4. Design

Introduction :-

- The design activity begins when the requirements document for the software to be developed is available.
- During design phase we further refine the architecture.
- Design focuses on module view.
- The design process for software systems often has two levels.
- At the first level the focus is on deciding which modules are needed for the system, also called by module design or the high-level design.
- In the second design level, the internal design of the modules, are decided. This design also called as detailed design or logic design.
- A design methodology is a systematic approach to creating

4. Design

Introduction :-

- The design activity begins when the requirements document for the software to be developed is available.
- During design phase we further refine the architecture.
- Design focuses on module view.
- The design process for software systems often has two levels.
- At the first level the focus is on deciding which modules are needed for the system, also called by module design or the high-level design.
- In the second design level, the internal design of the modules, is are decided. This design also called as detailed design or logic design.
- A design methodology is a systematic approach to creating

Roll No. _____

a design by applying set of techniques & guidelines.

4.1. Design concepts :-

- The design of a system is correct if a system built precisely according to the design satisfies the requirements of that system.
- The goal of design process is not simply to produce a design for the system. Instead, the goal is to find the best possible design.
- A system is considered modular if it consists of discrete modules so that each module can be implemented seperately.
- A software system can not be made modular by simply chopping it into a set of modules.
- For modularity, each module needs to support a well-defined

abstraction & have a clear interface through which it can interact with other modules.
- Coupling & cohesion are two modularization criteria that are often used together.

* Coupling :-

- Two modules ~~can~~ are considered independent if one can function completely without the presence of the other.
- The fewer & simpler the connections between modules, the easier it is to understand one without understanding the other.
- The notion of coupling attempts to capture this concept of "how strongly" different modules are interconnected.
- "Highly coupled" modules are joined by strong interconnections, while "loosely coupled" modules

Roll No. _____

have weak interconnections.
- Coupling between modules is largely decided during system design & can not be reduced during implementation.
- Coupling is reduced if only the defined entry interface of a module is used by other modules.
- For example, passing information to & from a module exclusively through parameters.
- Complexity of the interface is another factor affecting coupling.
- However, often more passing parameters than minimum parameters are passed.
- For example, if a field of a record is needed by a procedure, often the entire record is passed, rather than just passing that field of record.
- By passing the record we are increasing the coupling unnecessarily.

- The type of information flow along the interfaces is the third major factor affecting coupling.
- There are two kinds of information that can flow along an interface: data or control.
- Interfaces with only data communication result in the lowest degree of coupling.

Table 1. Factors affecting coupling.

	Interface complexity	Types of connection	Type of communication
Low	simple obvious	To module by name	Data
High	complecated obscure	To internal elements	Hybrid

(8) Roll No. _____

- The manifestation of coupling in object-oriented (OO) systems.
- In object oriented systems, three different types of coupling exist between modules.
  1. Interaction coupling
  2. Component coupling
  3. Inheritance coupling
- Interaction coupling occurs due to methods of a class invoking methods of others classes.
- Coupling is higher if the amount of data being passed is increased.
- The least coupling situation therefore is when communication is with parameters only, with only necessary variables being passed & these parameters only pass data.

- Component coupling refers to the interaction between two classes where a class has variables of the other class.
- when C is component coupled with C1, it has the potential of being component coupled with all subclasses of C1.
- Inheritance coupling is due to the inheritance relationship between classes.
- Two classes are considered inheritance coupled if one class is a direct or indirect subclass of the other.
- If a class A is coupled with another class B & if B is a hierarchy with B1 & B2 as two subclasses, then if a method m() is forced out of B1 & B2 and put in the superclass B, the coupling drops as A is now only coupled with B

Roll No. _____

Roll No. _____

whereas earlier it was coupled with both B1 & B2

Cohesion:-

- To strenghten the bond between elements of the same module.
- cohesion is the concept that tries to capture intramodule
- Cohesion of a module represents how tightly bound the internal elements of the module are to one another.
- The greater the cohesion of each module in the system, the lower the coupling between modules
- There are several levels of cohesion:-

1. Coincidental
2. Logical
3. Temporal
4. Procedural
5. communicational
6. sequential
7. functional.

Roll No. 5

- Coincidental is the lowest level & functional is the highest level.
- Coincidental cohesion occurs when there is no meaningful relationship among the elements of a module.
- A module has logical cohesion if there is some logical relationship between the elements of a module.
- Example :-
  module that performs all the inputs or all the outputs.
- In general, logically cohesive modules should be avoided, if possible.
- Temporal cohesion is the same as logical cohesion, except that the elements are also related in time and are executed together.
- Modules that perform activities like "initialization", "cleanup" & "termination" are usually temporally bound.

Roll No. _____

Roll No. _____

- ~~8 productivity~~
- A procedurally cohesive module contains elements that belong to a common procedural unit.
- For example, a loop or a sequence of decision statements in a module may be combined to form a seperate module.
- A modules with communicational cohesion has elements that are related by a reference to the same input or output data.
- An example of this could be a module to "print & punch record".
- communicationally cohesive modules may perform more than one function.
- communicational cohesion is sufficiently high as to be generally acceptable if alternative structure with higher cohesion cannot be easily identified.

- When the elements are together in a module because the output of one forms the input of another, we get sequential cohesion.
- If we have sequence of elements in which the output of one forms the input to another,
- Functional cohesion is the strongest cohesion.
- In a functionally ~~cohesion~~ bound module, all the elements of the module are related to performing a single function.
- Functions like "compute square root" and "sort the array" are clear examples of functionally cohesive modules.
- Modules with functionally cohesion can always be described by a simple sentence.

Roll No. _____

(2)

Roll No. _____

- cohesion in object-oriented systems has three aspects
  1. Method cohesion
  2. class cohesion
  3. Inheritance cohesion

- Method cohesion is the same as cohesion in functional modules.
- Class cohesion focuses on why different attributes of methods are together in this class.
- There are multiple concepts encapsulated within a class, the cohesion of the class is not as high as it could be, if a designer should try to change the design to have each class encapsulate a single concept.

- For cohesion, it is best to represent them as two seperate classes with relationship among them.
- This improvement in cohesion results in an increased coupling
- However, for modifiability and understandability, it is better if each class encapsulates a single concepts.
- Inheritance cohesion focuses on the reason why classes are together in a hierarchy.
- cohesion is considered high if the hierarchy supports generalization specialization of some concepts.

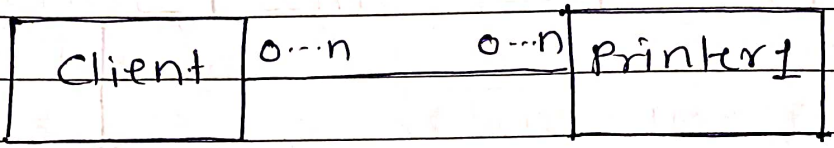
* The open-closed principle:-
- This is design concept.
- The basic goal is to promote building of systems that are easily modifiable.
- The basic principle stated by Bertrand Meyer, is "software entities should be open for

Roll No. _____

extension, but closed for modification.

- A module being "open for extension" means that its behavior can be extended to accommodate new demands placed on this module due to changes in requirement / system functionality
- The module being "closed for modification" means that the existing source code of the module is not changed when making enhancements.
- This principle restricts the changes to modules to extension only, that is, it allow addition of code but disallows changing of existing code.
- programmers typically prefer writing new code rather than modifying the old code
- This principle can be satisfied in object oriented designs by properly using inheritance & polymorphism.

- The class diagram for this will be shown in figure.



4.1

fig. Example without using subtyping.

- Client class is not closed against change.

Fig.2 - In this design, instead of directly implementing the printer1 class, we create an abstract class printer that defines the interface of a Printer & specifies all the methods a printer object should support.

- The client does not need to be aware of this subtype as it interacts with objects of type Printer.
- The class diagram shown in figure 4.2

Roll No. _____

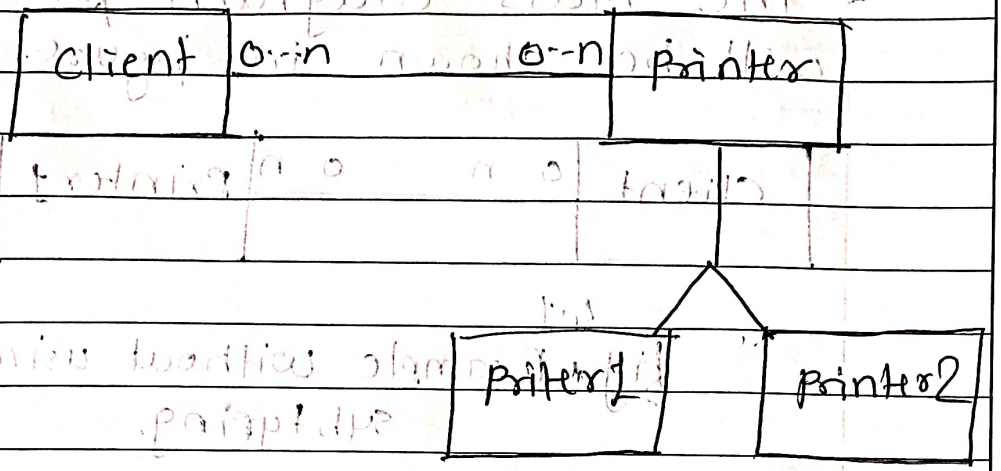


fig 42.

