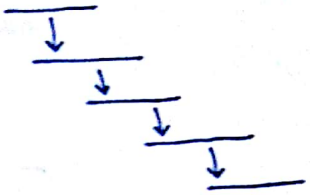
Software Engineering

① Phases of Software Development (Waterfall Model):



1. Feasibility Study :
2. Requirement Analysis and Specification (SRS document)
   - Software Requirement Specification
3. Design (Unified Modelling Language)
4. Coding and Unit Testing
5. Integration and System Testing
6. Maintainence

- We can't go back and rectify our error in this model.
  Eg: If after integration, some design issues appe

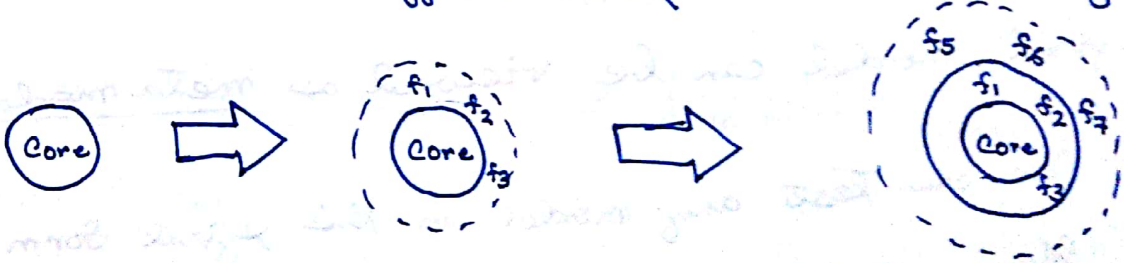
② Iterative Waterfall Model (If this feedback path is given)

3. Prototyping Model

When the user is not aware of system requirements.

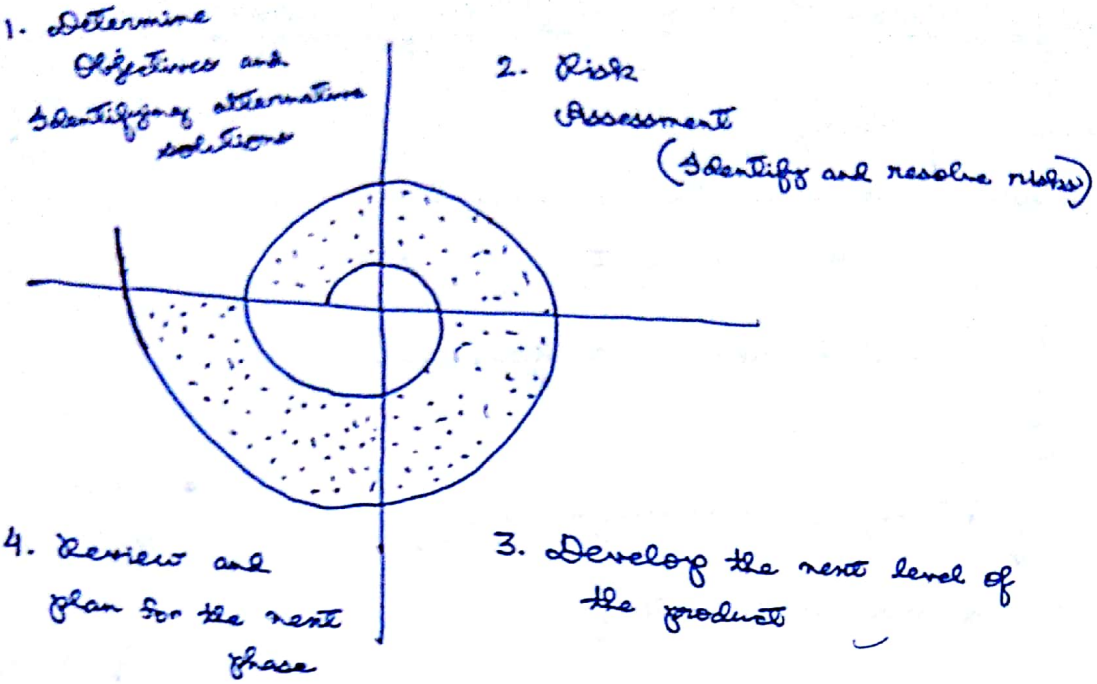
4. Evolutionary Model (Incremental Model)

SRS Document can be appended here (it is better not to change the e



SRS

5. Spiral Model



i) Each loop of the spiral represents a phase of the software development process, eg. the inner most loop might be associated with feasibility study.

ii) The exact number of loops is not fixed.

iii) The radius of the spiral at any point represents the cost incurred till then.

iv) The angular dimension represents the progress made in the current phase

Imp

v) Spiral model can be viewed as meta model. → Model of the Model

— We can test any model in this spiral form, where each spiral represents a ph

Hence it is called meta model.

Requirement Analysis and [illegible]

1. Requirement gathering and analysis
2. Requirement Specification

1.i) Requirement gathering —

a) Interviewing

ii) Analysis: To clearly understand the exact requirements of the customer.

