MODULE - II

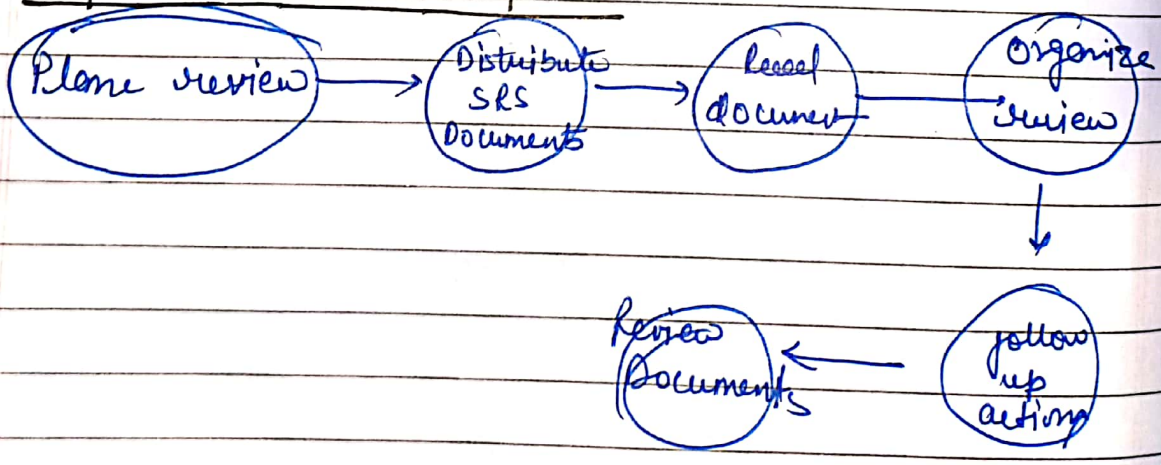
REQUIREMENT DOCUMENTATION

requirements are wr
in a structured formate i.e. known as SRS
SRS should - correctly
→ Correctly define all requirements
→ not describe any design details
→ not impose any additional constraints

Characteristics for a good SRS
As SRS should be
→ Correct
→ Unambiguous
→ Complete
→ Consistence
→ Ranked for important and / or stability
→ Verifiable
→ Modifiable
→ Traceable

Organisation of SRS → only for read

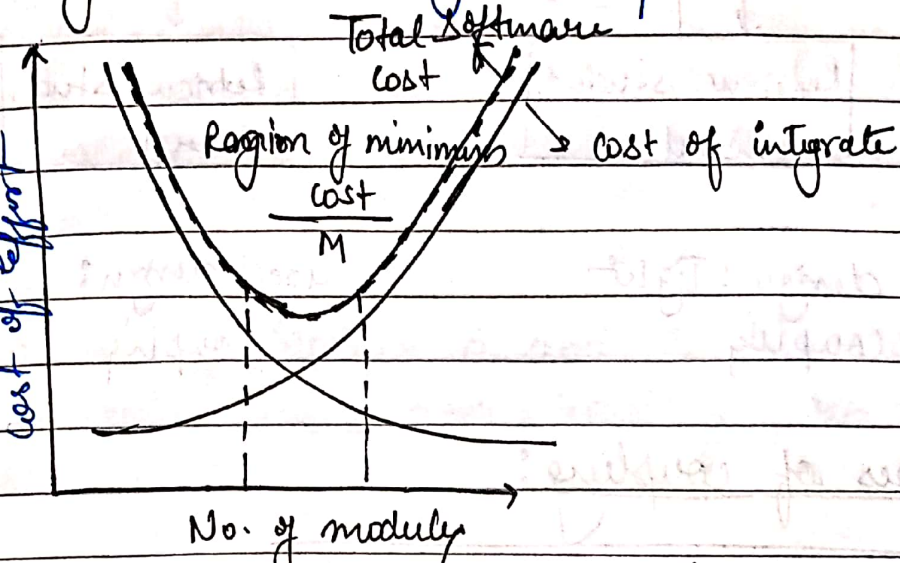
1. Requirement Review Process :



Software design documents

In this phase the focus is on how.

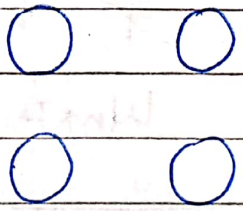
Modularity → Work assignment for an individual program



MODULITY AND SOFTWARE COST

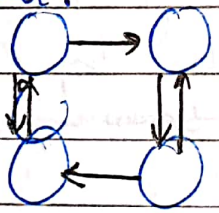
Module Coupling

Coupling is a the measure of the degree of interdependence of between modules

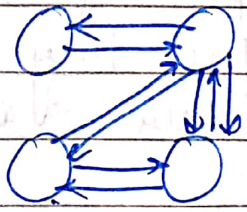


(Uncoupled & No dependencies)

→ loosely coupled: design is better than the highly coupled.

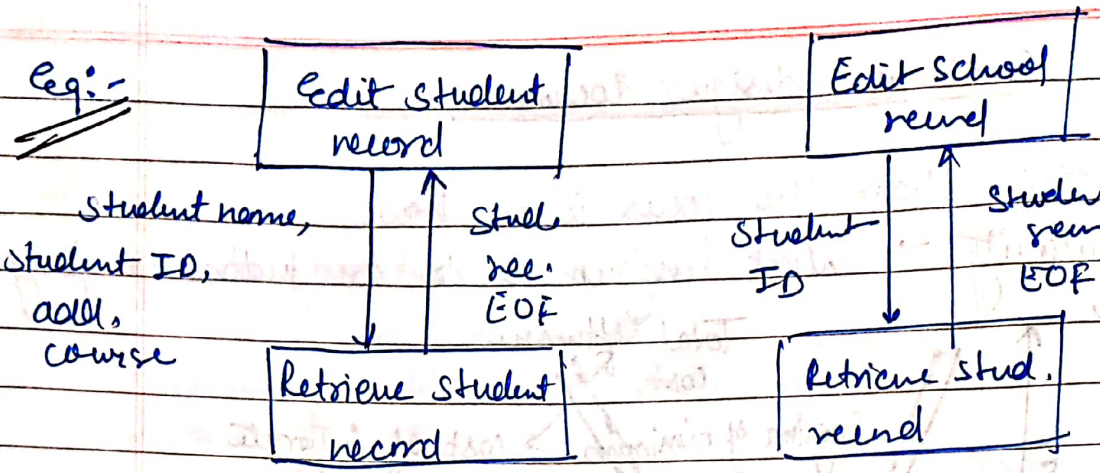


Loosely coupled



highly coupled

Eg:-



Poor design: Tight coupling

Good design: loose coupling

Types of coupling:

Data Coupling	Best
Stamp "	↑
Control "	
External "	
Common "	
Content "	Worst

Data Coupling: The dependency between module A and B is said to be data coupled if their dependency is based on the fact they communicate by only passing of data. Other than communicating through.

