DATE

Software Engineering

I Software Design

(i) Software Design is a process through which requirements are converted into blueprints to construct the software.

(ii) The design should have the explicit requirements and should accomodate implicit requirements to make a good software

(iii) Implicit requirements are given by the customer

(iv) Explicit requirements are given by the people designing the software.

(v) The design should have modularity

(vi) The design should be easy & ~~understable~~ understandable for the people who will code or test the software in the future

(vii) The design should have the picture of the entire software.

(viii) The design should have data structures

Classification of Design Activities

↳ It is a process of converting the blueprints into working executable program or application that can be used.

↳ Architectural model

- It's same like floor plan of a house, just it shows a bigger picture of the software.
- It shows us how different part of the system will work together.
- The type of software we will use in the model
- The requirements of the customer
- The design & patterns to the software.

PAGE

CLASSMATE

DATE

Interface Design → user, Internal & external.

- It's like creating a detail drawing of a software.
- It show us how the information flows in & out of the system.
- It show us how different parts communicate with each other.

Component Level Design

- Component Level Design is same as drawing each room of the house.
- In component level design we know the internal of all the software components.
- It define us the data structure & algos we used for local objects.
- In OOP, we make uml diagram.

Data Structure Design

Algorithm Design.

Deployment Design

- It will show us how different software are distributed throughout the hardware to make the software work.

Design Principles

- Design isn't codding, coding isn't design.
- Design shouldn't be stuck in tunnel vision.
- Design should accomodate changes.
- Design should be flexible
- Design should be flexible enough to help us PAGE
  analyse the model.
- Design Should be reviewed to reduce conceptual error

DATE

Abstraction

- Modular: The ~~solution~~ Problem is divided into smaller & easier parts is called abstraction
- High abstraction: The solution is described in general, broad & easy terms.
- Low abstraction: The solution is described into small, easy parts to completely understand the software
- Data abstraction: It is hiding the unneccessary data & showing only the essential features.
- Procedural: It is a set of instruction to complete a task.
- Abstraction: As you build a software each step makes the solution cleaner & easy.

Refinement → It uses top down strategy

- Refinement elaborate the problem statement
- It uses top-down strategy.
- Refinement is a process where a designer elaborate the original statement, by adding all details as possible.
- Abstraction & refinement ~~is~~ complement each other.

Modularity

- The software is divided into modules which work together to solve a problem
- It uses divide & conquer technique
- It divides a complex problem into small & Managble pieces.
- ~~Eof~~ Decomposibility modularity: It divides the problem into small
- Composibility: To use previous modules in order ~~to make~~ to not start from scratch
- Understability: each module should be easy to understand.

PAGE

classmate

DATE

Continuity : Small changes should affect the module & no the system.

Error : To solve the error so it doesn't affect others

Architecture

* Software architecture show us the software structure & the different modules connected to each other.
* Hierarchial architecture means show components are connected with each other & how data work.
* A good architecture structure means detailed design that can be used by other system.

SFFDF

- Structured : How components are connected with each other
- Framework : Use common design patterns
- Process : Focus on how business & technical works
- Dynamic : How software works if there is an operation.
- Functional: Show hierarchical function.

Patterns

- A design pattern solves a design structure that help us to solve a design problem.
- Wheather the design pattern is used in current work
- Wheather the design pattern is reusable.
- Wheather the design pattern can make a similar, but structural & function different pattern.

Information Hiding

- Hidding the information & showing only the essential features.
- Information hiding is great when modification are required.

PAGE

CLASSMATE

DATE

Independance

- Software should be independent.
- The modules that software is made up of should satisfy one requirement & be completely independent of each other.
- A good software have modularity & are independent of each other, ~~because~~.
  - to reduce error
  - easier to maintain
  - reusable modules are possible.
  - independence is a key to good design & good design is key to software quality.

Cohesion & Coupling

Refactoring

It is a process of changing the software system in such way that it alters the external code & yet improve the internal structure.