Example using subtyping

Roll No. 9

6.

4.2 Function-oriented Design:-

- The aim of design methodologies is not to reduce the process of design to a sequence of mechanical steps but to provide guidlines to aid the designer during the design process.
- The methodology uses the structure chart notation for creating the design.

*Structure chart:-

- The structure of program is made up of the modules of that program together with the interconnections between the modules.
- The structure chart of a program is a graphic representation of its structure.
- In structure chart a module is represented by a box with

Roll No. _____

module name written in the box.
- An arrow from module 'A' to module 'B' represents that module 'A' invokes module 'B'. B is called the subordinate of 'A' & A is called the superordinate of B.
- The arrow is labled by the parameters received by B as input & the parameters returned by B as output, with the direction of flow of the input & output parameters represented by small arrows.
- The parameters can be shown to be data (unfilled circle at the tail of the lable) or control (filled circle at the tail)

```
main()
{
  int sum, n, N, a[MAX];
  readnums(a, &N); sort(a, N);
  scanf(&n);
  sum = add_n(a, n); Printf(sum);
}
```

Roll No. 10

```
readnums (a, N)
int a[], N;
{
   :
   if (a[i] > a[t]) switch (a[i], a[t]);
   :
/* Add the 1st n numbers of
      a */
add_n(a, n)
int a[], n;
{
   :
}
```

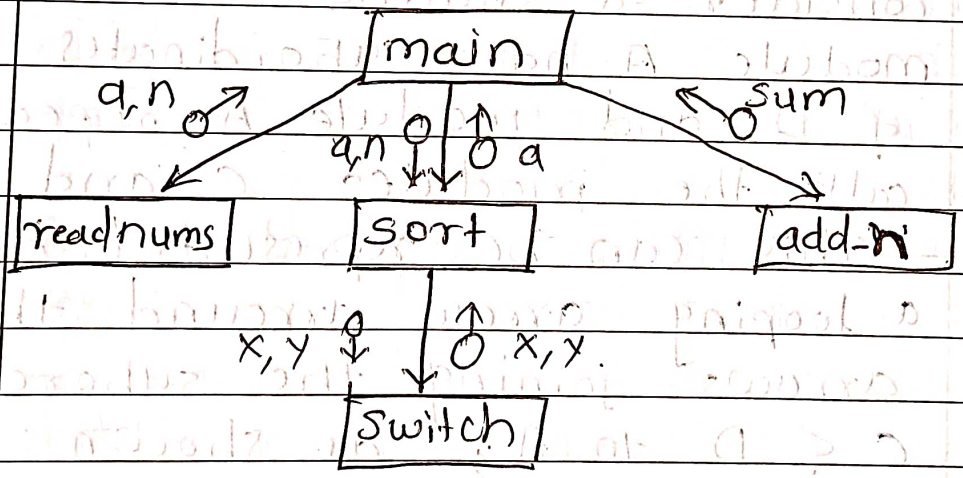


fig. 4.3 The structure chart of the sort program.

Roll No. _____

Roll No. _____

- In general, procedural information is not represented in a structure chart, and the focus is on the hierarchy of modules.
- There are situations where the designer may wish to communicate certain procedural information explicitly, like major loops & decisions. Such information can also be represented in a structure chart.
- for example:-

consider a situation where module A has subordinates B, C & D, and module A repeatedly calls the modules C and D.

- This can be represented by a looping arrow around the arrows, joining the subordinates C & D to A; as shown in figure 4.4.
- All subordinates modules activated with in a common loop are enclosed in the same looping arrow.

Roll No. 13

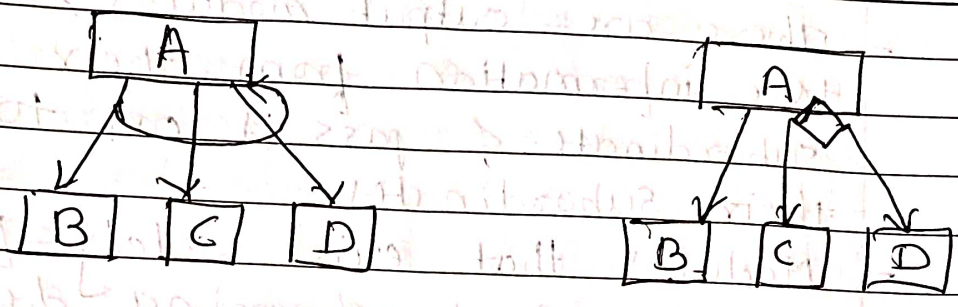


Fig. 4.4. Iteration & decision representation.

- Major decisions can be represented similarly.
- eg.
  If the invocation of modules C & D in module A depends on the output of some decision, that is represented by a small diamond in the box for A, with the arrows joining C & D coming out of this ~~represented by a small~~ diamond as shown in figure 4.4.
- There are some modules that obtain information from their subordinates & then pass it to their subordinates. These modules are called Input modules.

Roll No. _____

- There are output modules, which take information from their subordinate & pass it on to their subordinates.
- Modules that exist solely for the sake of transforming data into some other forms. such modules are called as transform modules.
- Managing the flow of data to & from different subordinates. Such modules are called as coordinates modules.
- The aim is to design system so that programs implementing the design would have a hierarchical structure.

* Structure design methodology :-
- The basic principle behind the structured design methodology is problem partitioning.
- Structure design methodology partitions the system at the very top level into various

subsystems, one for managing each major input, one for managing each major output & one for each major transformation

- The modules dealing with inputs have to deal with issues of screens, reading data, formats, errors, exceptions, completeness of information, structure of the information, etc.
- The modules deadlines with output have to prepare the output in presentation formats, make charts, produce reports etc.

* There are four major steps in the methodology:-

1. Restate the problem as a data flow diagram.
2. Identify the input & output data elements.
3. first-level factoring
4. factoring of input, output, and transform branches.

