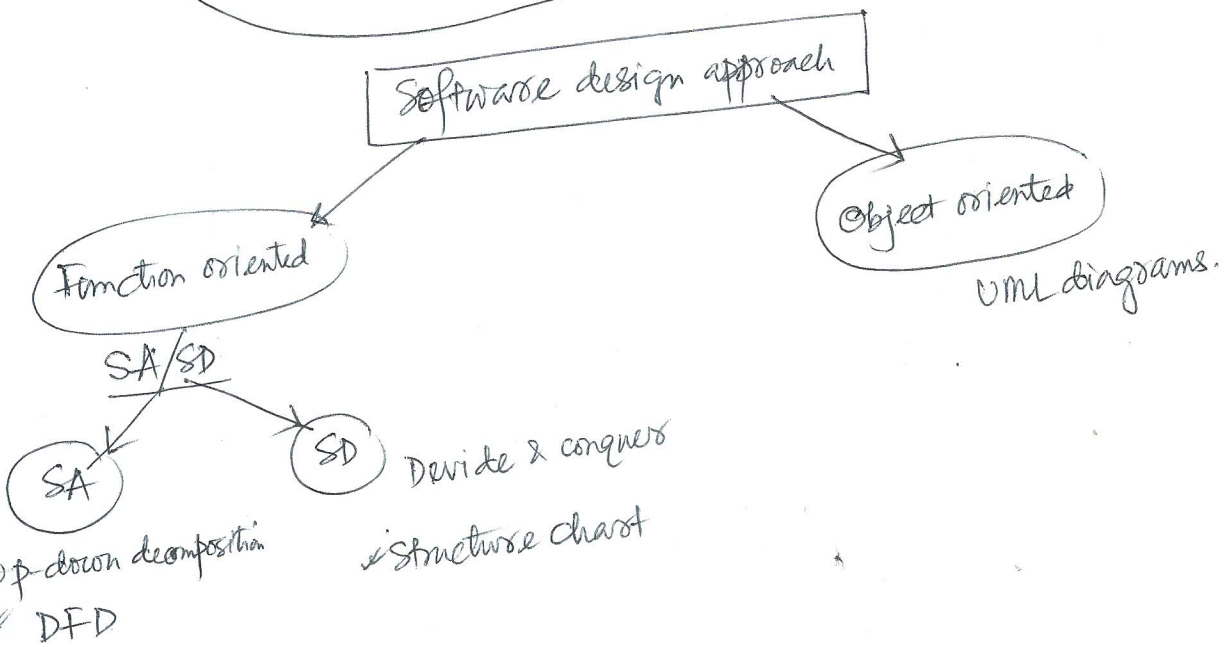
~~Class diagrams with relations.~~



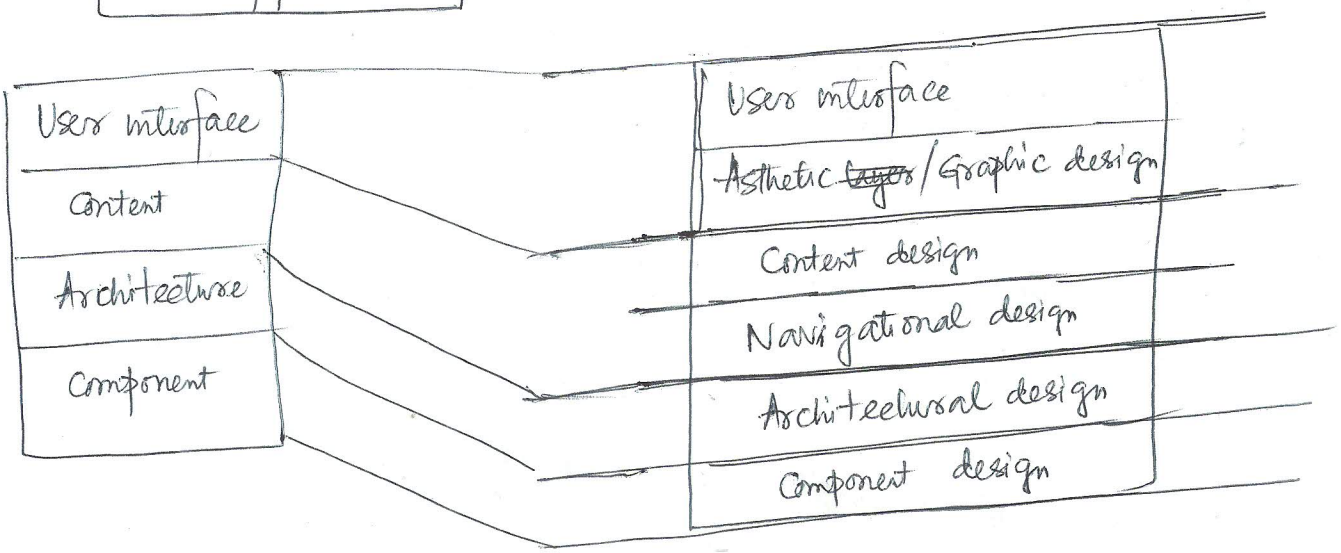
UML diagrams.

E. Layered architecture





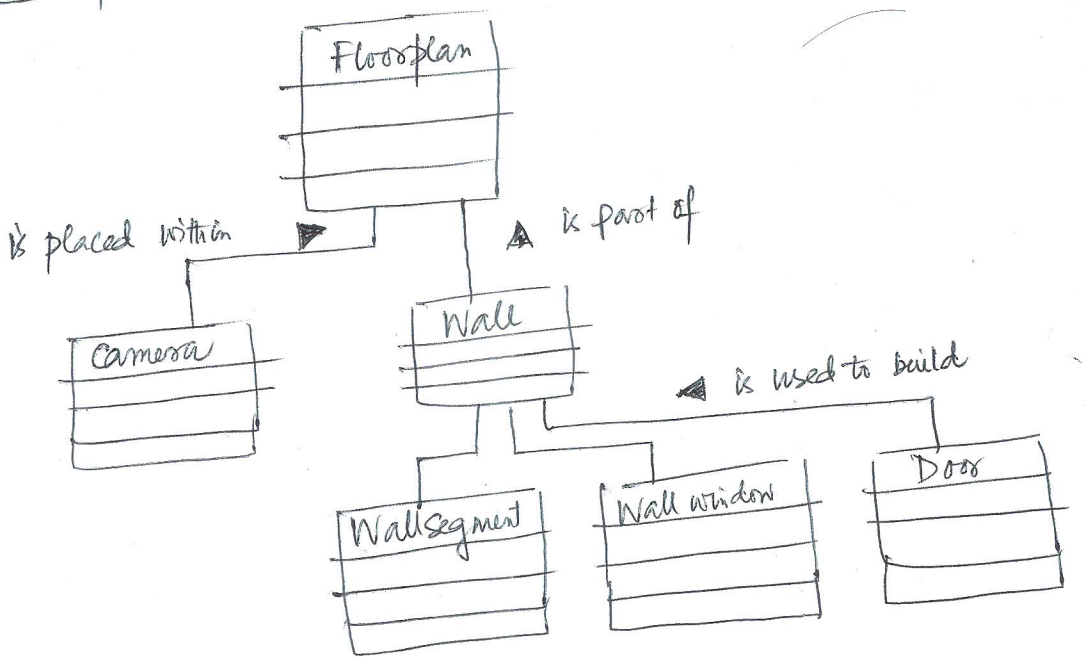
Webapp design



Navigational design

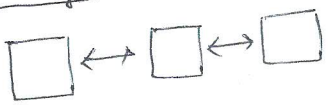
Design navigational pathways that enable users to access Webapp contents and features.