Questions:

- What is the problem?
- What are possible solutions of the problem.

Resolving various requirement problem —

i) Anomaly: Ambiguity in the requirement, eg. if the temperature is high then the heater should be switched off.

ii) Inconsistencies: One of the requirements contradicts another.
eg. one end user: If the temperature is high then heater should be switched off.
another end user: If the temperature is high then water shower should be turned off.

iii) Incompleteness: Some of the ~~not~~ requirements have been overlooked.

Cause: Inability of the customer to visualize and anticipate all the features

2. ~~Requirem~~ SRS Preparation (Requirement Specification)

Among all the document produced during the life cycle of the software development, writing the SRS doc is the toughest. Different people need the

SRS doc for many purposes, eg: i) users, customers, marketing personell: The goal is To ensure that the system as described in the SRS doc will meet their needs.

ii) Software Developers: The goal is To prepare exactly what is required by the customers.

iii) Test Engineers: To ensure that the system is understandable from a functionality point of view so that they can test the software and validate its working.

iv) Project Manager: Estimate the cost ed Time of the project.

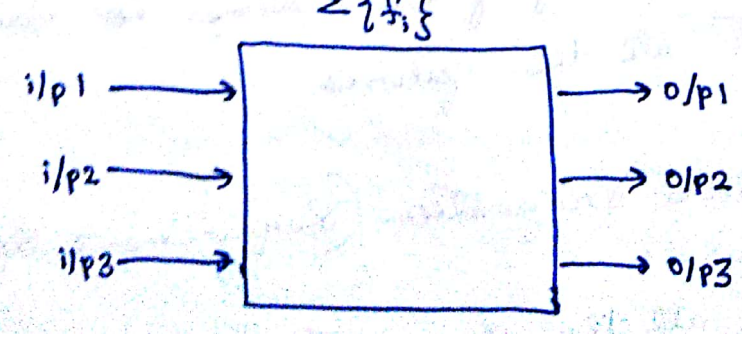
Contents of the SRS Document:

i) Functional requirement

ii) Non-Functional req.

iii) Goal of implementation

Functional Requirement:



i) Reliability issues

ii) Security issues

iii) Human Computer Interface issues

Techniques for Representing Complex Logic

Amazon Search (Example)

R1 : Search items → Description

. . . . . .

R1.1 : Write the name of the item → Options

(Eg. ISBN no. for book)

. . . . . → Some Criteria may also be included

R1.2 : Display the search item → How things will be displayed

. . . . . .

Amazon Search

R1 : Search items → Description

. . . . . .

R1.1 :

Case Study:

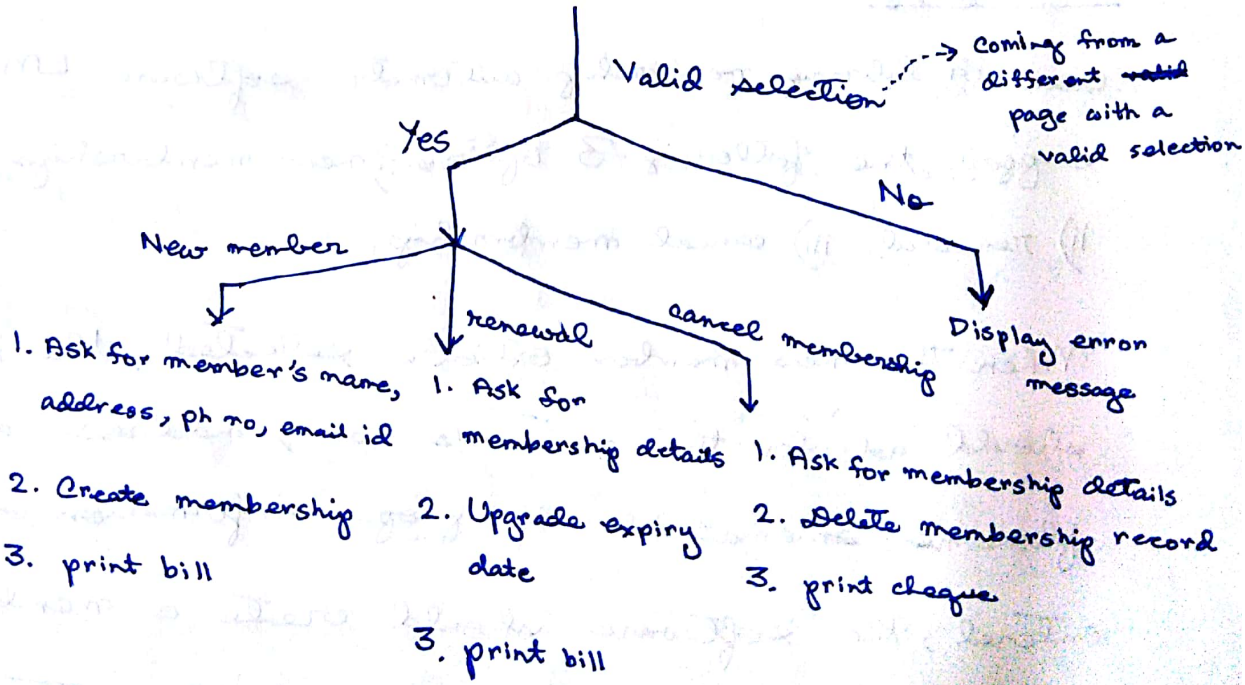
A library membership automation software LMS should support the following 3 options : i) new memberships, ii) renewal, iii) cancel membership.