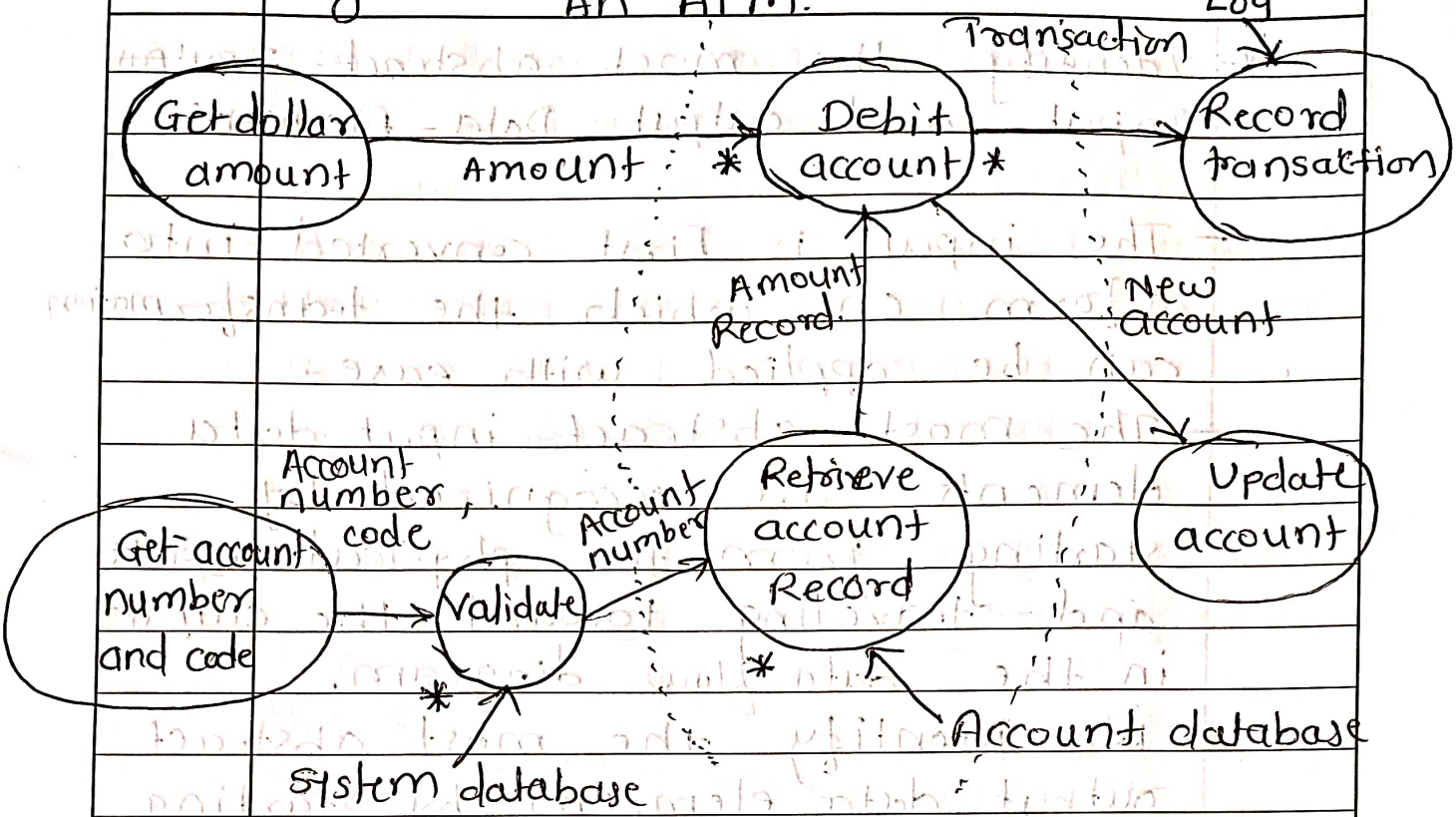
Roll No. _____

* Restate the problem as a Data flow Diagram:-

- There is a fundamental difference between the DFD's drawn during requirements analysis & those drawn during structured design.
- In requirement analysis, a DFD is drawn to model the problem domain.
- The analyst has little control over the problem, & hence his task is to extract all the information from the probem and then represent it as a DFD.
- During design activity, we are no longer modeling the problem domain, but are dealing with the solution domain.
- In this modeling, the major transforms or functions in the software are decided.
- A DFD of an ATM is shown in figure. 4.5.

Roll No. 13

fig. Data flow Diagram of an ATM.



- There are two major streams of input data in this diagram.
- The 1st is the account number & code, & the second is the amount to be debited.
- Notice the used of * at different places in the DFD.

Roll No. _____

* Identify the most abstract ~~Input~~ Input and output Data Elements.

- The input is first converted into a form on which the transformation can be applied with ease.
- The most abstract input data elements are recognized by starting from the physical inputs and traveling toward the outputs in the data flow diagram.
- We identify the most abstract output data elements by starting from the outputs in the data flow diagram & traveling toward the inputs.
- There will usually be some transforms left between the most abstract input & output data items. These central transforms perform the basic transformation for the system.
- Consider now the data flow diagram of the automated teller shown in ~~di~~ figure 4.5.

Roll No. 14

- The Two most abstract inputs are the dollar amount & the validated account number.
- The validated account number is the most abstract input, rather than the account number read in, as it is still the input.
- The abstract inputs & outputs are marked in the data flow diagram.

* First-Level Factoring :-

- We first specify a main module whose purpose is to invoke the subordinates.
- The main module is a co-ordinate module.
- A subordinate module that is an output module that accepts data from the main module is specified.
- central transform, a module

Roll No. _____

subordinate to the main one is specified.

- These modules will be transform modules, whose purpose is to accept data from the main module and then returns the appropriate data back to the main module.
- Let us examine the DFD of ATM.
- Drawing a module for each of these, we get the structure chart shown in figure 4.6.

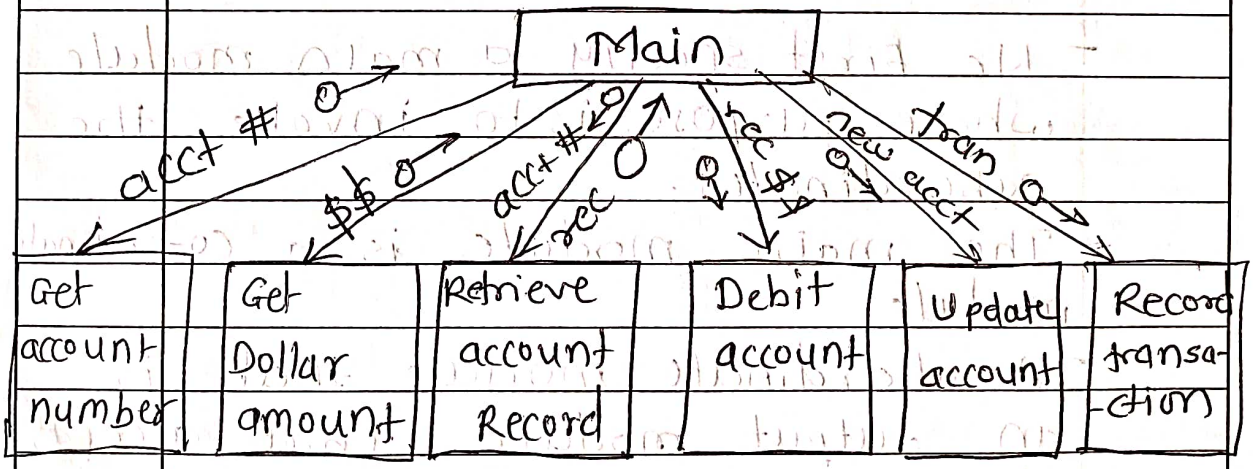


fig 4.6 First-level factoring for ATM.

Roll No. 15

* Factoring the Input, output, & Transform Branches.

- The first-level factoring results in a very high-level structure, where each subordinate module has a lot of processing to do.
- To simplify these modules, they must be factored into subordinate modules that will distribute the work of a module.
- The process performed for the first-level factoring is repeated here with the new central transform, with the input ~~method~~ module being considered the main module.
- The new input modules now created can then be factored again, until the physical inputs are reached.
- The factoring of the output modules is symmetrical to the factoring of the input modules.

Roll No. _____

- During the factoring of output modules, there will usually be no input modules.
- One way to factor a transform module is to treat it as a problem in its own right and start with a data flow diagram for it.

* An example :-

- Consider the problem of determining the number of different words in an input file.
- The data flow diagram for this problem is shown in figure 4.7

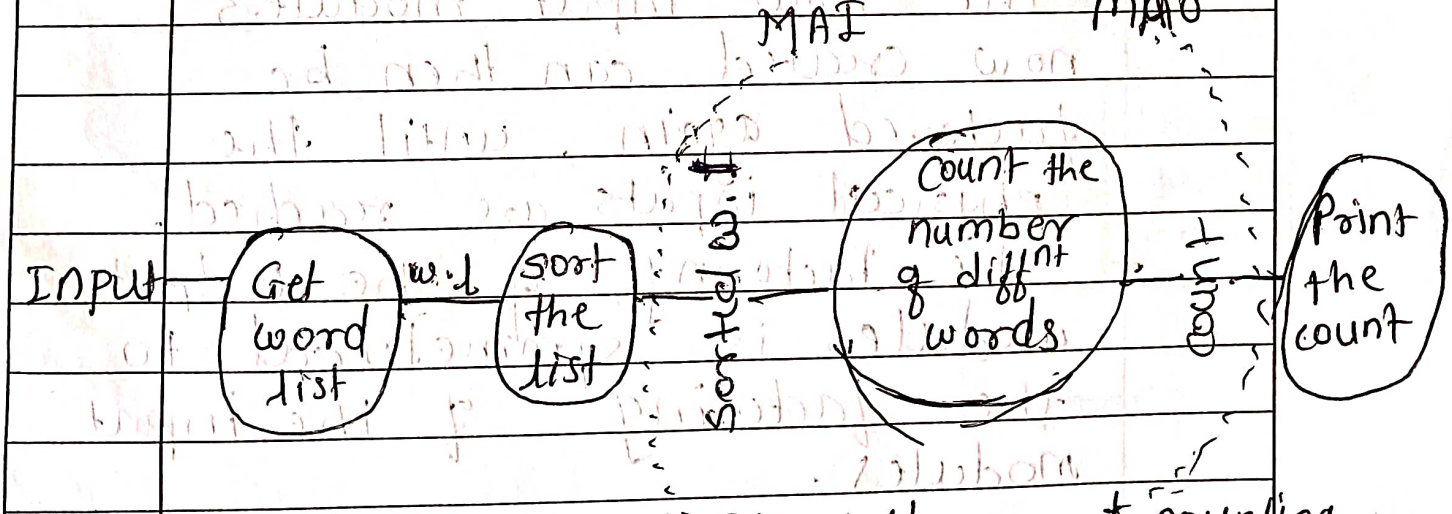


fig. 4.7 DFD for the word-counting problem.