Stamp coupling:
Stamp coupling occurs b/w module A and B. When complete data structure is passed from one module to another.

Module A and B are said to be control coupled if they communicate by passing of control information.

Control Coupling: if they communicate by passing of control information.

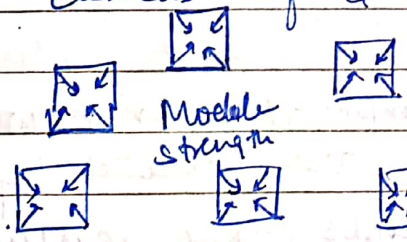
External coupling: A form of coupling in which a module has a dependency of to other module external to the of software being developed ~~of two~~ or to a particular type of hardware

Common coupling: Module A and B have shared data. Global data areas are commonly found in programming language.

Content Coupling: Content coupling occurs when module A changes data of module B or when control is passed from one module to the ~~another~~ middle of another.

MODULE COHESION

Cohesion is a measure of the degree to which the elements of a module are functionally related.



→ types of cohesion →

- Functional cohesion
- Communicational cohesion
- Procedure cohesion
- Temporal cohesion
- Logical cohesion
- Coincidental cohesion

Best (High)

↑

Worst (low)

Functional Cohesion:
A and B are single functional task. This is very good reason for them to be contained in the same procedure.

Sequential Cohesion:
Module A outputs some data which form the input to B. This is the reason for them to be contained in the same procedure.

Procedural Communicational Cohesion:
A and B both operate on same input data or contribute towards same output data.

Procedural Cohesion:
It occurs in modules whose instructions although accomplish different tasks yet have been combined because there is a specific order in which the tasks are to be completed.

Temporal Cohesion:
Module exhibits temporal cohesion when it contains tasks that are related by the fact that all tasks must be executed in the same time-span.

Logical Cohesion:
It occurs in modules that contain instructions that appear to be related because they fall into the same logical class of functions.

Coincidental cohesion:
It exists in modules that contain instructions that if have little or no relationship to one another.

For a good design strategies coupling should be low and cohesion must be high.

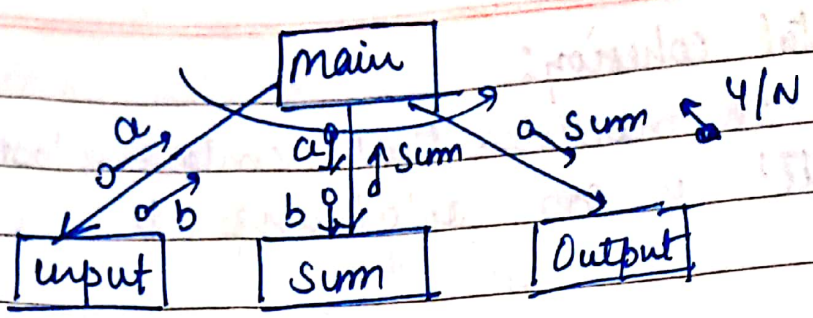
Strategy of design
A good system design strategy is to organise the program modules in such a way that are easy to develop and latter to change.

1. Bottom-up design
2. Top-down "
3. Hybrid "

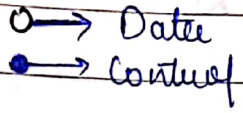
Functional oriented design

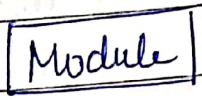
Functional oriented design is an approach to software design where the design is decomposed into a set of interating units, where each unit has a clearly defined function. Thus, system is designed from a functional viewpoint.

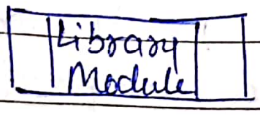
Structure chart:
→ Structure chart partiate a system into ~~block bo~~ black boxes.
→ A black boxes means that functionality is known to user without the knowledge of internal design.
Example:- Create a structure chart for calculation of addition of two no.s.

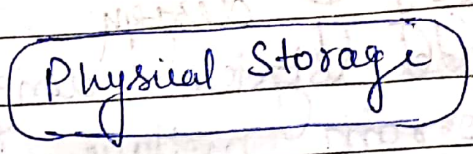


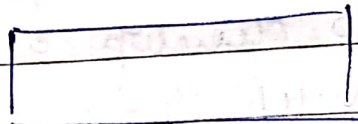
NOTATION :



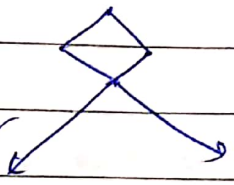


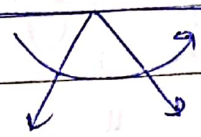






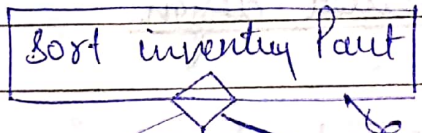
diamond
Symbol for
Conditional
call of module