When the new member option is selected, the software should ask for the member's name, address and phone no, email id. If proper information is entered, the software should create a membership record for the new member and print a bill for the annual membership charge.

If the renewal option is chosen, the LMS should ask the member's name and his membership number and check whether he is a valid member or not. If the member details entered are valid

then the membership charge payable by the member should be printed. If the membership details entered are invalid, an error message should be displayed. If the cancel membership option is selected and the name of a valid member is selected, then the membership is cancelled, a cheque for the balance amount due for the member is printed, and his membership record is deleted.

Decision Tree



Conditions				
Valid selection	No	YES	YES	YES
New member	-	YES	NO	NO
Renewal	-	NO	YES	NO
Cancellation	-	NO	NO	YES

Actions				
Display error message	X			
Ask for member's name etc.		X		
Create membership record		X		
~~display~~				
Print bill		X	X	
Ask for membership details			X	X
Update expiry date			X	
Delete record.				X
Print cheque				X

Decision Tree VS Decision Table

1) **Readability**: When no. of conditions are small, Decision tree should be used in SRS. To look at every possible ~~card~~ combination of conditions, Decision Table is used.

2) **Order of Decision Making**: For multi-level decision making Decision Tree is used. Decision Table abstracts the order of decision making.

3) For very complex decision logic, Decision Tables is preferred.

Design

Problem Partitioning:

Dividing a problem into smaller pieces recursively so that each piece can be conquered seperately.

Two different Design Approaches:

1) Function Oriented Design: A system is viewed as something that performs a set of functions. Starting at the high level view of the system, each function is successively refined into more detailed functions.

2) Object Oriented Design: In the object oriented design approach, the system is viewed as a collection of objects.

Example: Library Automation System —

i) Function Oriented Approach:

```
create_new_library_member()
delete_existing_member()
issue_book()
search_book()
update_member_record()
```

Now consider create_new_library_member().
~~Enter details of member~~ The sub-modules could be — enter_details_of_new_member(), ~~assign~~ assign_membership_number(), print_bill(), generate_card().

ii) Object Oriented Approach:

staff
books
member
librarian inherits staff

Each of them are considered as an object with its own data, and functions (methods) to operate on these data.

Function Oriented Software Design

1) Structured Analysis (SA):

- Top-down decomposition approach
- Divide and conquer
- graphical representation using dataflow diagram (D

- **Cohesion**: How the pieces are related to each other in a single module. More cohesion is better.
- **Coupling**: How two or more modules are interconnected with each other. Less coupling is better.

Cohesion:

Cohesion refers to the degree to which elements of a module belong together. It is a result of how strongly related each piece of elements of a module is. When function of one module co-operates with another function to perform single objective, then the module is said to be in good cohesion. Cohesion is a measure of functional strength of a module.

Classification of cohesion:

1) **Co incidental Cohesion**: A module is said to have co-incidental cohesion if it performs a set of tasks that relate to each other very loosely. In this case, a module contains a random collection of functions. It is likely that the functions have been put in the module out of pure co-incidence without any thought or design. Different functions in the module carry out different activities.
Eg: In a transaction processing system, the gets_input(), print_error(), summerize_menu() functions are grouped into one module. The grouping does not have any

relevance to the structure of the problem.

2) Logical Cohesion: A module is said to be logically cohesive if all elements of the module perform similar operations, eg. error handling, data ~~input~~ input etc.

Eg: In case there a set of print functions generating different output reports are arranged into a single module, such as ~~module contain sets~~ salary, grade sheet, manual report etc. Here parts of the modules are logically categorized to do same thing even if they are different by nature.

3) Temporal Cohesion: When a module contains functions that are related with other, and all the functions must be executed in the same time span, then ~~the module is~~ said to exhibit Temporal cohesion.

Ex: When a computer is booted, several functions need to be performed at the same time one by one such as initialization of memory, loading OS etc.

Another eg: Functions which is called after catching an exception, which closes open files, gives notifications to users, create error log etc.

4) Procedural Cohesion: A module is said to possess procedural cohesion if the set of functions of a module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective.

Eg: The algorithm for decoding a message.

5) **Communicational Cohesion:** It is called **Informational Cohesion.**

A module is said to have communicational cohesion if all functions of the module refer to or update the same data structure, eg. a set of functions defined on an array or stack.

6) **Sequential Cohesion:** A module is said to possess sequential cohesion if the elements of a module form the parts of a sequence where the output of one function is input to the next.