Roll No. 16

- This problem has only one input data stream, input file, while the desired output is the count of different words in the file.
- The arcs in the data flow diagram are the most abstract input (MAI) & most abstract output (MAO).
- 1st, the input file is converted into a word list,
- A data that is a form of input will not usually be a candidate for the most abstract output.
- Thus, we have one central transform count - number of different words.
- The structured chart after the 1st level factoring of the word-counting problem is shown in fig. 4.8.

Roll No. _____

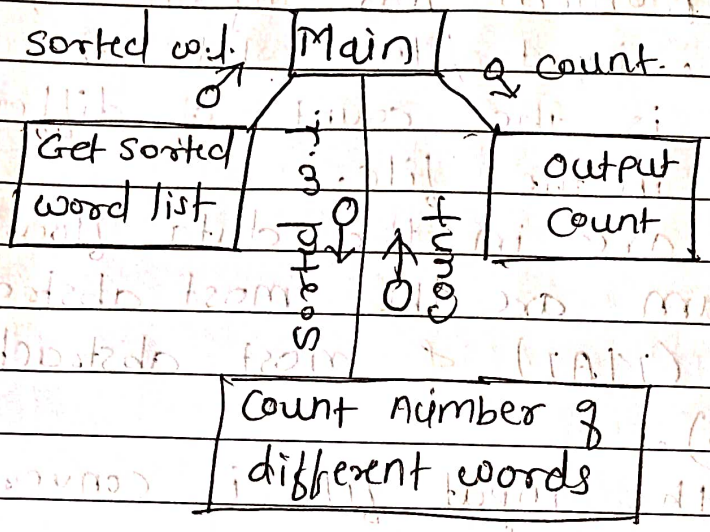


fig 4.8. First level factoring.

- In this structure, there is one input module, which returns the sorted word list to the main module.
- The output module takes from the main module the value of the count.
- There is only one central transform in this example.
- The factoring of the input module get-sorted list in the first-level structure is shown in figure 4.9.

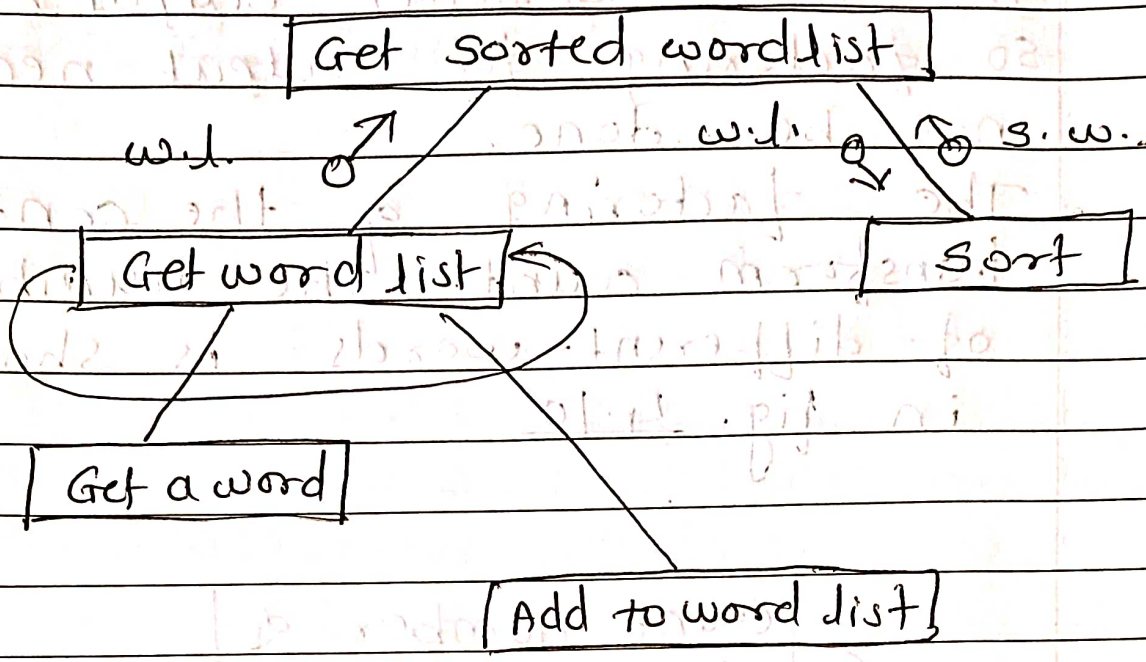


fig. 4.9. factoring the input module.

- The transform producing the input returned by this module is treated as a central transform.
- Its input is word list.
- The input module can be factored further as the module needs to perform two functions,
① getting a word ② then adding to the list.
- Note that the looping arrow is used to show the iteration.
- There is only one transform after

Roll No. _____

the most abstract output, so factoring for output need not be done.

- The factoring of the central transform count - the number of different words is shown in fig. 4.10

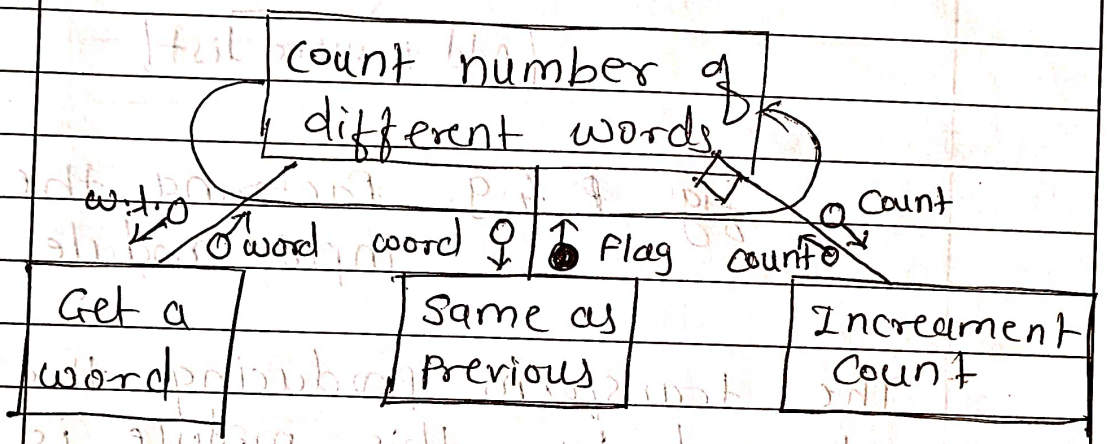


fig. 4.10 Factoring the central transform.

- To determine the number of words, we have to get a word repeatedly, determine if it is the same as the previous word, and then count the word if it is different.

Roll No. 18

4.3 Object-oriented Design :-

- An object oriented systems offers many advantages.
- An object oriented model closely represents the problem domain, which makes it easier to produce & understand design.
- Inheritance & close association of objects in design to problem domain entities encourage more reuse, thereby reducing development cost & cycle time.
- The object-oriented design approach is fundamentally different from the function-oriented design approach.
- It requires different way of thinking & partitioning.

* Object-oriented concepts :-

* Classes & objects :-
- classes & objects are basic building blocks of an OO design.

Roll No. _____

- Objects are entities that encapsulated some state & provide services to be used by a client
- The basic property of an object is: encapsulation:
- It encapsulates the data & information it contains & supports a well-defined abstraction
- The encapsulated data for an object defines state of an object
- The state & services of an object together defines its behavior.
- Objects represent the basic runtime entities in an OO system
- A class defines a possible set of objects.
- The relationship between a class & objects of that class is similar to the relationship between a type & elements of that type

Roll No. 19

* Relationships among objects:

- In object-oriented systems, an object interacts with another by sending a message to the object to perform some service it provides.
- If an object uses some services of another object, there is an encapsulation between the two objects.
- This association is also called a link — a link exist from one object to another if the object uses some services of the other objects.
- With association comes the issues of visibility, that is, which object is visible to whom.
- When A (client object) to be able to send a message to B (supplier object), B must be visible to A in the final program.

Roll No. _____

There are different ways to provide this visibility.

(1) The supplier object is globle to the client.

(2) The supplier object is a parameter to some operation of the client that sends the message.

(3) The supplier object is a part of the client object.

(4) The supplier object is locally declared in some operation.

- Another type of relationship between objects is aggregation, which reflects the whole/part of relationship.
- That is, if an object A is an aggregation of objects B & C, then objects B & C will generally be within object A.