Example of content design using class diagram with relationship

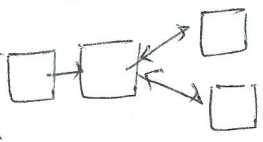


Architecture design

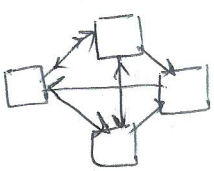
Linear:



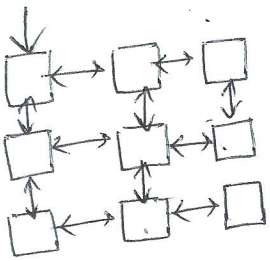
Linear with diversion:



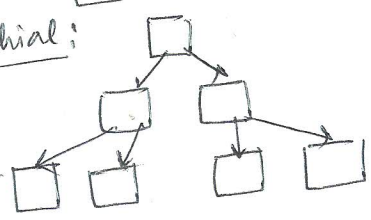
Network:



Grid:



Hierarchial:



User interface Design

Screen layout, mode of user's interaction, ~~description of navigation mechanism~~

Aesthetic/design (Graphic)

Geometric layout, text size, font, placement

Content design

Outline of the content, relationship between contents

Navigation design

Flow between contents

Architecture design

Contents are decomposed into Modules (class, function etc.)
Relationship among modules

Component design

Data structure & Algorithm

USER INTERFACE DESIGN

Good user interface design guideline

1. Easy for user

A user should not require to remember too many items at a time —— should not recall information of other page —— should not memorize command.

Interface is easy to remember —— once user learns about a command he/she should be able to use similar commands in other part of the software.

2. Language of interface

Language of user interface should be familiar to user.

Terms of technical depth (eg. related to OS, DBMS etc.) are hidden from casual user.

3. Appropriate labelling & Metaphor selection

Use appropriate labelling for navigation and understanding the content.

Metaphor (eg. symbols of whiteboard, pen, brush etc.) help in user interface development at lower effort and reduce costs for training the user —— metaphor is easily understood by the user

4. Visibility of system status

User is informed about what is going on. Absense of any response from computer for long time a novice may do system recovery/shutdown in panic. Interface should periodically inform about the progress. ——

- progress bar
- animated clock
- error message

etc.

5. Supports different types of mental models

Novice ——— Manual, help system, Guidance message

Moderately experienced — Reduced version of help & guidance + option to turn-on/off guidance.

Frequent user ——— hot-key

6. ~~Person~~ Scope for error correction and prevention

Presence of 'undo' & 'redo' operations

Control provides user to put only valid values.

7. Flexibility

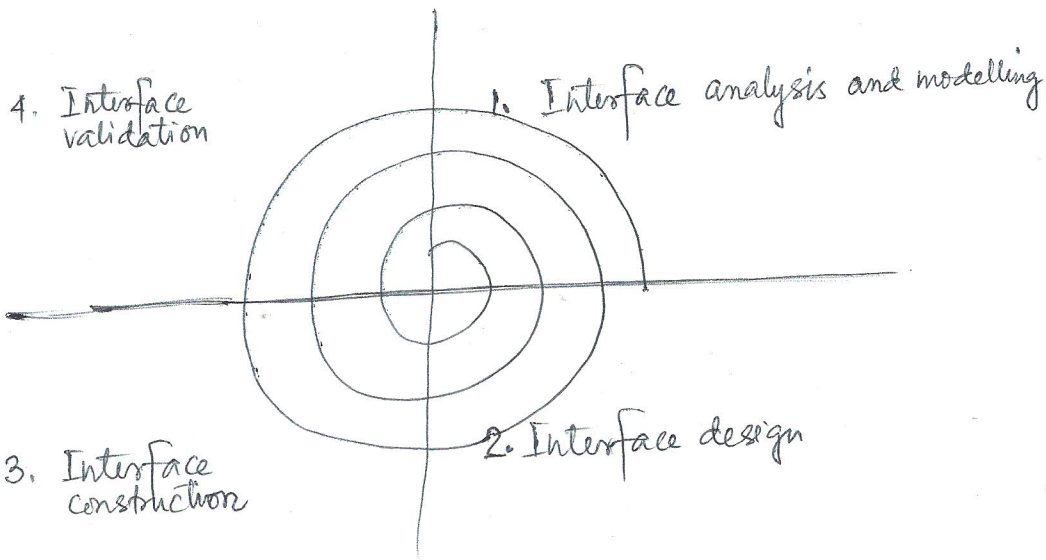
User can interact via keyboard, mouse, digital pen, touch screen, voice recognizer ...