Eg: In a transaction procesing system, gate_input(), valid_input(), sort_input() functions are grouped into one module.

7) **Functional Cohesion:** It is said to exist if different elements or parts or functions of a module co-operate to achieve a single task, eg. a module containing all the functions required to manage employee's pay roll exhibits functional cohesion such as compute_overtime(), compute_workhour(), compute_deductions().

Coincidental	Logical	Temporal	Procedural	Communicational	Sequential	Functional

Low ——————→ High

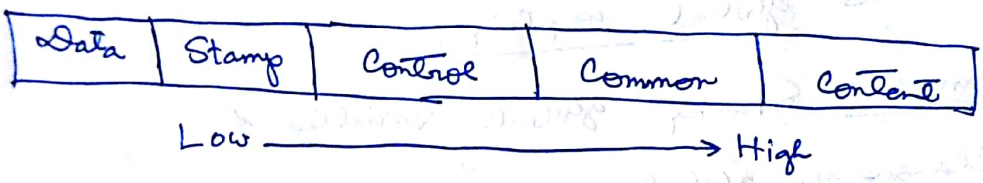
Coupling

Coupling between 2 modules/classes/components is a measure of the degree of interdependence or interaction between two modules.

Classification of coupling —

Classification of the different types of coupling will help to quantitatively estimate the degree of coupling between 2 modules.

5 types of coupling can occur between any 2 modules. These are shown below by lower coupling to higher indicated by an arrow —



1) Data Coupling:

Two modules are data coupled if they communicate by using an elementary data item as for example an integer, a float, a char etc such as passing an float value through a function calculating square root.

This data item should be problem related and not used for the control purpose.

2) Stamp Coupling:

Two modules are stamp coupled if they communicate using a composite data item such as a record in Pascal or a structure in C.

It exists between two modules if data from one module is used in another module to direct the order of instructions execution.

Example: A flag set in one module and tested in another module.

4) Common Coupling:

Two modules are common coupled if they share data through global data items. It is also known as global coupling.

Example: Sharing global variables.
Changes in global variables leaves changes in all modules.

5) ~~Content~~ Content / Pathological Coupling:

It exists between two modules if they share code.

Example: A branch from one module into another module or if one module changes internal work of another module as for accessing local da of another module.

No coupling exists when modules do not communicate with each other.

High coupling among modules make design solutions easier to understand and maintain, increase development effort, harder to reuse and Test particular model, assembling of modules may require more effort, change in 1 module usually leads a ripple effects and difficult modules in independent way.

Data Flow Diagram

Primitive Symbol:

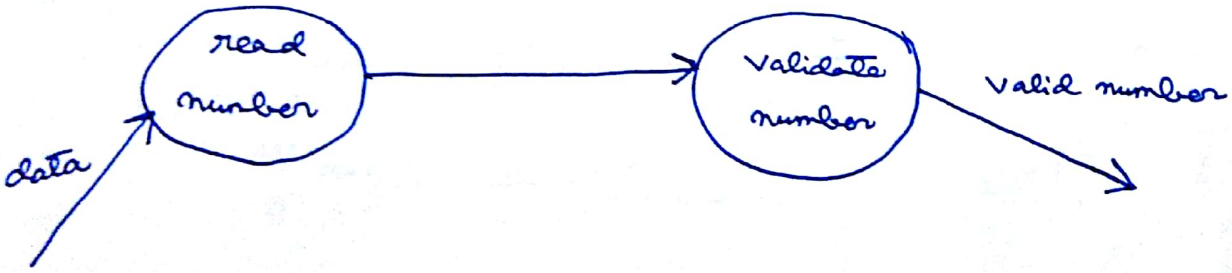
External Entity: Librarian, Library Member, Doctor

Process: Functions

Data Flow: This represents the data flow between 2 processes and between an external entity and process.

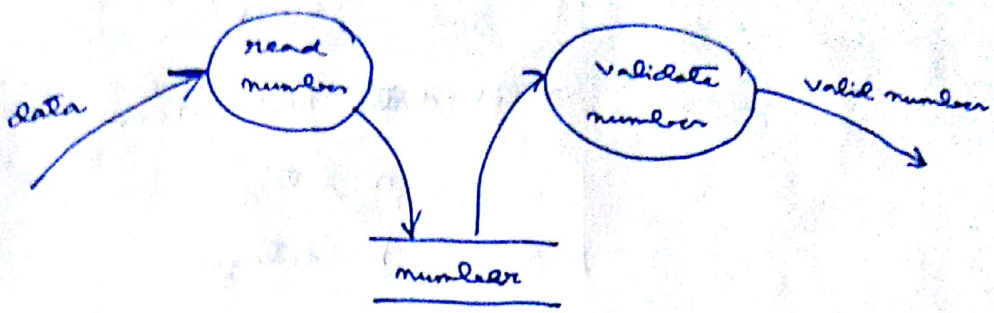
Synchronous Operation:

If two bubbles are connected directly, then they are synch



'Validate number' bubble can start processing only after th 'read number' has supplied data to it.