Roll No. 20

Inheritance & polymorphism :-

Inheritance is a relation between classes that allows for definition & implementation of one class based on the definition of existing classes.

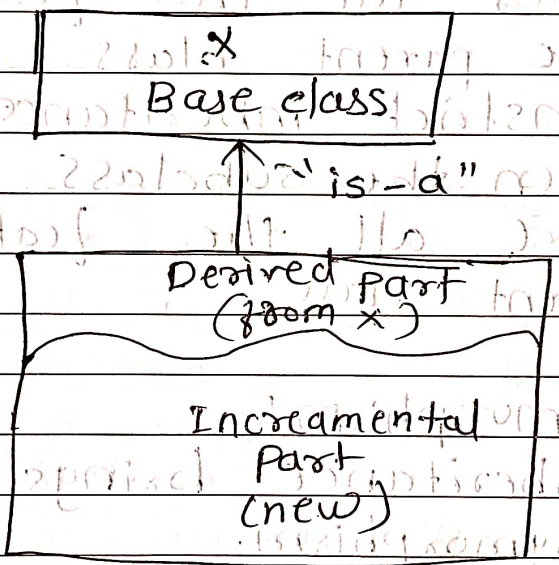


fig. Inheritance.

→ Inheritance is often called an "is-a" relation, implying that an object of type B is

Roll No. _____

also instance of type A.

- Inheritance relation beteen classes forms a hierarchy.
- Inheritance is broadly classified ~~into~~ as being of two types : strict inheritance & nonstrict inheritance
- In strict inheritance a subclass takes all the features from the parent class
- Nonstrict inheritance occurs when the subclass does not have all the features of the parent class.

Polymorphism :-

- Inheritance brings in polymorphism.
- In object-oriented programming, polymorphism comes in the form that a reference can refer to objects of different types at different ~~type~~ times.
- with polymorphism, an entity has static type & dynamic type

Roll No. 21

- The static type of an object is the type of which the object is declared in the program text & it remains unchanged.
- The dynamic type of polymorphism requires dynamic binding of operations.
- The Dynamic binding means that the code associated with given procedure call is not known until the moment of the call

* Unified Modeling Language (UML) :-

- UML is a graphical notation for expressing object-oriented designs.

Class diagram :-

- The class diagram of UML is the central piece in a design or model.
- A class diagram defines -

Roll No. _____

① classes that Exist in the system
② Association Between classes.
③. subtype, supertype Relationship.

- A class itself is represented as a rectangular box which is divided into three areas.
- The top part gives the class name.
- class name should be start with capital letter.
- The middle part lists the key attributes or fields of the class.
- The bottom part lists the methods or operations of the class.
- If a class is an interface (having specification but no body), this can be specified by marking the class with the stereotype "<<interface>>"
- If class/method/attributes has some Properties, it can be specify by tagging the entity

by specifying the property next to the entity by specifying the property next to the entity name within "{" and "}".

Queue
{private} front: int {private} rear: int {readonly} MAX: int
{public} add(element :int) {public} remove(): int {protected} isEmpty(): boolean.

<< interface Figure
area: double perimeter: double
calculateArea(): double calculatePerimeter(): double.

fig. 4.12 class, stereotypes, and tagged values.

The divided-box notation is to describe the key features of a class as a stand-alone entity.

Roll No. _____

- The generalization - specialization relationship is specified by having arrows coming from the subclass to the superclass, with the empty triangle-shaped arrowhead touching the superclass.
- In the hierarchy, specialization is done on the basis of some discriminator - a distinguishing property.
- An example of shows relationship between classes.

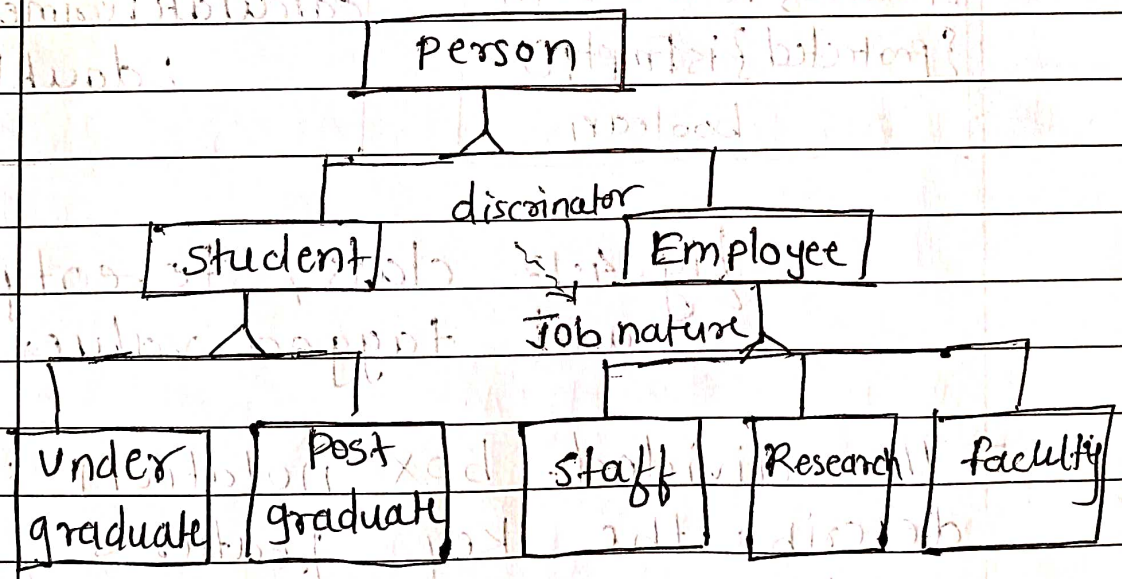


fig. A class hierarchy

Roll No. 23

- Besides the generalization-specialization relationship, another common relationship is association.
- In an association, an end may also have multiplicity allowing relationships like one-to-one, one-to-many etc. to be modeled.
- Another type of relationship is part-whole relationship. which represents an object is composed of many parts.
- For representing aggregation relationship, the class which represents "whole" is shown at the top of a line emanating from little diamond.
- The association & aggregation are shown in fig.

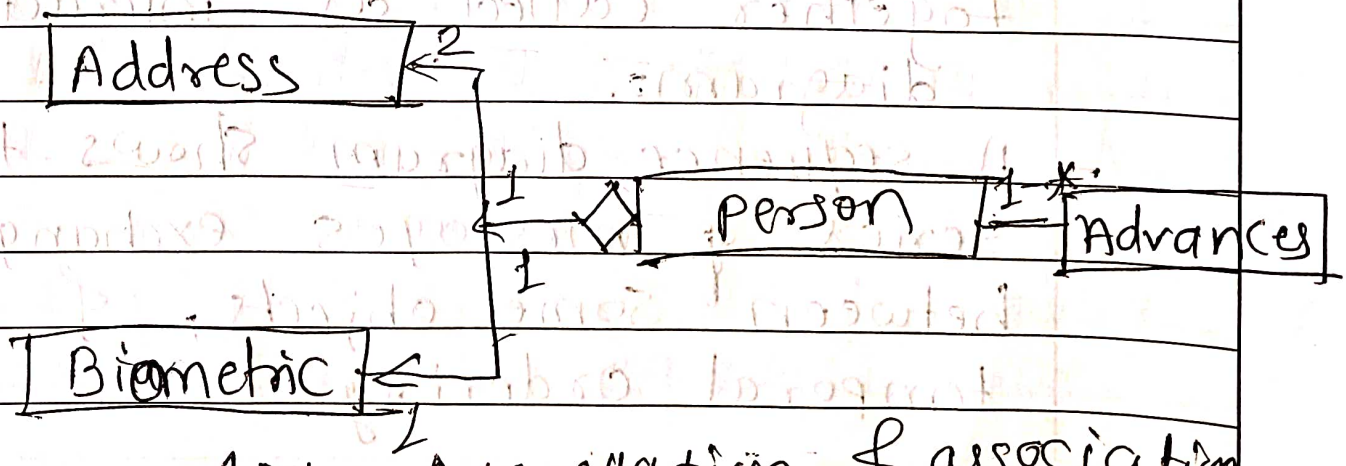


fig. Aggregation & association among classes.

Roll No. _____

- class diagram focus on classes and should not be confused with object diagram.
- To further clarify, the entire name is underlined.
- For eg. myList : List.
- The attributes of an object may have specific values.
- These values can be specified by giving them along with attribute name (eg. name = "John").

* Sequence & collaboration Diagrams:-

- class diagram, however, do not represent the dynamic behaviour of the system.
- This is done through sequence diagram or collaboration diagrams together called as interaction diagrams.
- A sequence diagram shows the series of messages exchanged between some objects, & their temporal ordering.

