11/1/23

Highly Quality SRS : High Quality & Reduced cost

| Phase | PHs |
|---|---|
| Requirement | 2 |
| Design | 5 |
| Coding | 15 |
| Accep Test | 50 |
| Maintaince | 150 |

→ Additional 100 PH in req. phase, 50 new defects removed

→ Design 65% 32.5 × 5 = 162.5

Coding 2% 1 × 15 = 15

Testing ~~2%~~ 30% 15 × 50 = 750

Maintenai 3% 1.5 × 150 = 225

= 1152.5 PH

↓

→ if 50 defects are detected in later phases, cost of fixing them.

→ If we invested additional 100 hrs, in req. phase then 1152 − 100 = 1052 PHs would have been saved

x x

* 20% − 40% work in s/w development is rework (due to changes in requirements)

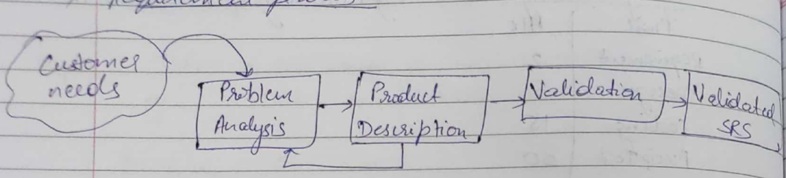
e.g. → cost of req. phase is 6% of total cost. consider a project of 50 Person Months, ⇒ req. phase consume 3 months.

→ If by spending 33% effort in req. phase, we reduce the req. change request by 33%.

→ So total rework will reduce from 10 − 20 PM to saving of 5 − 11 PM.

i.e. 10% to 20% of total cost!

* Requirement process



1) Understand the system youre going to create (behaviour, ~~constraints~~ limitation, i/p, o/p)

2) Write down or list all the requirements clearly.
3) Represent req. in words / diagrams / charts etc.
4) Specify which tools & languages will be used.
5) Put everything in a Good Quality SRS.

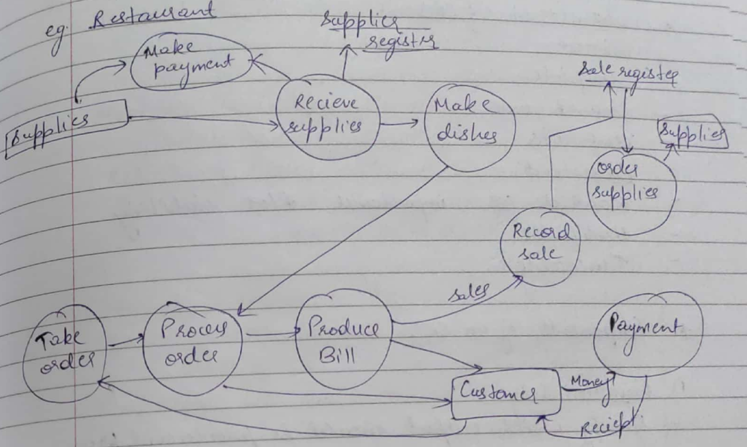
* Problem Analysis
→ Clear understanding of user needs
→ Divide & conquer approach.

Methods

1) Informal approach : interaction with client, end users, surveys, study documents, brainstorming
2) Data Flow Modeling : divide problems into functions. DFD + Data Dictionary.
3) Object Oriented Modeling : - view system as obj. classes.
   - how they interact with others
   - what services they provide (methods)
   - relation b/w objects.
4) Prototyping :- partial system is developed, used by client or end users.
   - throwaway or evolutionary.