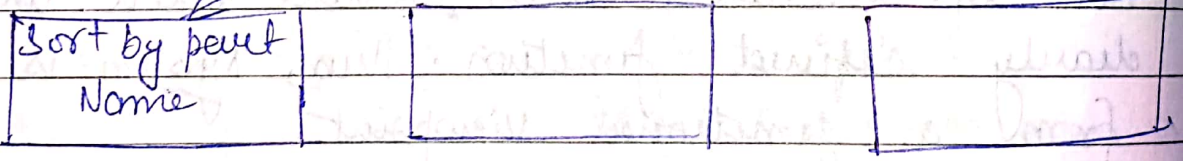


Repetitive Call
of module

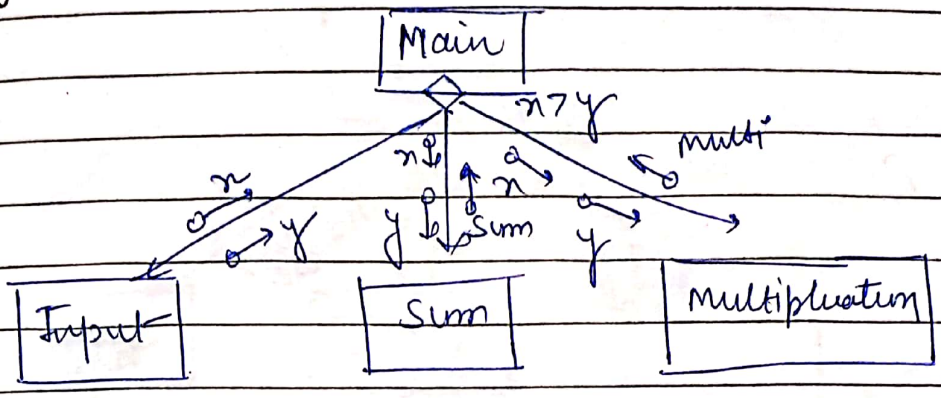
level 0



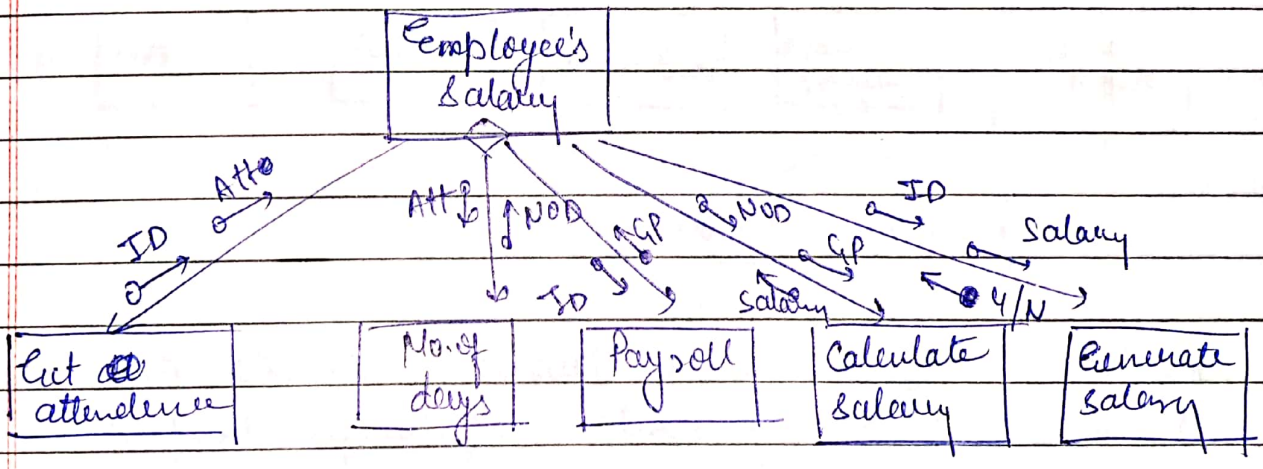
level 1



Ques:- Consider a program for two ~~values~~ variable extends if x>y then perform addition and multiplication.

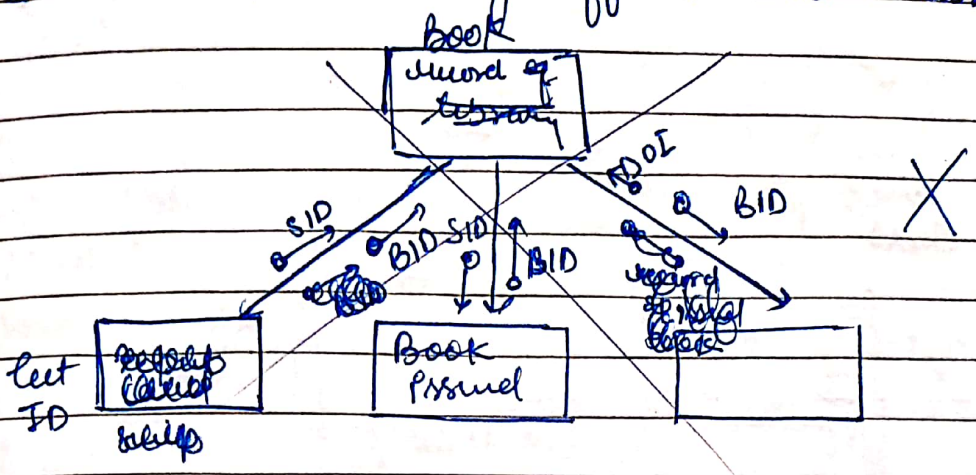


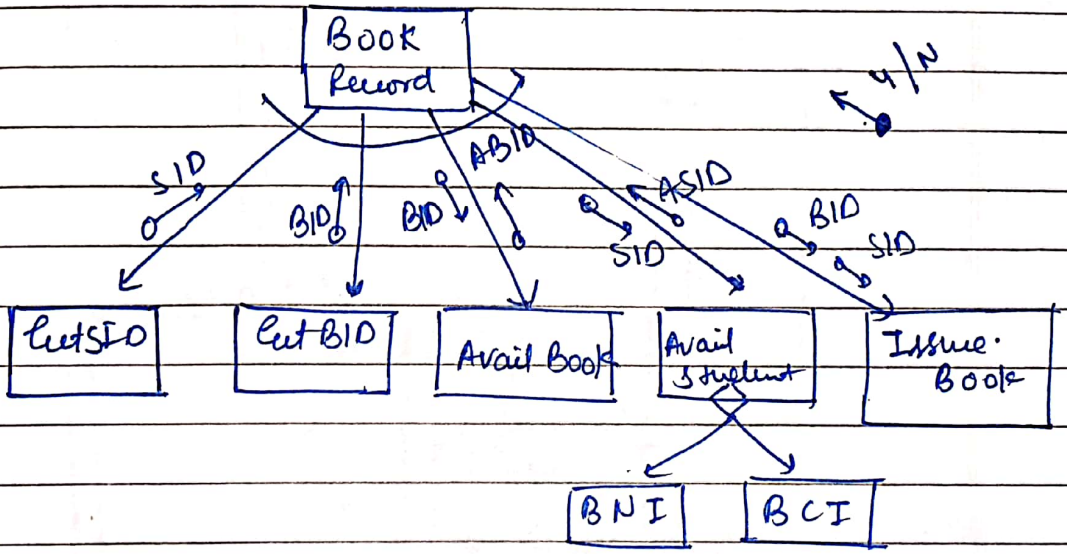
Ques:- Create structure chart to calculate salary of employees on the basis of their attendence in the Company and great pay.
Also print summary of salary of employee.