Pattern Based Software Design

- A good designer can make a pattern in a problem & match them with the solution
- It's better to use old patterns rather than making new patterns
- Pattern ~~We use~~ ~~Or~~ make classes, functions
- Pattern should be flexible to accomodate new requirements.

PAGE

classmate

DATE

Pattern Based Design in Context

- Use methods, tool for architecture, mod component & user interface design.

Thinking in Patterns

- new way of thinking
- first understand the requirements & software
- Identify common pattern at a higher level of abstraction
- Start your design by focussing on bigger level picture.
- gradually move towards the inner structure.
- Keep repeating the process until you have understable, easy design.

Design task

- For user interface design issue look up through different pattern libraries.
- Compare different pattern to check which suits you the best

Pattern based Design in tables

- It is implemented using spreadsheet model

DATE

Architecture

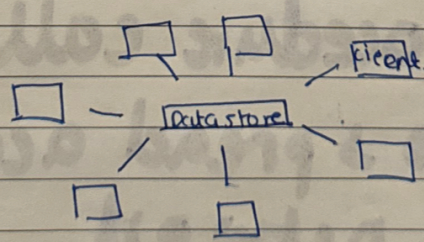
Software architecture is the software structure & how the components are connected with each other.

- set of components: It performs the function in a system
- set of connector: It connects the component (tell us how it)
- set of constraits: Rules how components are connected
- semantic model: It shows overall view of the system

Data centered
Call & return
Layered
Data flow

Data Centered Architecture.

- A (center) file or database is generally used by other components.
- Data centered architecture promotes integration ability
- New client component can be added or integrated easily
- Client can execute there process independantly.
- Software is connected to main data storage

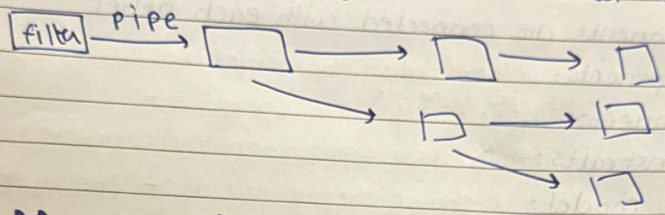


Data flow

- Data flow architecture is used when input data needs to flow through computational & manupilative components to output data.

DATE

- It's like pipe & filter pattern.
- Each filter pattern is independent
- If data flows in straight line is called batch sequential.



* Call & return
  - ↳ This style makes it easy to modify & ~~scale~~ scale
  - ↳ The main program call other components, which may call others
  - ↳ Remote Procedure call: Program control is spread across the computer network.

Main
↓
Controller

classmate ↓ Appli

PAGE

DATE

Layered:

Layered architecture, divides the application into layers. Each layer perform a task, they are arranged in a stack.

* **Concurrency** (Application needs to perform multiple task)

* **Distribution** (different components are connected in different system / location.)
  → Broker pattern

* **Persistance** → when a program stops executing it doesn't mean we have lost the program
  → It is stored in the CPU.

DATE

Data Centered [.....]
Data Flow [....]
Layered [...]
Call & return [.. Rps]

* Concurrency →
* Distribution →
* Persistance →

DATE

Cohesion

It's a degree of which a module performs one & only task.

- FLC

Functional : A module performs one specific task, Process the data & returns the value

Layered : Higher layer use the benefits of lower layer
eg: package, classes

Communication : When some data perform same operation they are put together in class

Coupling

It's a degree of which one module connected to another

* C³ SD RT IE

C³ SD RITE

→ Content	Stamp	Routine	Inclusion
→ Common	Data	Type use	External
→ Control			

External: When one part of a program connects to the external system.

PAGE

C³ SD RITE

DATE

Common: Multiple components use same global variable this may lead to error spreading across the system.

Content: One component changes another data that is internal to that component

Control: One part of program control others by sending signals

Stamp: When operation are nested making the system harder to change

Data: ~~when~~ During operation long arguements are send
↳ ↑ communication, maintaince
↓ complexity

Route: It happens when one component invokes another.