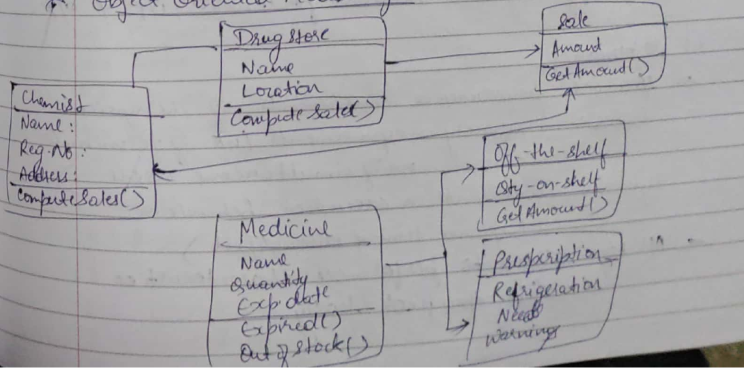
Data Flow Diagram (DFD)

eg. Restaurant



Data Dictionary
→ supplies_file = [date + [item_no + quantity + cost]*
→ order_file = [date + [menu + item no + quantity + status]*]*
→ status = satisfied | unsatisfied.

* Object Oriented Modeling



# SRS : (s/w Req. Specifications)

* Characteristics of SRS

- Correct
- Complete
- Unambigous
- Verifiable
- Consistent
- Ranked for importance &/or stability.
- Modifiable
- Traceable

* Components of an SRS

1) Functionality :

- Specify which output should be produced from the given input
- Relationship b/w i/p & o/p.
- specify description of each data i/p, source, unit of measure, range etc.
- Specify validity checks (parameters, equations, logical operation) to give o/p from i/p.
- specify system behavior for invalid i/p & o/p.

2) Performance

- specifies performance constraints of the system.
- Static : capacity requirements (no. of terminals, no. of simultaneous users..)

Dynamic : constraints on execution behaviour (Response time & throughput)

- All req. related to performace characteristics should be specified clearly.

3) Design Constraints

- Standard compliance : standards system must follow. (report, accounting procedure, audit tracing).
- Hardware limitations : s/w may have to operate on predetermined or existing h/w (restrictions on design)
- Reliability & fault tolerance -
- Security : restrictions on use of certain commands, control access to data, log of activities etc.

4) External Interface :

- specify interactions of s/w with people, h/w & other s/ws
- specify UI characteristics
- specify logical char. of interface b/w s/w & h/w (e.g. memory restrictions, load char...)
- Interface with other s/ws the system may use.

* Structure of a SRS

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, Acronyms & Abbrevations
   - 1.4 References
   - 1.5 Overview
2. Overall Description
   - 2.1 Product Perspective
   - 2.2 Product Funct<sup></sup>n
   - 2.3 User char.
   - 2.4 General constraints
   - 2.5 Assumptions & dependencies.

* Use Cases

→ a use case is a written description of how users will perform tasks on the system.

→ outlines, from user's pov, a system's behaviour as it responds to a request.

→ Each use case is represented as sequence of simple steps, beginning with user's goal & end when goal is fulfilled.

Example

Use Case 1: Put an item for auction

Primary Actor: Seller

Precondition: seller has logged in

Main Success Scenario:

1. Seller posts an item (category, description, picture etc)
2. System shows past prices of similar items to seller
3. Seller specifies starting bid price & end date
4. system accepts the item & posts it.

Exception scenarios

2. a) There are no past items of this category
   * System tells the seller this situation.

Use Case 2: Make a bid

- - - - -

* Validation of SRS

→ Ensure if SRS reflects actual requirements accurately & clearly

→ The SRS is of good quality.

Types of errors

- Omission
- Inconsistency
- Incorrect fact
- Ambiguity

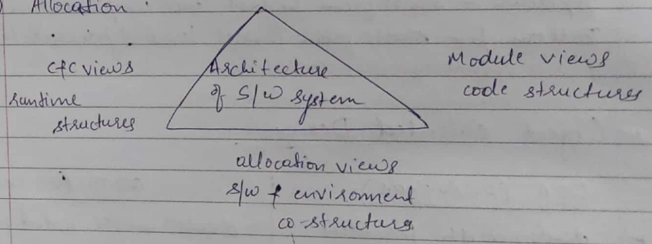
pg 61

* Software Architecture

- gives a high level view of parts of the system & how they are related to form whole system.
- divides the system in logical parts
  - solve each part independently
  - & then for relationship b/w these parts

Uses of architecture

1) Understand & communicate
2) Reuse
3) Construction & Evolution
4) Analysis
5) Module
6) Component & connector
7) Allocation.