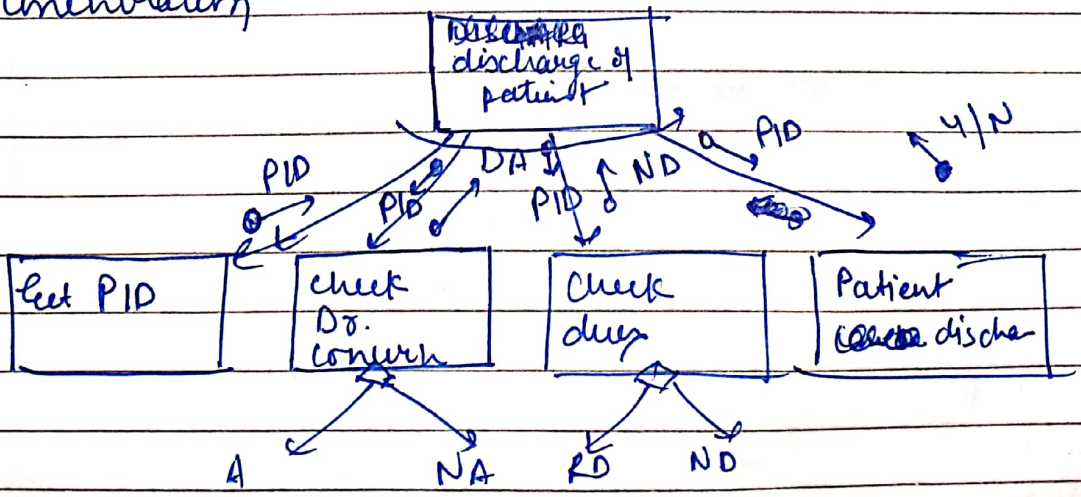
Ques:- Structure chart for maintaining the record of ~~student~~ books issued by diff students in library

Ans:-

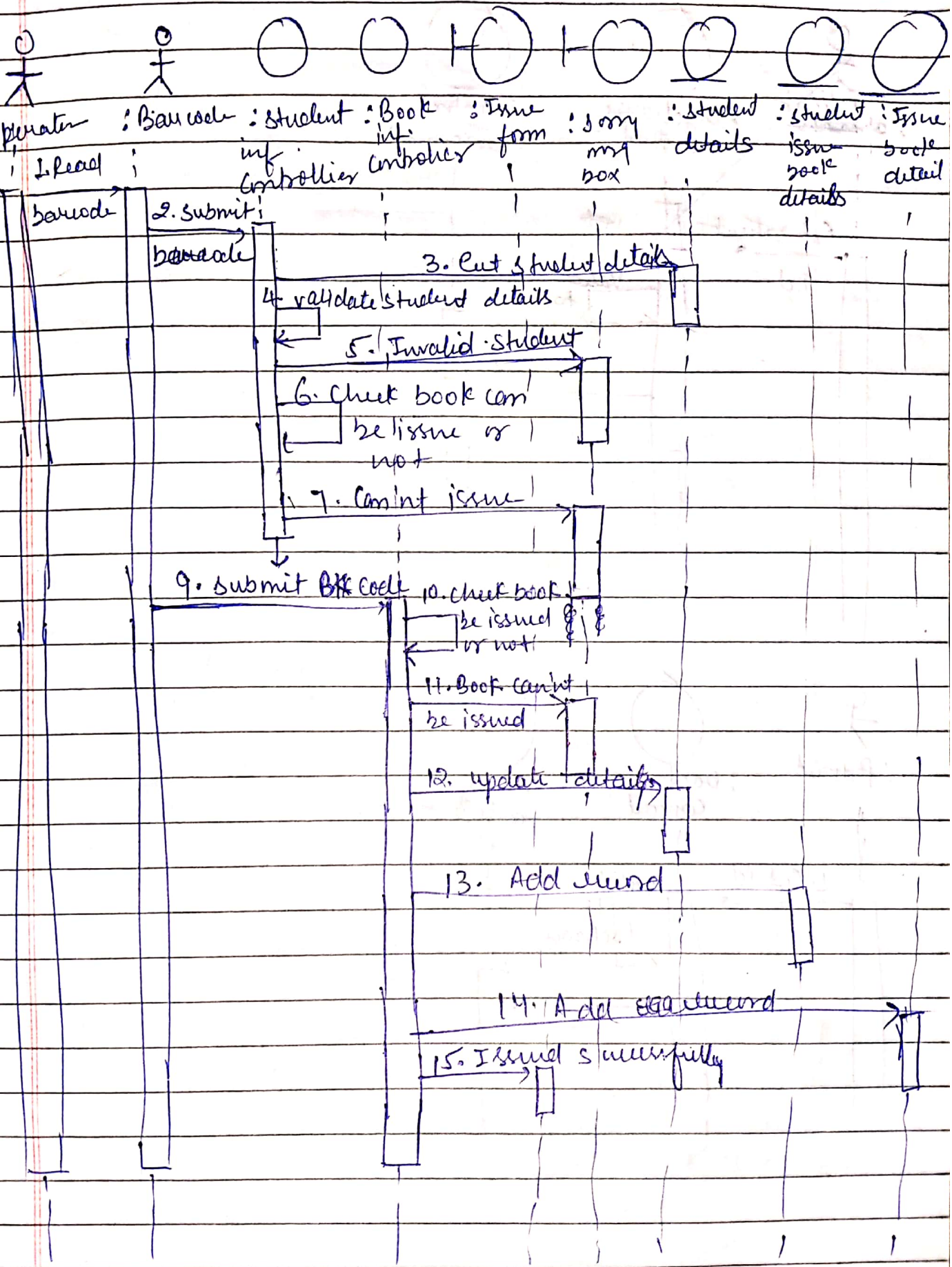




Ques:- Create structure chart to discharge a patient in hospital ~~after~~ after clearing all dues and Dr. consultation



Ques:- Create sequence diagram issue of book library management system.