If two bubbles are connected through a data store, then it is called asynchronous.



Data Dictionary:

Every DFD model of a system must be accompanied by a data dictionary. Data dictionary can be defined as a centralised repository of information about data such as meaning, relationships, origin, uses and format.

Developing the DFD Model of a System

Context Diagram:

Most abstract data flow representation of a system. Also called level-zero DFD.

Example:

RMS calculating software

A software system called RMS calculating software reads 3 integral nos from the user in the range between -1000 and +1000, and determines the root mean square of the 3 input numbers and then displays it.

Solution:

Data Dictionary:

data_items : {integer} 3

rms : float

valid-data : data_items

a : integer

b : integer

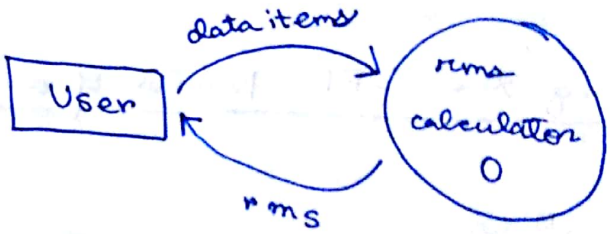
asq : integer

bsq : integer

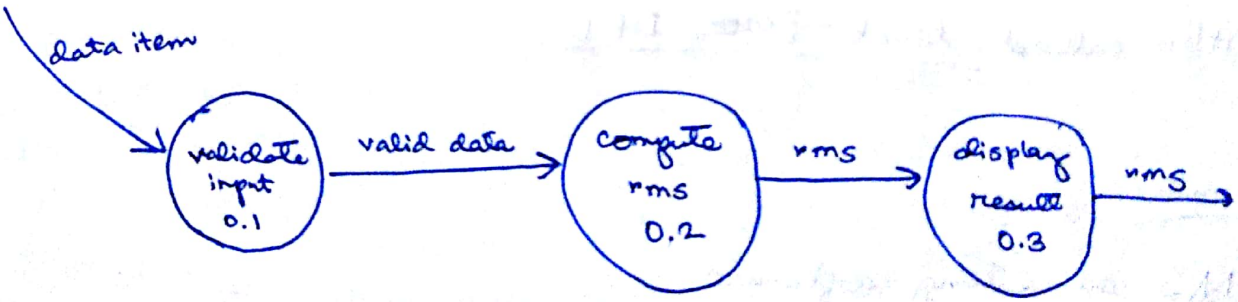
csq : integer

msq : integer

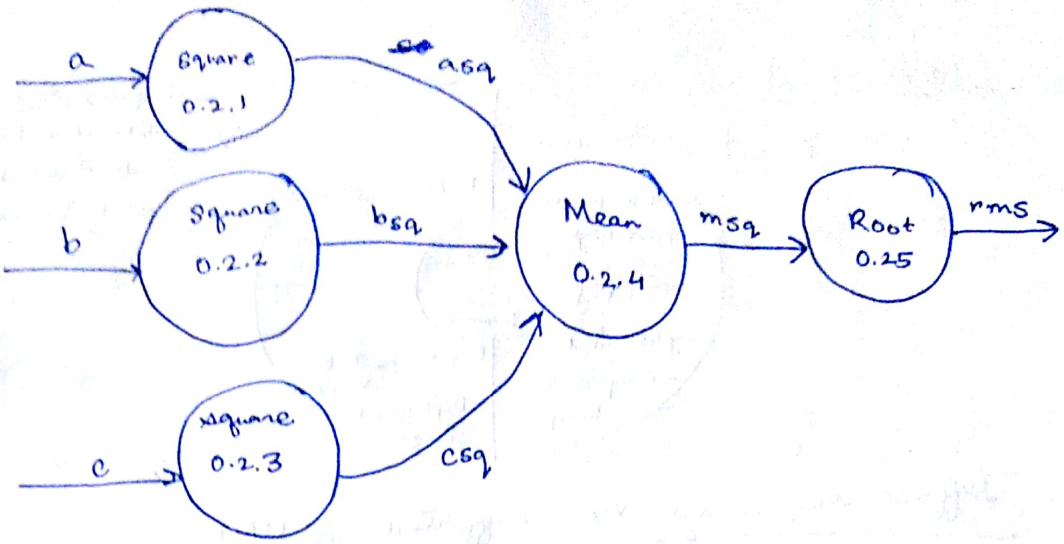
Context Diagram (Level 0 DFD):