* Design Methodology: is a systematic approach to creat a design by applying ~~so~~ techniques & guidelines.

⇒ Design process has 2 levels

① First: decide modules, specification of modules & how modules should be implemented (toplevel d

② Second: internal design of modules, how specificatio can be implemented (logical design).

## Design Principles

1) Efficiency : use scarce resources efficiently

2) Simplicity : easy to understand

3) Problem partitioning & hierarchy : divide & conquer

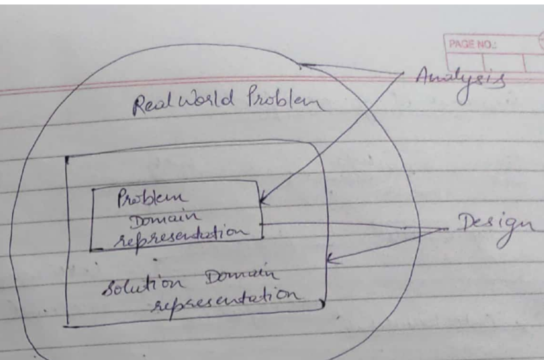
4) Abstraction :- Functional abstraction:
(external behavior without bothering internal details) - Data abstraction.

5) Modularity :- system has different modules.
- change in one should not affect other.

6) Top-down & Bottom-up strategies
Top down - starts from highest level component in hierarchy
Bottom up - starts from lowest level & proceeds to higher

## * Object - Oriented Design

| OOA (OO Analysis) | OOD (OO Design) |
|---|---|
| - deals with problem domain | - deals with solution domain |
| - understanding & specification of problem. | - it models the solution to the problem. |



## * UML (Unified Modeling Language)

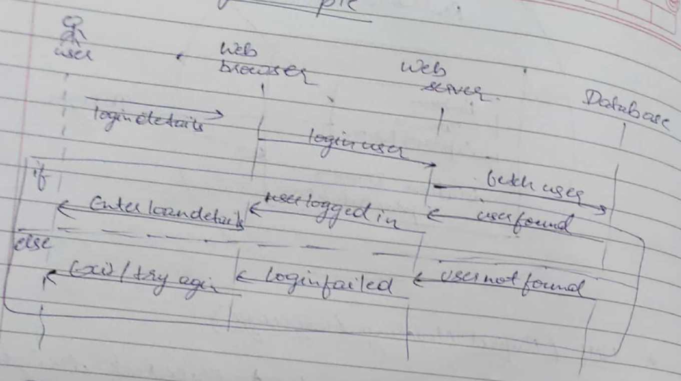
- Graphical notation of object oriented designs
- build designs, that satisfy all the requirements of the system.

(1) Class Diagram :- static structure
- classes in the system & methods
- association b/w classes
- subtypes (hierarchies)

(2) Sequence & collaboration diagram
- represents how system behaves when it performs some of it's functions.
- how objects collaborate to implement use case.

(3) Other diagrams & capabilities
- Specify ~~not~~ notations for components.
- notation for subsystems & packages.

## Sequence diag. example



## Design Methodology to produce Obj. Ori. design

- Produce class diagram
- " dynamic model & use it to define operations
- " functional model " "
- Identify internal classes & operations
- optimize & package.

### Verification

- reviews & inspection
- is each functional req. taken into account?
- limitation & constraints ?
- exception handling?
- data formats?
- interface ?
- size of data structure? guard against overflow?

## Software Testing

→ Testing is the process of executing a program with the intent of finding errors.

### Principles

1) A necessary part of a test case is a definition of the expected output or result.
2) A programmer should avoid attempting to test his or her own program.
3) Test cases must be written for input conditions that are invalid & unexpected, as well as for those that are valid & expected.
4) Examining a program to see if it does not do what it is supposed to do is only half the battle, other half is seeing whether the program does what it is not supposed to do.
5) Avoid throwaway test cases unless the program is truly a throwaway program.
6) Do not plan a testing effort under the tacit assumption that no errors will be found.
7) The probability of the existence of more errors in a section of a program is proportional to the no. of errors already found in that section.
8) Testing is an extremely creative & intellectually challenging task.