8. Design of window

Procedural design of interface focuses on tasks whereas object oriented design focuses on object.

9. Each of the tasks / subtasks has a closure point. User does not need to unnecessary keep a task/subtask open.

Process of interface design

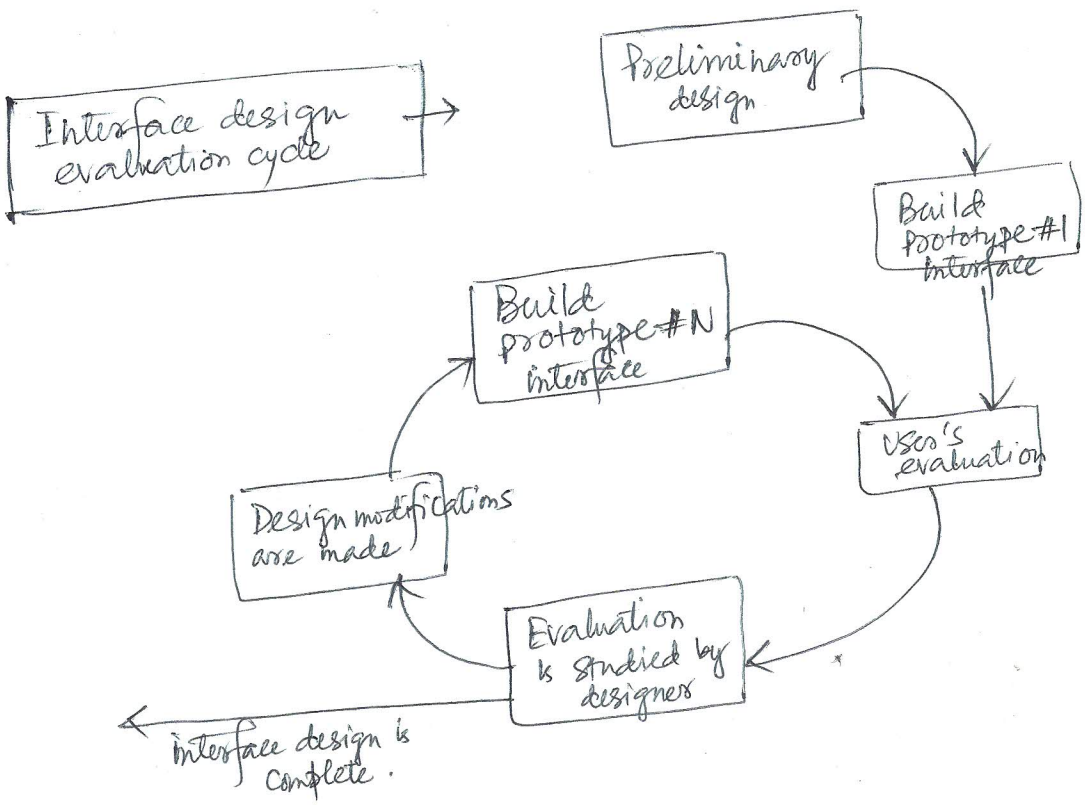


1. Interface analysis and modelling
   Understand interface

2. Interface design
   Define elements of interface

3. Interface construction
   Create prototype of interface

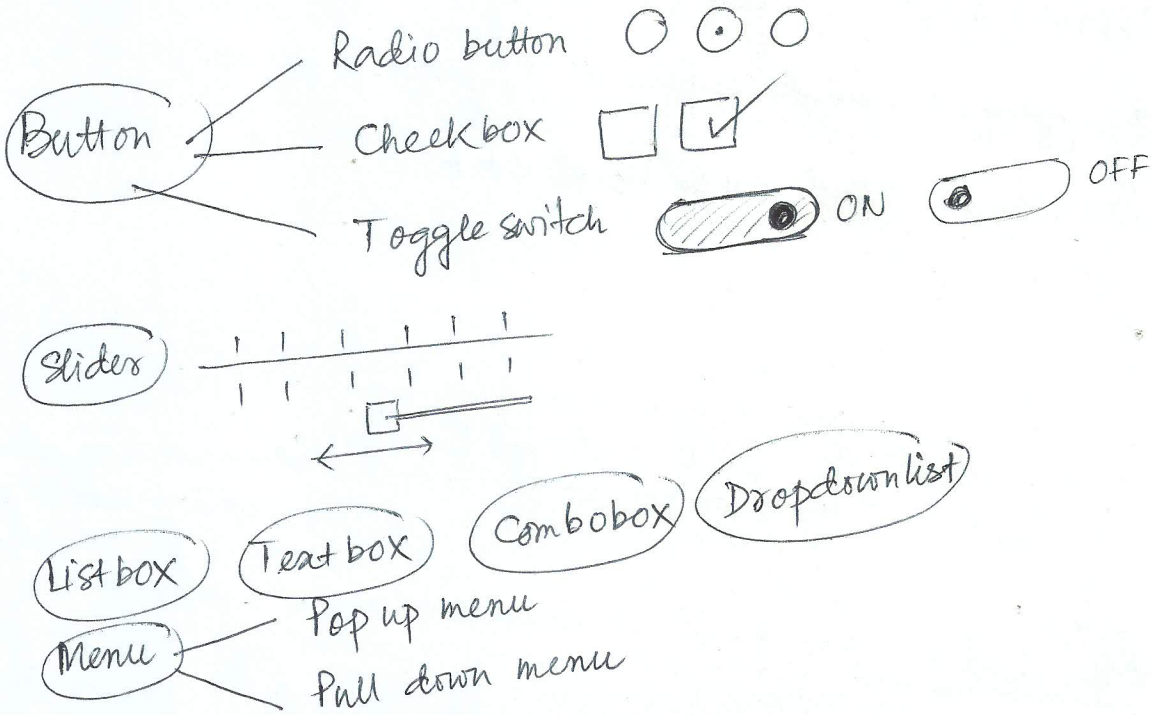
4. Interface validation
   Conduct review of interface



Graphical Widget

/Graphical control element
/Control

User interacts through controls.



Window

Frame

Label

Progress bar

Status bar

Visual Programming

Drag and drop style

- Number of icons are provided by the programming environment
- Application programmer can easily develop users interface by dragging required components (eg. menu, forms, etc.)
- Example: Visual basic, Visual c++

CLI and GUI

Command line interface
Graphical user interface

CLI: Allows user to enter commands to the terminal to perform the task

eg. UNIX, MS-DOS
Command Shell in Windows, LINUX

- User types command in
- Allow composing complex command
- Fast interaction
- Easy to develop —— can be implemented even on alphanumeric terminal
- Consume less memory

- Difficult to learn
- Need to memorize command + chance of error while typing in
- Interface with keyboard only
- Suitable for technical person only

GUI : Allows user to interact through widgets

eg. Windows, LINUX

Beginner can handle, Easy to use

Slower

Customizable options available to change the appearance

Supports K/B, Mouse, Digital pen etc.

Typing errors are often avoided

Consumes more memory

Asthetic design | Graphic design

Style Font, Size, Colour, ...