Roll No. 24

- In a sequence diagram, it is objects that participate & not classes.
- In this diagram, all the objects that participate in the interaction are shown at the top as boxes with object names.
- For each object, a vertical bar representing its lifeline is drawn downwards.
- Each message is labled with the message name.
- Time is represented by the Y-axis increasing downwards.
- Frequently message is sent from one object to another only under some condition.
- This condition can be represented in the sequence diagram by specifying it within brackets before the message name.
- If message is sent to multiple receiver objects then this multiplicity is shown by having '*' before the message name.

Roll No. _____

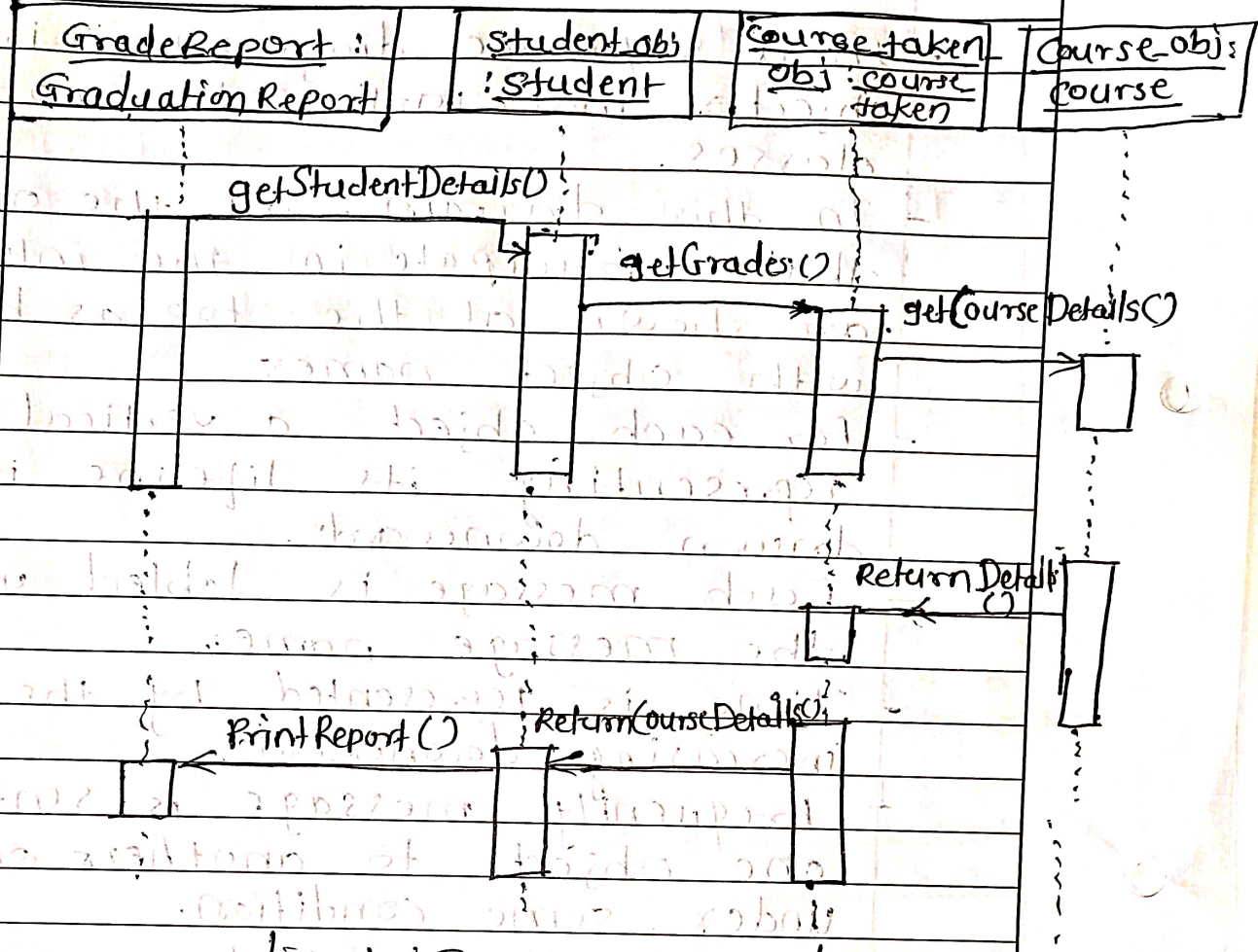


Fig. 4.15 Sequence diagram for Printing graduation Report.

- Return message can be shown by dashed arrow.

- A collaboration diagram also shows how objects ~~are~~ communicate.
- Collaboration diagram looks more like a state diagram.
- The message sent from one object to another are shown as numbered arrows from one object to the other.

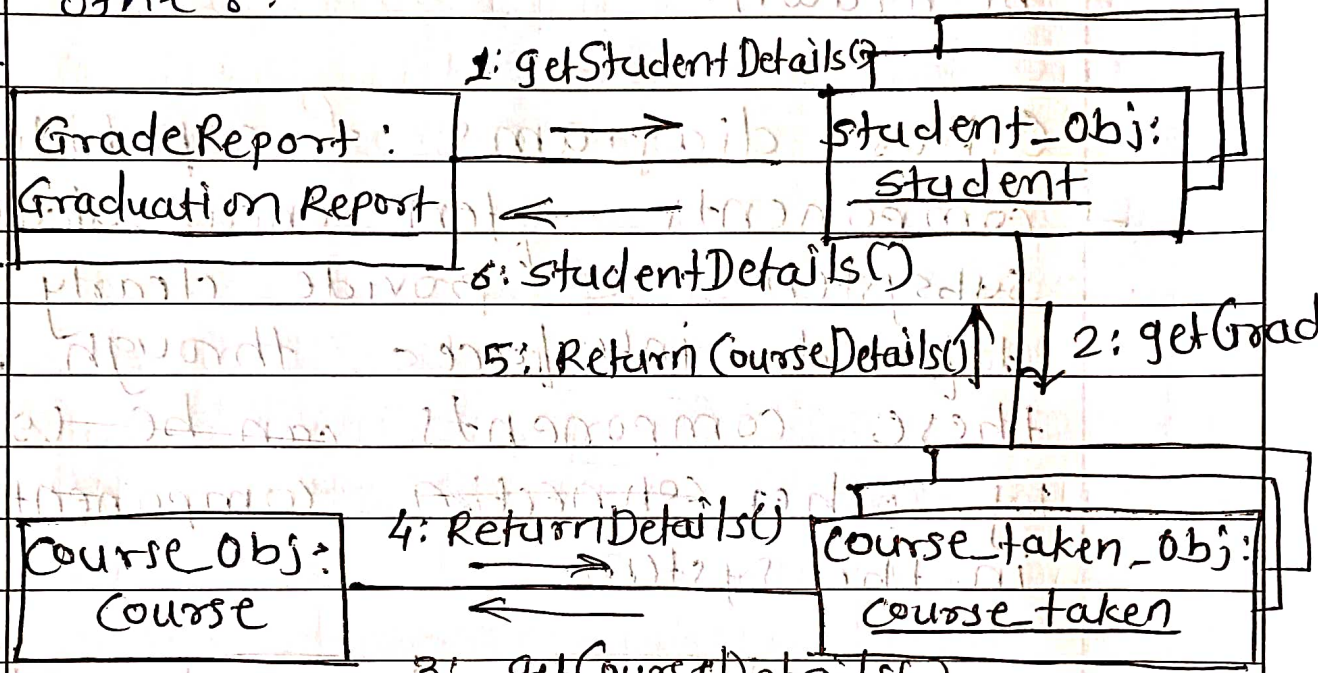


fig. 4.16. collaboration diagram for Printing a graduation Report.

- One class diagram can capture the structure of the system's code, for the dynamic behaviour many

Roll No. _____

diagrams are needed.
- However, it may not be feasible or practical to draw the interaction diagram for each use-case scenario.
- An interaction diagram of some key use cases or functions will be drawn.

Other diagrams & capabilities:-
- components often encapsulate subsystems & provide clearly defined interfaces through which these components can be used by other ~~connecting~~ components in the system.