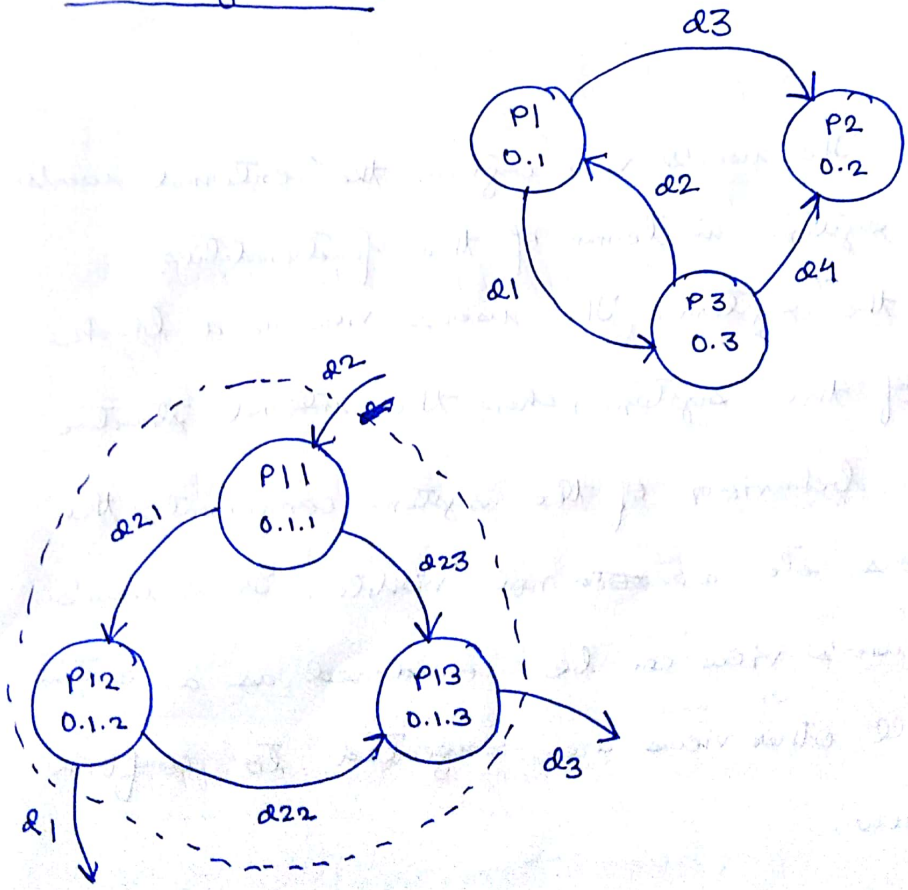
Level 1 DFD:



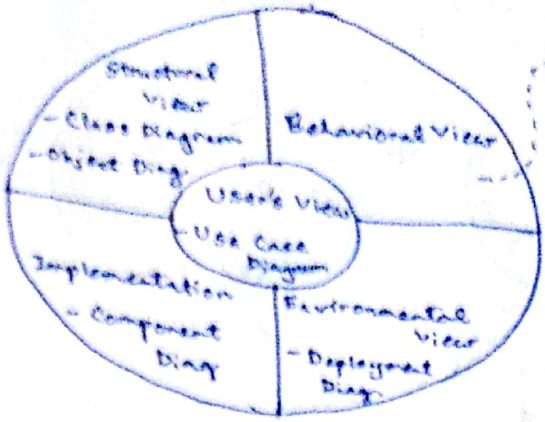
Level 2 DFD:



Balancing DFDs:



The data that flow into or out of a bubble must match the data flow at the next level of DFD. This is known as balancing a DFD.



Different diagrams and views supported in UML

UML can be used to construct 9 different types of diagrams to capture 5 different views of a system — 5 views given in diagram.

User's View: The user's view captures the external user's view of the system in terms of the functionalities offered by the system. The user's view is a black box view of the system, where the internal structure and dynamic behavior of the system components, the implementations etc are ~~not~~ not visible. This is user's view. The user's view can be considered as a central view and all other views are expected to conform to this view.

Structural View: The structural view defines the kinds of objects (in classes) ~~and~~ important to the understanding of the working of a system and its

implementations. It also considers the relationships among classes (objects). The structural model is also called Static model since the structure of a system does not change with time.

Behavioral View: It captures how objects interact with each other to realise the system behaviour. The system behaviour captures the time dependent (dynamic) behaviour of the system.

Implementation View: It captures the important components of the system and their dependencies.

Environmental View: It models how the different components are implemented on different uses of hardware.

Coding

Objective:

To transform the design of a system as given by its module specification into a high level language code and to unit test this code.

Coding Standard:

Reasons:

i) Uniform appearance
ii) Sound understanding
iii) Encourage good programming practices

Several Rules:

i) Variable naming
ii) Code layout
iii) Error return conventions
iv) Commenting
v) Global Variable Declarations → How many global variables one can declare?

Good software development organizations usually develop their own coding standards and guidelines.

Some General Coding Standards:

i) Rules for limiting the use of globals.
ii) Contents of the headers
   - Name of the Module
   - Date
   - Author's Name

— Modification History
— Synopsis of the module

iii) Naming conventions for global, local variables and constants.

iv) Error return conventions and exception handling mechanisms.

Some General Coding Guidelines:

1) Not to use clever coding techniques.

2) Not to use 'difficult to understand' code.