## Test Design Techniques

→ Testing cannot guarantee absence of all errors
→ Test case design is important bcoz complete testing is impossible.
→ Make test cases as complete as possible.

Black Box

Equivalence partitioning
Boundary Value Analysis
Cause-effect graphing
Error guessing

White Box

Statement coverage
Decision coverage
Condition coverage
Decision / condition coverage
Multiple - condition coverage.

1) Equivalence partitioning :

→ Divide the program in parts (equivalence class) such that one class will give similar results,

- if one test case gives errors, almost all other will also.
- if on test case did not detect an error, then others will also not.

Example

① Equivalence class 1 : Age less than 18 (minor)
Equi " class 2 : Age b/w 18 & 65 (adult)
" " 3 : Age greater than 65 (senior citizen).

② Eq. class 1 : File size less than 1MB
" " 2 : " b/w 1MB & 10MB.
" " 3 : " " greater than 10MB.

2) Boundary Value Analysis

→ Boundary conditions : situations directly on, above, & beneath the edges of input.

Examples

① 0 – 100
(-1, 0, +1) (99, 100, 101)
If range given is 0-100, test for the boundary value & ±1 of that value.

② 1 Dec 23 to 1 Jan 24
(30/11/23, 1/12/23, 2/12/23) (31/12/23, 1/1/24, 2/1/24).

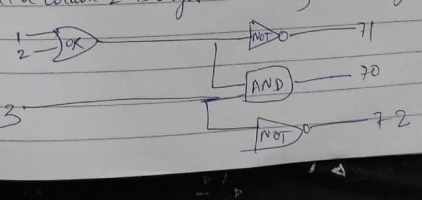
3) Cause effect Mapping

- points out incompleteness & ambiguities in specification.
- A cause is a distinct input condition
- An effect is an o/p condition or a system transformation

Example : The char in column 1 should be either A or B & in Column 2 should be a digit. If both contain
→ appropriate value then update is made.
→ If neither A nor B, message X will be displayed.
→ If column 2 is incorrect, then message Y will be displayed.

Causes
1) char in column 1 is "A"
2) char in column 1 is "B"
3) char in colum 2 is digit

Effect
1) 70 - update made
2) 71 - msg X is issued
3) 72 - msg Y is issued.



4) Error Guessing

- Some people are expert in program testing
- without using any technique.
- By intuition & experience
- By practice

* Software Reliability (trustable)

- know reliability in quantifiable terms
- testing impacts reliability, use data obtained during testing to predict reliability.
- depends on quality of testing.
- Reliability = failure free operation of that prodlu for given time duration.
- If X is random variable that represents the life of a system, Reliability is the probability that the system has not failed by time t.

$$R(t) = P(X > t)$$

- can also be specified as Mean Time to Failure (MTTF). MTTF represents expected lifetime of system.

Estimation

→ It is a process of approximating time & cost of completion of the project.

Why estimate Time & cost?

- to support good decisions
- to schedule work
- determine how long a project takes & its cost -
- develop cash flow needs
- determine how well project is progressing.

Consider this:

- Inaccurate estimates lead to fasle expectations & customer dissatisfaction.
- Accuracy is improved with greater efforts.
- life line for control.

Factors influenzing the quality of Estimates

1) Planning Horizon : current events are accurate but distant events are not.
2) Project Duration : longer the project, difficult to estimate
3) People : skills of people.
4) Project structure & Organization.
5) Padding estimates : padded for safety & risks
6) Other factors : - equipment down time
   - Holidays
   - Legal limits
   - industry standards.

Estimating Guidelines

- Responsibility : responsible people
- several people to estimate
- Normal conditions
- Time units
- Independence
- Contingencies
- Add Risk Assessment.

* Top Down approach (Estimation)

- use experience
- combined experience of seniors & managers
- Use ratios or surrogates
- Historical data with customization.
- Function Point Analysis
- learning curves. repetitive tasks