Inclusion: occurs when component A include pachages from component B

Type: It happens when one component A use data type that is defined in B

PAGE

classmate

Interface Design Rules

DATE

Put A user in Control

- Make a system or interface using user in the mind
- The interface should be easy & understable.
- Flexible user interaction
- Don't allow a user to think or write this will cause more error by the user.
- Let the user be allowed to undo their actions
- ⊙ Hide complicated details from regular users.
- Let user directly interact with what they see on a computer.

Reduce User Memory

- Design a user interface which ~~have~~ reduces user memory.
- If a user needs to remember or write there is more chance of errors.
- Use common shortcuts CTRL+S or Ctrl+P
- Use common features like undo or default setting.
- Use of real time object to make the interface.
- Make the interface in logical & organized format.

Make the Interface Consistent

- ⊛ If the inteface is same it should work the same, if the interface is different it should work the different.

PAGE

classmate

DATE

- If a user is performing a task, we need to make them understand each & every step.
- If there are different app or product the design should be the same for user friendly interaction.
- Don't change the flow of the ~~software~~ interaction, unless its a strong reason.

Process

Interface Analysis

1. First we need to analyze what kind of users will be used in that particular system.
2. Then when we get to know what kind of users are used we get to know the requirements of the users.
3. Then we do deep analysing of the problem.

Interface Design

- ~~we defines~~ that are being used in that ~~we use~~ object & function ~~for that~~ inteface.
- These should help us to get ~~what user requirement~~ the goals of the user.

Interface Constructing

- we create different prototype.
- This prototype are tested ~~by~~ in different scenarios.

PAGE

classmate

DATE

Interface Validation

- We meet the goal of the user.
- They complete all the task.
- Easy to learn
- Whether the user likes it or not.

User Analysis

- Usen Analysis → It directly talks to the user & there requirement
- Sales Analysis → The sale person talks to the user or meet them
- Market Analysis → get insights from the marketing team
- Support Analysis → Support team interact with user & get their feed back.

Task Analysis

- ~~The The System~~ The user performs task in the system, we need to know the flow.
- Use different case diagram
- Divide task into smaller steps to achieve the goal.
- Organize the task

PAGE

classmate

3.4

DATE

Reuse

* Making a software with reusable components.
* That is using components from other system.
* It reduces cost/time
* It becomes more ~~reability~~ reliability.
* Application → entire system can be reused

Component → Part of application to single object

Object → Small software components can be reused

PAGE

classmate

(1) ↑ Realibity ( )
(2) Reduce risk
(3) Faster development
(4) efficient use of experts
(5) Follows standards

CBSE

It stands for component based SE
It is made up of reuse components
OOP made it difficult to reuse the program.
CBSE components are simple, easy to use & can work independantly
They are like mini servlo
They can be as small as a program or as large as a application.

↳ less cost & time
↳ less errors
↳ more rediable
↳ Improve efficiency & quality

DATE

Provide interface

Service

Require interface

Requirements → Find reuse components → Modify requ

Modify ← Find re ← Arch

DATE

user interface principle
- ↳ User in the Control
- ↳ Reduce user memory
- ↳ Interface should be consistent.

User interface process
- ↳ Interface analyses
- ↳ Interface design
- ↳ Interface constructing
- ↳ Interface feedback
- ↳ User analysis → User
  - → Marketting
  - → Sales
  - → Support
- ↳ Work analysis
- ↳ Design analysis
- ↳ environment analysis

PAGE

DATE

* Design Activities
  - Architecture
  - component
  - User Interface → external, internal, user
  - Deployment
  - Data Structure
  - Algorithm

Pattern thinking
- Pattern Based in Context
- Thinking pattern
- Design task
- Pattern Based in Organi... tables.

* Design Process
  - Abstraction → Procedural Abstraction, High Abstraction, Low Abstraction, Data Abstraction, Abstraction
  - Refinement
  - Modularity → Decomposibility, Composibility, Understability, Error, Functionality
  - Coupling
  - Cohesion
  - Architecture → Dynamic, Structured, framework, Process, functional
  - Pattern
  - ~~Data~~ Hiding
  - Independent

Refactoring

PAGE