3) Avoid obscure side effects.
— The side effects of a function call include modification of parameters passed by reference, modification of global variables and I/O operations. An obscure side effect is one that is not obvious from a casual examination of the code. Eg: If a global variable is changed in a called module, it becomes difficult to understand the code.

4) Not to use an identifier for multiple purposes, eg. using loop variables for storing the final results.

5) The code should be well documented (Comments).

6) The length of any function should not exceed 10 source lines.

7) Not to use goto statements.

Code Reviews:

Code review is carried out after successful compilation.

1. **Code Walkthrough:** An informal code analysis technique.

After a module has been coded, it is successfully compiled and tested on some test cases, the algorithmic and logical (as well as syntax) errors are eliminated.

2. **Code Inspection:** Aim is to discover some common types of error caused due to oversight and improper programming.

eg: i) Use of uninitialised variables
ii) Jumps into loops
iii) Non-terminating loops
iv) Incompatible assignments
v) Array indices out of bounds.
vi) Improper storage allocation and deallocation
vii) Mismatches between actual and formal parameters in procedure calls.

3. **Clean Room Testing:** Pioneered by IBM, this type of testing relies heavily on walkthroughs, inspection and formal verification. The programmers are not

Code Review:

Code review is carried out after successful ~~compilation~~ compilation.

1. **Code Walkthrough:** An informal code analysis technique

After a module has been coded, it is successfully compiled and based on some test cases, the algorithmic and logical (as well as syntax) errors are eliminated. * Next Page

2. **Code Inspection:** Aim is to discover some common types of error caused due to oversight and improper programming.

eg: i) Use of uninitialised variables
ii) Jumps into loops
iii) Non-terminating loops
iv) Incompatible assignments
v) Array indices out of bounds.
vi) Improper storage allocation and deallocation
vii) Mismatches between actual and formal parameters in procedure calls.

3. **Clean Room Testing:** Pioneered by IBM, this type of testing relies heavily on walkthroughs, inspection

allowed to test any of their code by executing the code, other than doing some syntax testing using a compiler.

* From last Page

Code review practices ~~code~~ fall into 2 main practices — formal code review and lightweight code review such as Fagan inspection involves a careful and detailed process with multiple participants and multiple phases in which software dev. attend a series of meetings and review code line-by-line.

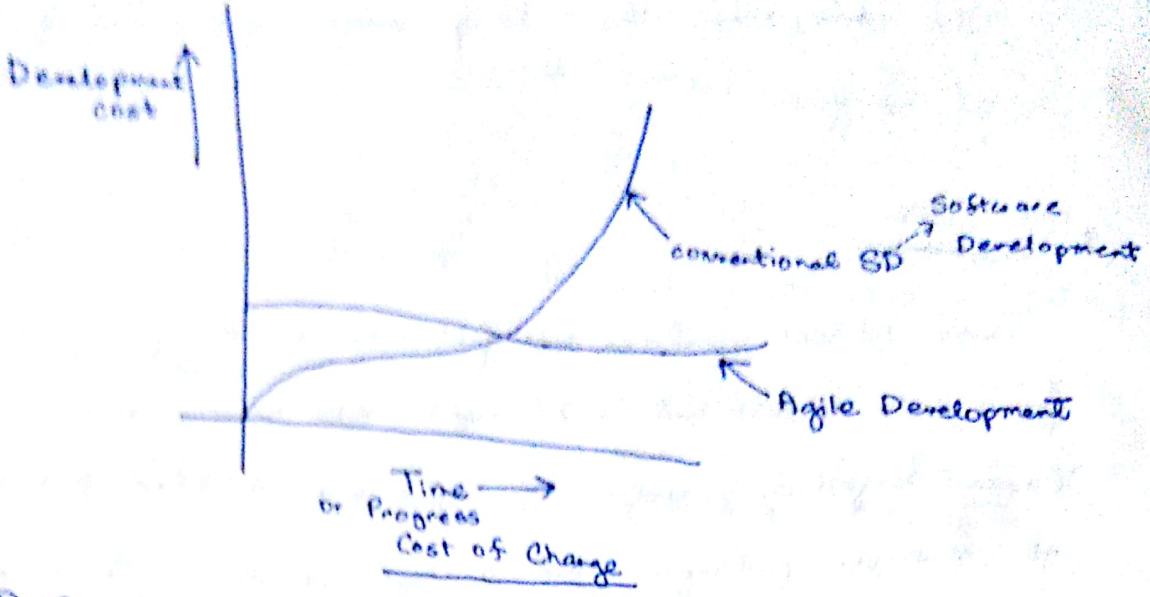
Lightweight code review typically requires less overhead than formal code inspection. It will be typically effective when done properly. Some typical examples are— over the shoulder: One developer looks over the author's shoulder as the latter walks through the code.

Email pass around: Source code management team emails code to reviewers automatically after check in is made.