Beautification purpose

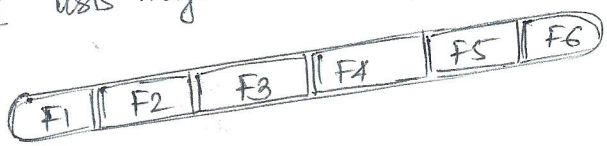
- Attractive
- Avoid visual chaos that is not pleasing for the eye.
- Proper allotment of space
  Content : 80%.
  Remaing (Navigation etc) : 20%.
- High priority elements are placed in top left corner of page ——— organize elements from top-left to bottom-right.
- Group navigation, content and functions ——— user is dissatisfied by unnecessary searching of needed information.
- Avoid scrolling bar
- Specify item considering resolution of a screen.
  ~~Percentage of space~~ | percentage of space

Navigation design

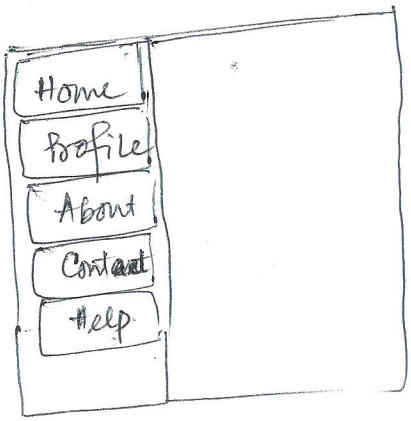
Grouping navigation

Navigation Syntax

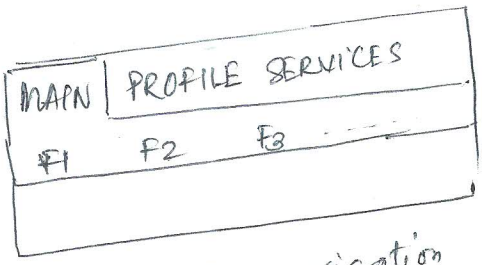
1. Link : Text based link, icon, button, graphical metaphors etc.
2. Horizontal navigation bar lists major content or functionality



3. Vertical navigation column

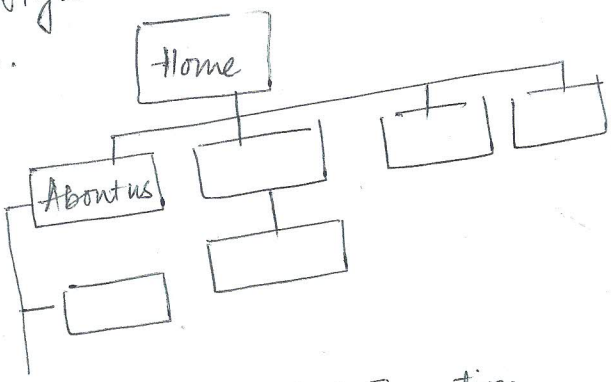


4. Tab



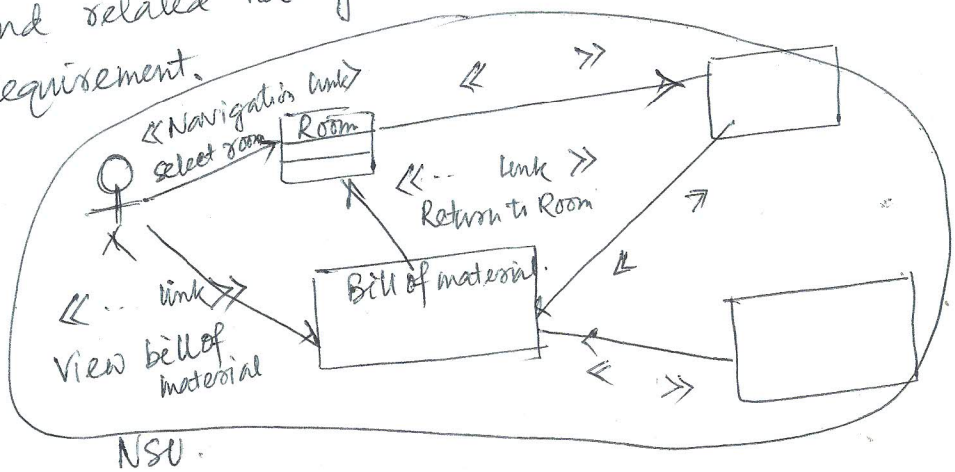
5. Sitemap

A table of content for navigation to every content within App.