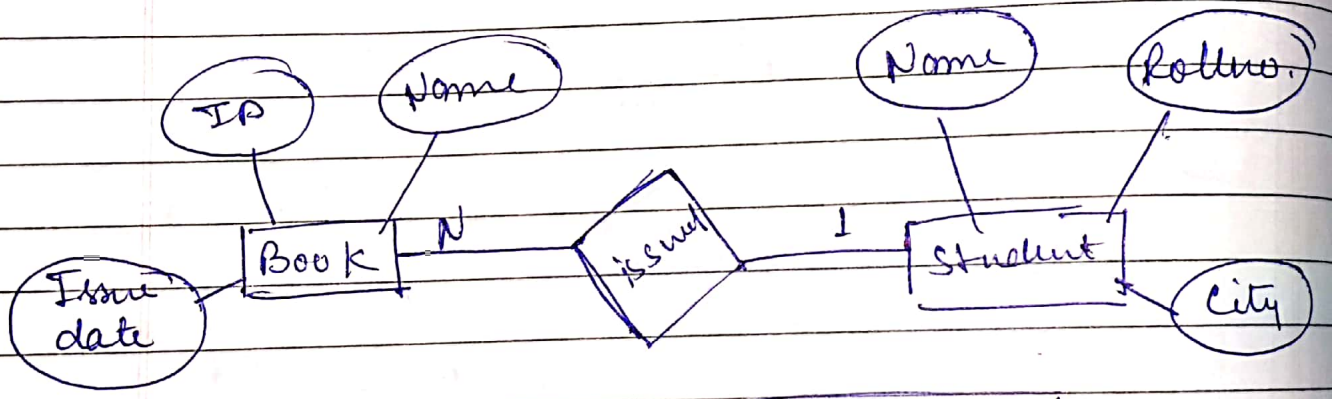
Ques:- Consider the following scenario and create ER- diagram and and class diagram

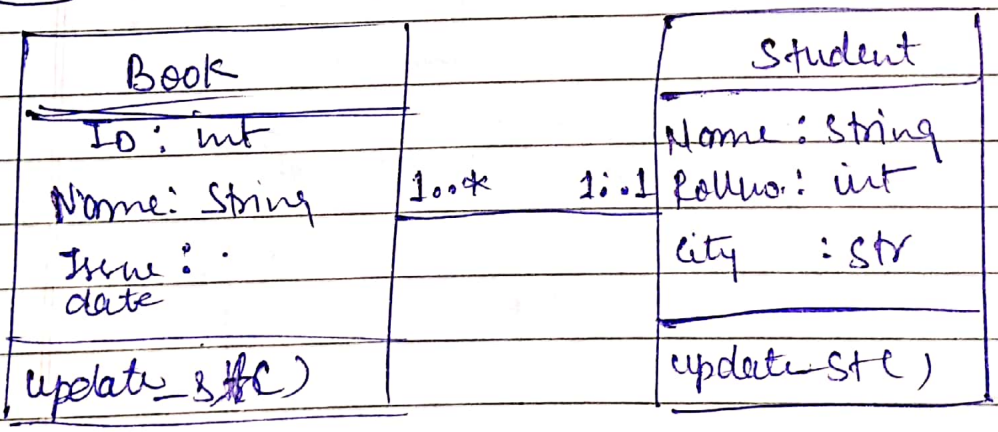
(I) - Suppose in library management system we can store information of books like: Book ID, Book name, and issue date

(II) We can also store information of different Student who are issuing the books from library management system like: Student roll no., name, and city.

(III) One student can issue multiple books.

(IV) We also use a function to update inform. of diff. students and we are using of function to add new book in library management system





Ques:- Create ER diag and class ~~dig~~ diag by considering the following schema.

(I) Suppose in a company we can store the information of different employee like emp_ID, name and balance

(II) Company also ~~info~~ stores information of diff depart-ment like depart no., name and location.

(III) Multiple employee can work in one department but one employee can not work in more than one department.

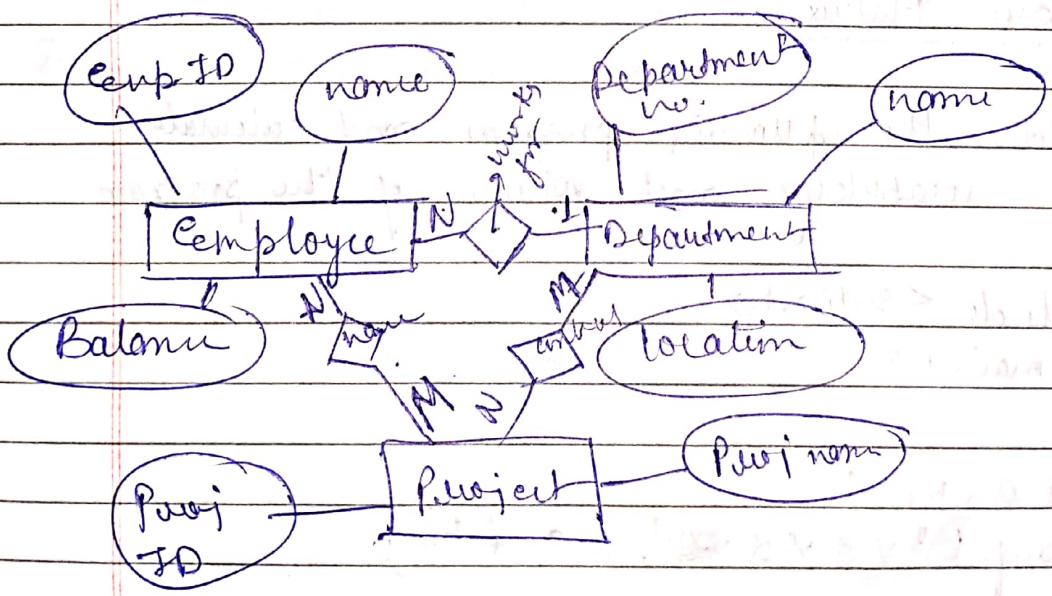
(IV) Company also keep track of diff projects information like proj_ID, proj name.

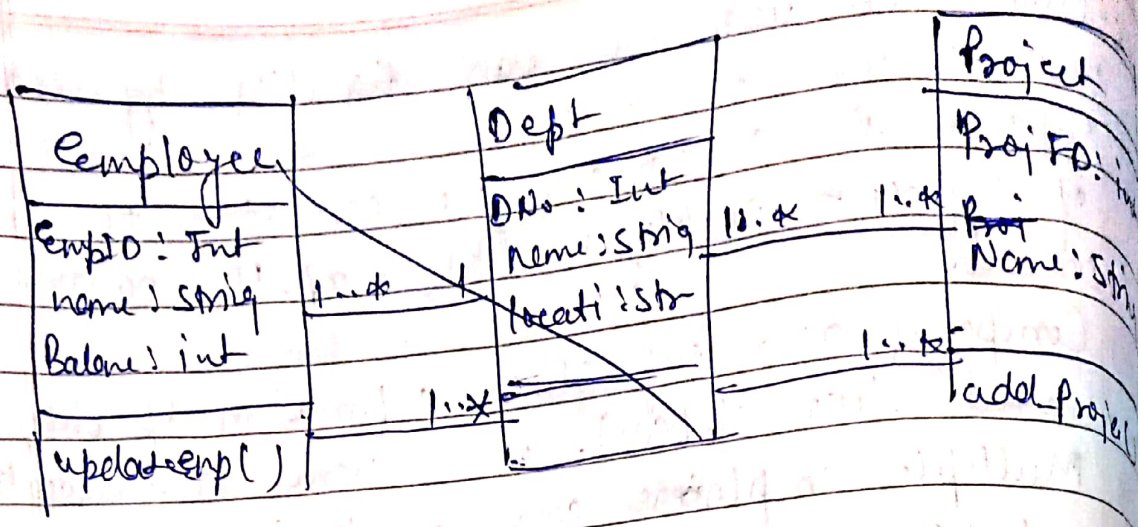
(V) One department can have multiple projects and multiple department can also work in one project