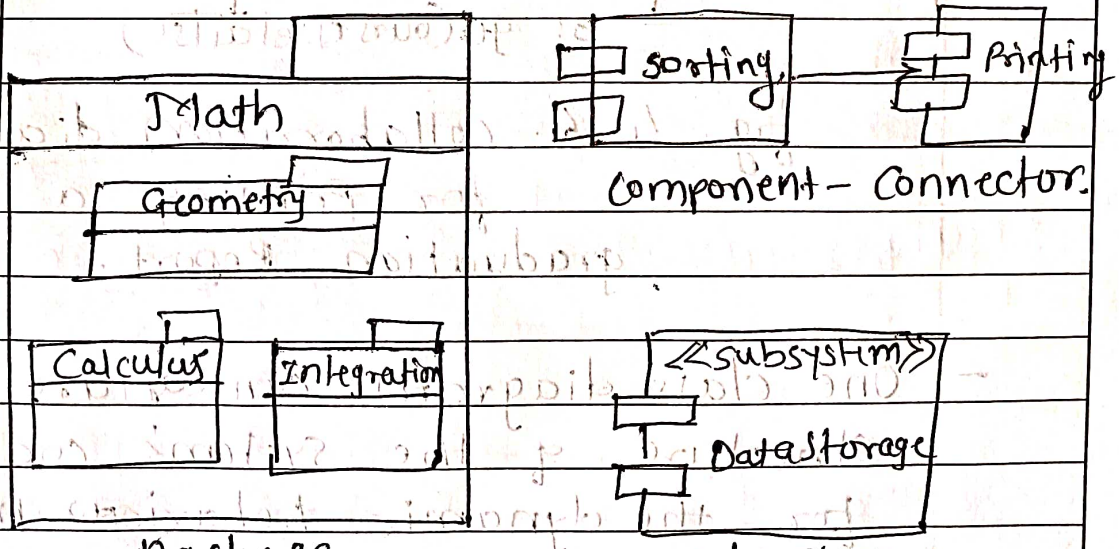


fig. 4.17 subsystems, components & Packages.

Roll No. 26

- The UML has notation for representing a deployment view.
- the main element is a node, represented as a named cube,
- If different nodes communicate with each other, that this is shown by connecting the nodes with lines.
- A state diagram is a model in which the entity being modeled is viewed as a set of states.
- State is represented as a rectangle with rounded edges or as ellips or circle., transitions are represented by arrows. connecting two state.
- state diagrams are often used to model the behavior of object of a class.
- Activity diagram is another diagram for modeling dynamic behavior.
- Each activity is represented as an oval, with the name of activity within it.
- often, which activity to perform

Roll No. _____

next depends on some desisions. This decisions is shown as a diagmond leading to multiple activity.

- These diagrams are like flow-chart, but also have notation to specify parallel execution of activities

Design methodology.

- A methodology basically uses the concepts to provide guidline for the design activity.
- Most of the classes & objects and their relationships are identified during design.
- An approach for creating an object oriented design consist of the following sequences of steps:-

1. Identify classes & relationships between them.
2. Develop the dynamic model & use it to define operations on classes.

Roll No. 27

3. Develop the functional model & use it to define operations on classes.
4. Identify internal classes & operations.
5. Optimize & package.

* Identify classes & Relationships:-

- Identifying the classes & their relationship requires identification of object types in the ~~system~~ problem domain, the structures between classes (both inheritance and aggregation)
- Include an entity as an object.
- class have attributes.
- for eg. class person, the attributes could be the name, sex & address.
- while modeling a hospital system, for the class person, attributes of height, weight & date of birth may be needed.

Roll No. _____

- If the attributes is a common attribute, it should be placed in the superclass.
- If it is specific to a specialized object, it should be placed with the subclass.
- For class diagram, we also need to identify the structures & association between classes.
- Identify the classification structures as generalization & specialization relationship.
- For association we need to identify the relationship between instances of various classes.
- For eg. an instance of the class company may be related to an instance of the class person by an "employs" relationship.
- An instance connection may be of 1:1 type or it could be 1:M, or it could be M:M.

Roll No. 28

* Dynamic Modeling:-

- The dynamic model of the system aims to specify how the state of various objects, an event is essentially a request for an operation.
- Events can be internal events to the system, in which case the event initiator & the event responder are both within the system.
- An event can be an external event, in which case the event initiator is outside the system
- Dynamic modeling involves preparing interaction diagrams.
- First the main success scenarios should be modeled & then scenarios for exceptional cases should be modeled
- For eg. in a restaurant, the main success scenario for placing order could be the following sequence of actions:

Custome reads the menu;

Roll No. _____

customer places an order; order is sent to the kitchen for preparation; ordered items are served; customer requests a bill for the order; bill is prepared for this order; customer is given the bill; customer pays the bill.

- An exception scenario could be if the ordered item was not available or if the customer cancels the order.
- A possible sequence diagram of main success scenario of the ~~sys~~ restaurant is given in fig. 4.18.

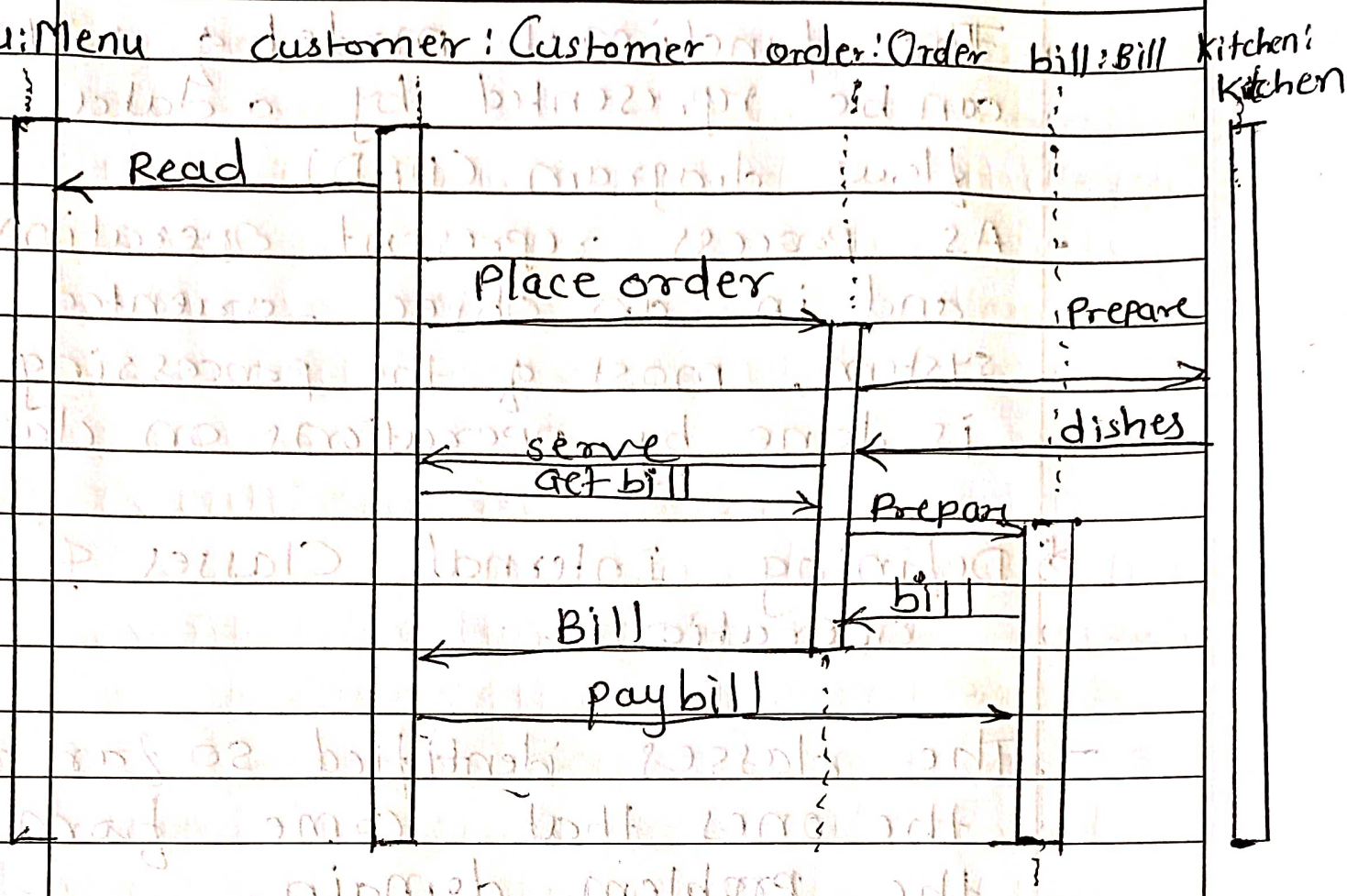


fig. A sequence diagram for the restaurant.
or Event trace diagram.

* Functional Modeling :-

A functional model of a system specifies how the output values are computed in the system from the input values, without considering the control aspect of the computation.

Roll No. _____

- The functional model of a system can be represented by a data flow diagram (DFD).
- As process represent operations, and in an object-oriented system, most of the processing is done by operations on classes.