6. Scrollbar

NSU (Navigation semantic unit) —— A set of information and related navigation & structure that satisfies fulfillment of requirement.



DESIGN (Architecture)

Good software design

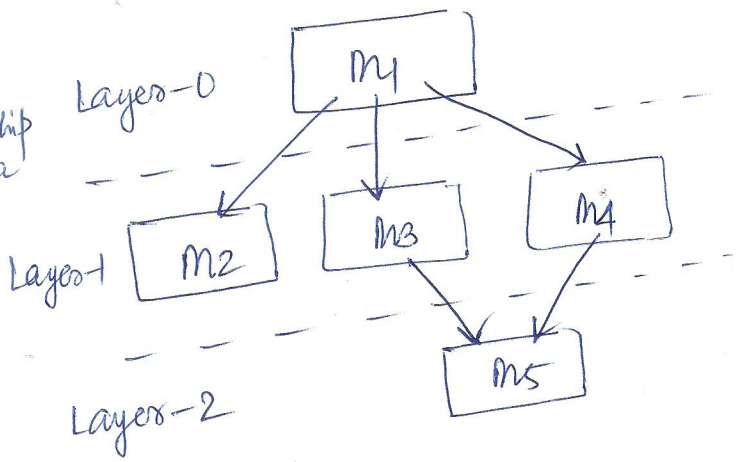
1. Decompose problem into modules

CLEAN DECOMPOSITION

2. Modules display high cohesion & low coupling

Coupling — interaction between modules.

Cohesion — strength of relationship between elements inside a module.



3. NEAT ARRANGEMENT

3. Modules are arranged in layers.

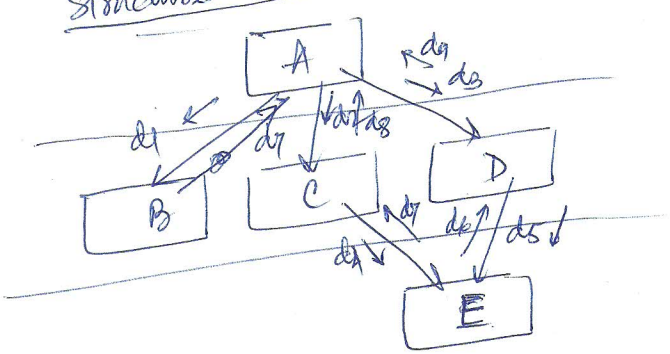
4. Abstraction: Module invoke the services of modules immidiately below it. Module at lower layer should not invoke services of module above.

5. Low FAN-OUT and high FAN-IN

FAN-OUT specifies span of control — lack of cohesion.

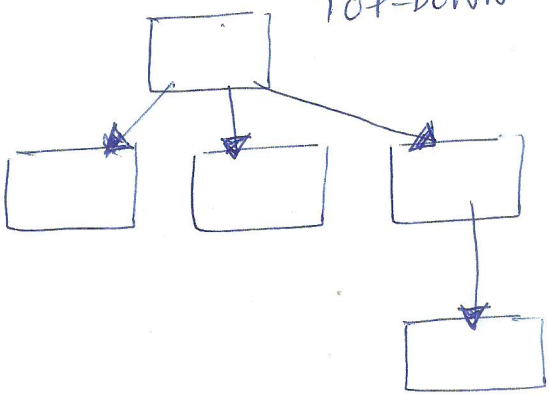
FAN-IN species code reusability

Structure chart



Design approach

TOP-DOWN



BOTTOM-UP