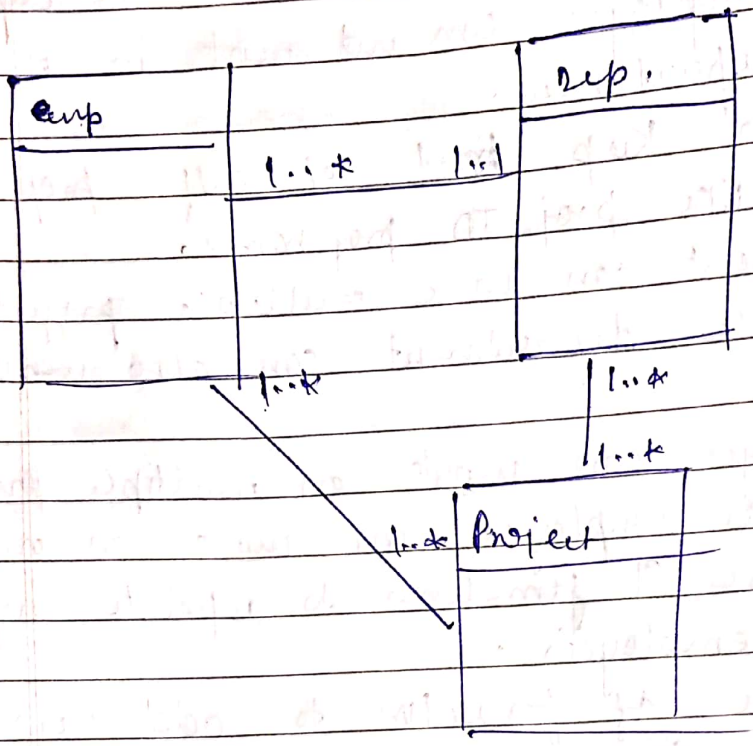
(VI) One employee can work on multiple projects and multiple employee can work on one proj

(VII) Company use of function to update info. of diff employees.

(VIII) Company use of function to add new projects in the company and departments







Software Matrix

Ques:- Consider the following program and calculate length vocabulary and volume of the program

```
#include <stdio.h>
void main()
{
    int a, b, c;
    scanf("%d %d", &a, &b);
```

```
c = a+b;
printf("%d", c);
}
```

Rules for counting:

(I) Comments are not considered.
(II) Function declarations are not considered
(III) All the variables and constants are considered operands
(IV) Function calls are considered as operator
(V) Looping statement like Do, while, for are considered as operator.
(VI) If, else also considered as operators.
(VII) All brackets in pair ((), {}, []) considered as operators.
(VIII) All reserved words like return, continue, break, are considered as operators.
(IX) All the (,) and the terminators (;) considered as operators
(X) All the unary and binary occurance of (+, -, *, /) are operator.
(XI) In the array variable array name and index considered as operand and bracket as operator.
(XII) All the # directive are ignored.

Operator	Occurrence
( )	3
{ }	1
%d	3
;	4
&	3
,	5
+	1
=	1
#	2
void	1
main	1
int	1
Scanf	1
printf	1

Operand	Occur
a	3
b	3
c	3

Ques:- Consider the following program and calculate a no. of operators and operands exist in that program:

```
# include <stdio.h>
int main()
{
    int a[10], i;
    for (i=1; i<10; i++)
    {
        Printf("%d", a[i]);
    }
    return 0;
}
```

operator	occur	operand	
int	2	a	2
main	1	i	5
( )	3	10	2
{ }	2	1	1
[ ]	2		
~~=~~	2		
,	2		
;	3		
return	1		
" "	1		
printf	1		
for	1		
+	2		
=	1		
<	1		
0	1		

Token Count

The size of vocabulary of a prog., which consists of the no. of unique tokens used to build a program is defined as!

$$n = n_1 + n_2$$

where n = vocabulary of a prog.

n_1 = no. of unique operator

n_2 = no. of ~~error~~ unique operand

The length of the program in the terms of the total no. of tokens used is!

$$N = N_1 + N_2$$

where N : program length

N_1 : total ~~occurrences~~ occurrences of operators

N_2 : total occurrences of operands

Volume:

$$V = N * \log_2 n$$

The unit of measurement of volume is the common unit for size "bits". It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

Ques: Consider the following prog. and find out the value of Calculate vocabulary, length & volume of the program using Halstead software science metrics method

```
int sort (int x[], int n)
{
    int i, j, im, save;
    if (n < 2)
        return 1;
    for (i = 1; i <= n - 1; i++)
    {
        im = i;
        for (j = i + 1; j <= n; j++)
            if (x[j] < x[im])
                im = j;
        save = x[i];
        x[i] = x[im];
        x[im] = save;
    }
    return 0;
}
```

Operator Symbol	occ	operand	occ
int	4	x	7
,	4	n	3
sort	1	i	8
()	5	j	7
*	1	im	3
[]	7	save	3
,	4	1	2
if	1	2	2
if()	1	1	3
<	2	0	1
=	6	sort	1
;	11		
if	2		
:	11		
for	2		
=	1		
<=	2		