Pair programming: Two authors develop code together at the same workstation. It is common in extreme programming. ~~Tool assisted co~~

Tool assisted code review: Authors and reviewers use software tools, informal ones such as pastbins and irc or specialised tools designed for peer code

AGILE Development



Agile Software Engineering combines a philosophy and a set of development guidelines. The philosophy advocates adaptive planning, evolutionary development early delivery of the product and continuous improvement, and it encourages rapid and flexible response to change.

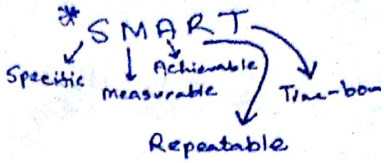
The manifesto for software Agile Development

[Feb 2001, 17 Software developers]

~~To value~~
To Value —

1) Individuals and interactions over processes and tools
2) Working Software over comprehensive documentation.
3) Customer Collaboration over contract negotiation.
4) Responding to Change over following a plan.

Agile Principles (12 Principles):



1) Customer Satisfaction by early and continuous delivery of valuable software.
2) Welcome changing requirements even in late development.
3) Working software is delivered frequently (in weeks rather than months).
4) Close, daily co-operation between business people and developers.
5) Projects are built around motivated individuals who should be trusted.
6) Face-to-face conversation is the best form of ~~communication~~ communication.
7) Working software is the principle measure of progress.
8) Sustainable development, able to maintain a constant pace. — SMART*
9) Continuous attention to technical ~~or~~ excellence and good design enhances agility.
10) Simplicity: The art of maximising the amount of work not done is essential.
11) Best architectures, requirements and designs emerge from self-organising teams.
12) Regularly the team reflects on how to become more effective and ~~at~~ adjusts accordingly.

Types of Agile Software Development:

- ✓ Extreme Programming (XP)
- ✓ ~~Scrum~~ Scrum
- ✓ Adaptive Software Development (ASD)

- **Extreme Programming (XP):**

In Oct 1999, Kent Beck published the book ~~on~~ Extreme Programming.

XP Values:

Beck defines a set of 5 values, which established a ~~set of~~ foundation for all ~~worked~~ works performed as part of XP.

i) Communication
ii) Simplicity
iii) Feedback
iv) ~~Courage~~ Courage
v) Respect

i) Effective Communication:

XP emphasises close, yet informal (verbal) collaboration between customers and developers avoiding voluminous documentation as a communication medium.

ii) Simplicity:

XP restricts developers to design only for immediate needs rather than considering future needs

iii) Feedback:

Feedback is derived from

a) the implemented software itself via unit test results
b) customer
c) other software team members.

iv) Courage / Discipline:

There is often significant pressure to design for future requirement. In order to manage this pressure an Agile XP team must have the discipline (courage) to design for today.

v) Respect:

By having each of these values, the Agile team inculcates respect among its members, between other stakeholders and team members and the software itself.

- Scrum Software Development:

Scrum is a framework for managing work with emphasis on software dev. It is designed for a team of 3-9 dev. who break their work into actions that can be completed within time boxed iterations called sprints (Typically 2 weeks) and track progress and replan 15 min standard meetings called daily scrums.

Roles: There are 3-4 roles in the scrum framework. These are ideally co-located to deliver potentially shippable product implemented in every sprint. Together these 3 roles form the

i) Product Owner: Represents the product's stakeholders and the voice of the customer and is accountable for ensuring that the team delivers value to the business.

ii) Development Team: Responsible for delivering potentially shippable product increments every sprint.

iii) Scrum Master: It facilitates a scrum, and is accountable for removing impediments to the ability of the team to deliver product goals and deliverables. The scrum master is not a traditional team lead or project manager but acts as a buffer between the team and any distracting influences.

Scrum Workload:

Sprint: A sprint (iteration) is the basic unit of development in scrum. A sprint is a time-boxed effort that is restricted to a specific duration. The duration is fixed for each sprint and is normally between 1 week and 1 month, with 2 weeks most common.

Sprint planning: At the beginning of a sprint, the scrum team holds a sprint planning event to: i) Mutually discuss and agree on the scope of work that is intended to be done during that sprint.

ii) Select product backlog items that can be completed in 1 sprint.

iii) Prepare a sprint backlog that includes the work needed to complete the selected product backlog items.

iv) the recommended duration is 4 hrs for a sprint.

v) Once the dev. team has prepared the sprint backlog, they prepare usually a voting to decide which tasks will be delivered with the sprint.

Daily Scrum: Each day within a sprint, a team holds a stand-up.

All members of the dev. team come prepared. The daily scrum starts precisely on time even if some dev. team members.

Should happen at the same time and place everyday. Is limited (time-boxed) with 15 mi. Anyone is welcome, though only dev. team

members should contribute.

Sprint Review and Retrospective: At the end of a sprint the team holds two events — the sprint review and retrospective.

At the end of the sprint review, the team reviews the work that was completed and plans the work that was not completed. Give demo to the stakeholders. The team members and stakeholders collaborate on what to work on next.

Retrospective; The team reflects on the past sprints, identifies and agrees on continuous process improvement actions.