++	2
return	2
24	3

$n_1 = 14$ $N_1 = 53$ $n_2 = 10$ $N_2 = 38$

$$n = 14 + 10 = 24$$

$$v = 53 + 38 + \log_2 24$$

$$= 47 \text{ bits}$$

$$N = 53 + 38$$

$$= 91$$

17/oct

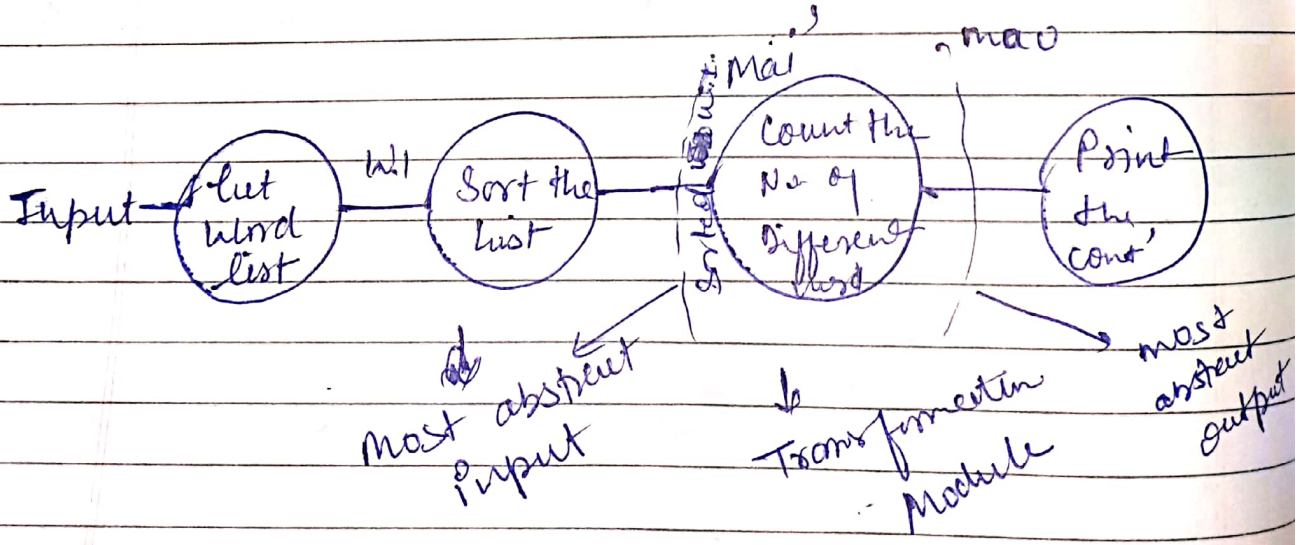
Structured design methodology

→ SDM views software as a transforming function that converts given inputs to desired outputs
→ The focus of SD is the transforming function
→ Uses functional abstraction and functional decomposition.
→ The objective is to achieve low coupling high cohesion.

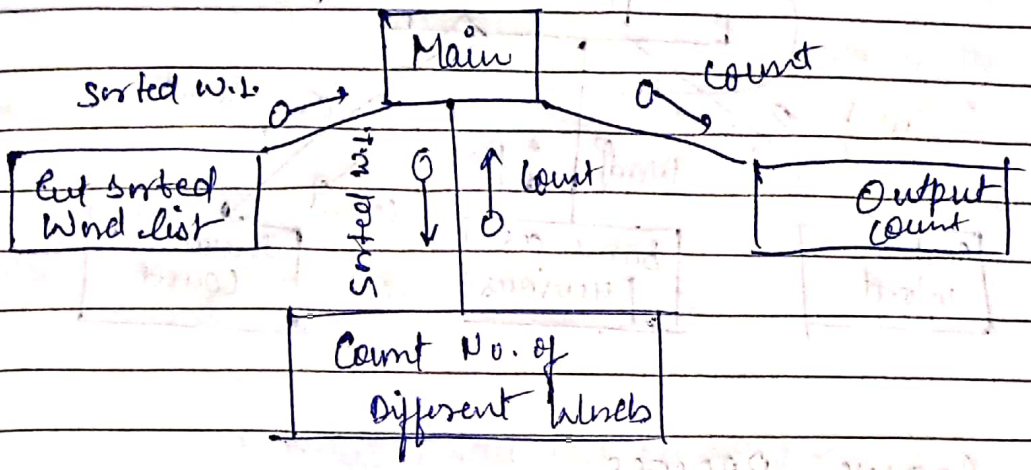
Ques:- Consider a problem for determining the no. of unique word in a file create structure design methodology by showing all the steps

There are four major steps for structure chart methodology -

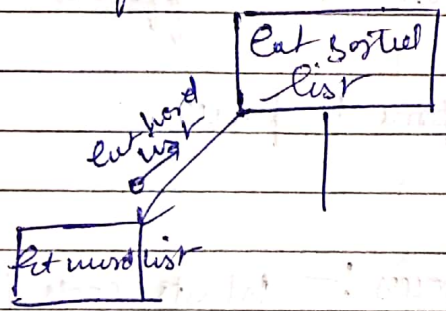
1. Draw a DFD of the system.
2. Identify most abstract input and most abstract output.
3. First level factoring
4. Factoring of input, output and transform branches.

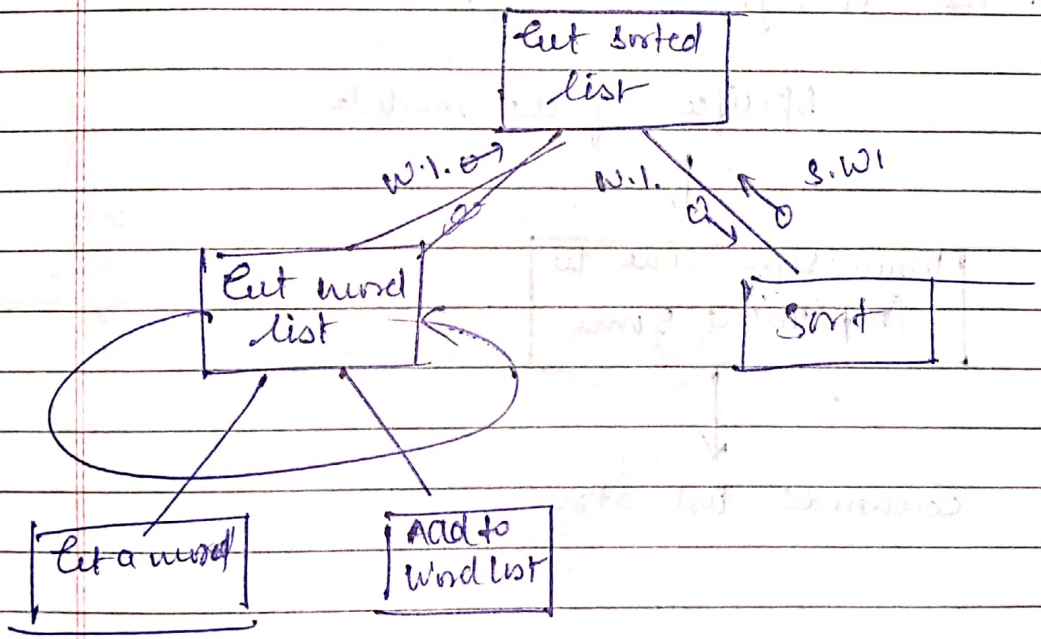


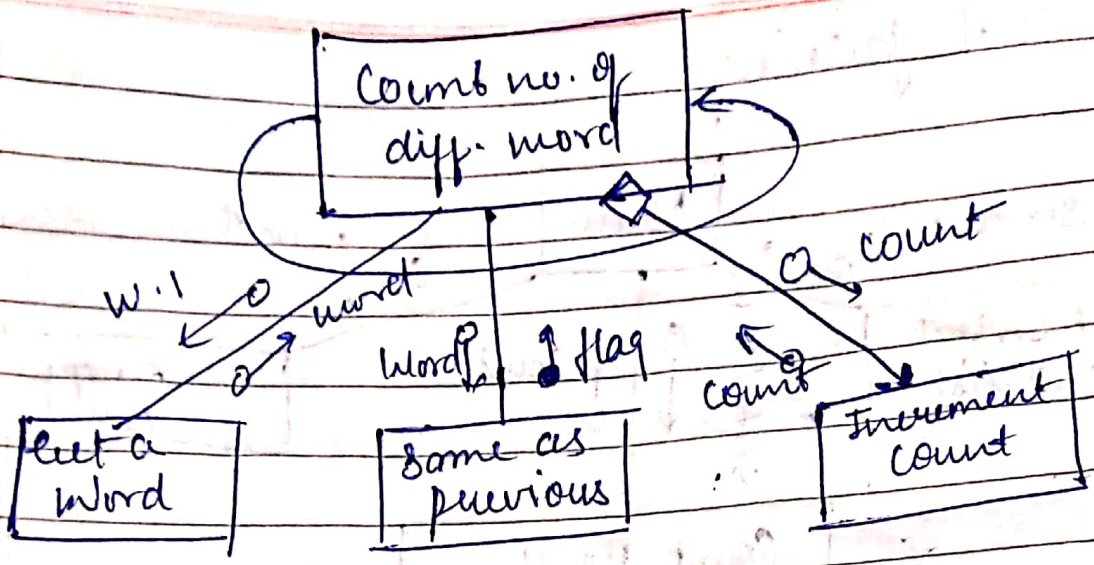
(3) First level factoring



(4):- factoring of input, output and transform branches







CODING PROCESS

Coding process is of three different type
- Implemental process
- Test driven development process
- Pair programming

(1) Implemental coding process :- Write code for module, unit test it, fix the bugs.

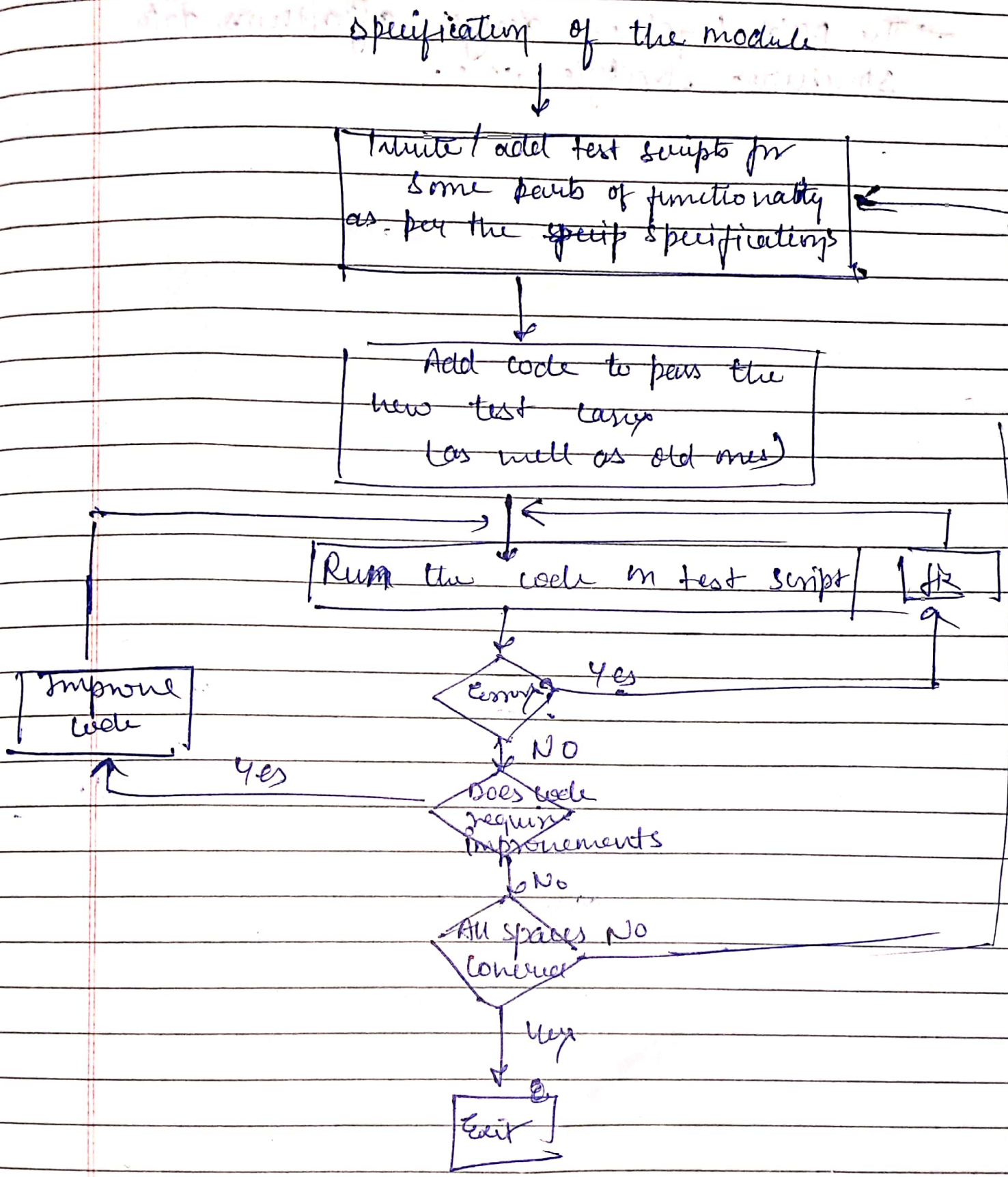
Specific of the module

↓

Write some code to Implement some

↓

Comprehensive test strategy



③ Pair programming

~~A coding process~~.
Code is written by pair of programmers.
~~It~~ rather than individuals.

→ The pair together design algorithms, data structures, strategies, etc.