* Defining Internal Classes & operations:-

- The classes identified so far are the ones that come from the problem domain.
- While considering implementation issues, algorithm & optimization issues arise. These issues are handled in this step.
- some of the classes might be discarded if the designer feels they are not needed during implementation.
- Then the implementation of operations on the classes is considered.

Roll No. 30

- Once the implementation of each class of each operation on the class has been considered and it has been satisfied that they can be implemented, the system design is complete.

* Optimize of Package :-

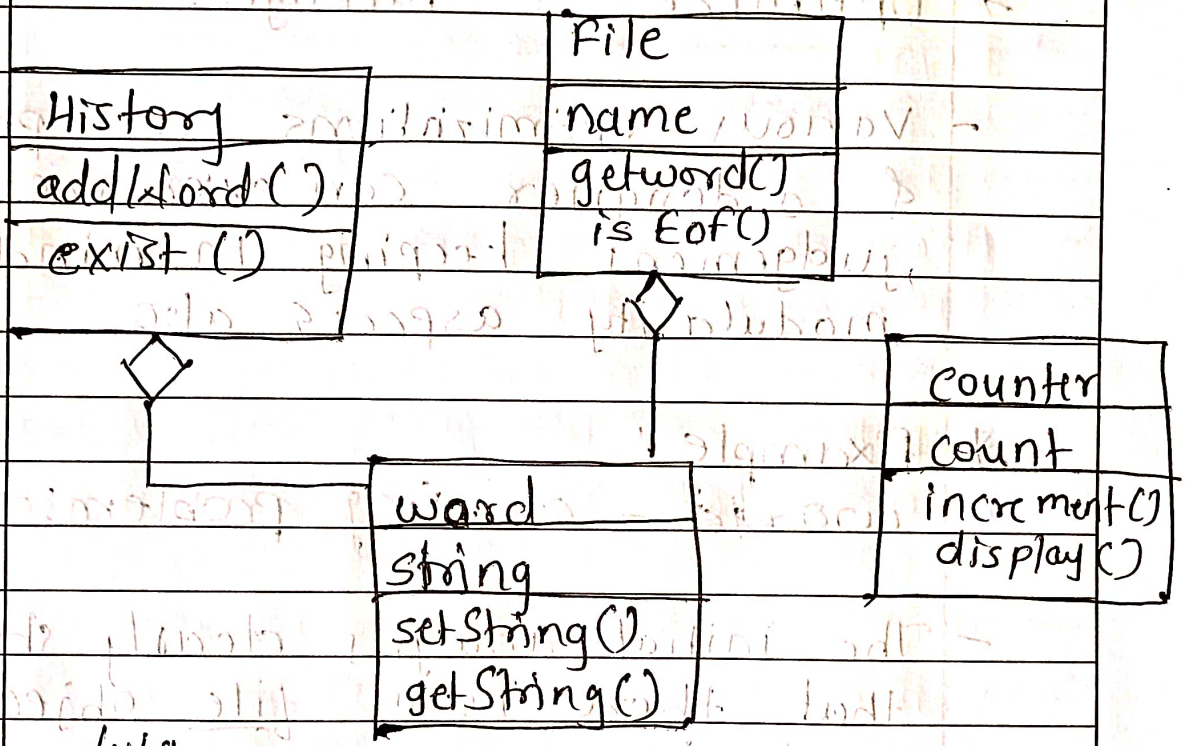
- Various optimizations are possible & a designer can exercise his judgement keeping in mind the modularity aspects also.

* Example :-
word - counting problem :-

- The initial analysis clearly shows that there is a file object which is an aggregation of many word objects.
- It is a matter of preference & opinion whether counter should be an object or counting should be implemented as an operation.

Roll No. _____

- However, further analysis for services reveals that some history mechanism is needed to check if the word is unique.
- The class diagram obtained after doing the initial modeling is show in figure 4.19.



4.19

fig A class diagram for the word-counting problem

Roll No. (31)

- Now let us consider the dynamic modeling for this problem.
- The scenario for the "normal" case can be:
  1) System prompts for the file name : user enters the file name.
  2) System checks for existance of the file
  3) System reads the words from the file
  4) System prints the count.

- From this simple scenario, no new operations are uncovered and our class diagram stays unchanged.
- One possible functional model is shown in figure 4.20.
- This model reinforces the need for some objects.
- This object is used to check the uniqueness of the words..

Roll No. _____

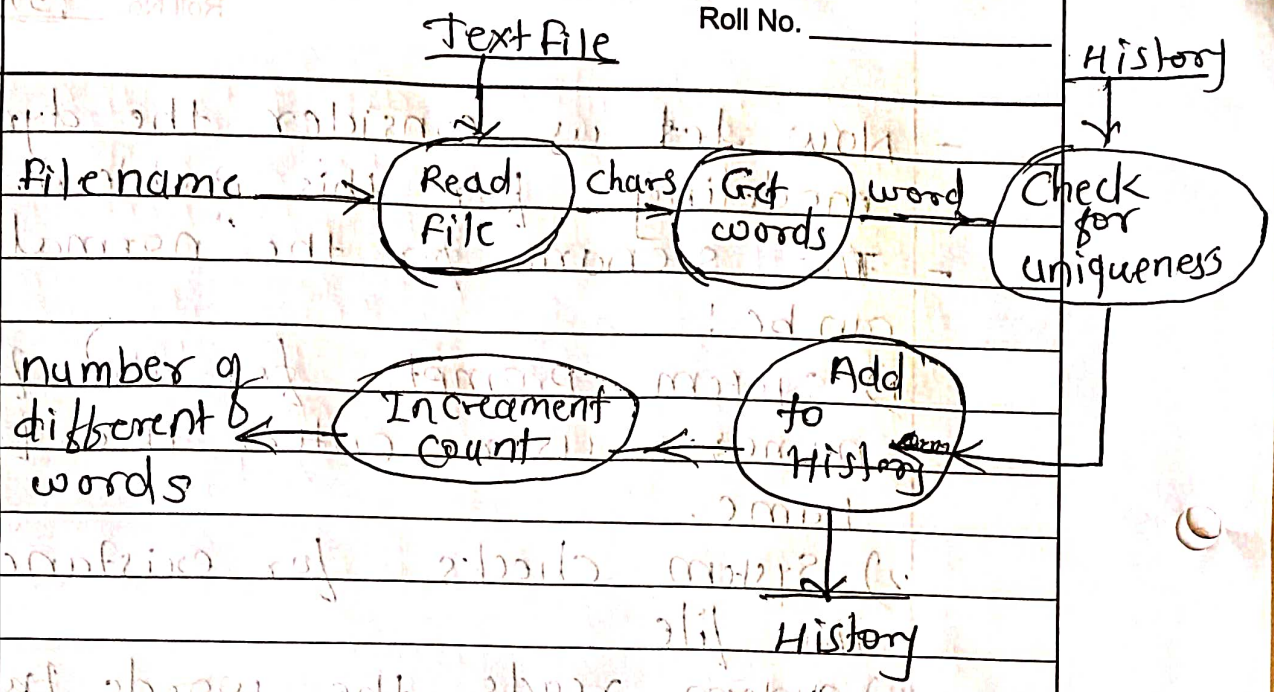


fig. 4.20. Functional model for the word-counting problem

- Now we are at the last two steps of design methodology, where implementation & optimization concerns are used to enhance the object model.
- First decision we take is that the history mechanism will be implemented by a binary search tree.
- Instead of the class History, we have a different class BTree.

The final specification of this design is given below. This specification can be used as a basis of implementing the design.

```
Class Word {
  Private:
    char * string ; // string representing the word.
  Public:
  bool operator == (word); // checks for
  bool operator < (word);      equality.
  bool operator > (word);
  word operator = (word); // The assignment operator.
  void setWord (char *); // sets the string for the word.
  char * getWord (); // gets the string for the word.
};
class File {
  Private:
      File infile;
      char * fileName;
  Public:
      word getword () ; // get a word;
                          invokes operation of word
```

Roll No. _____

```
bool isEof ();  // check for end of file
void fileOpen (char *);
};
class counter {
  private:
      int counter;
  public:
      void increment();
      void display();
};
class Btree : GENERIC in <ELEMENT_TYPE>{
  private:
    ELEMENT_TYPE element;
      BTree < ELEMENT_TYPE > * left;
    BTree < ELEMENT_TYPE > * right
  public:
  void insert (ELEMENT_TYPE); // to insert an
                                element
  bool lookup (ELEMENT_TYPE); // to check if an
                                element exists
};
```

Roll No. 33

4.4. Detailed Design :-

- Once the modules are identified and specified during the high level design, the internal logic that will implement the given specifications can be designed & is the focus of this section.

Logic & Algorithm Design.

- The basic goal in detailed design is to specify the logic for the different modules that have been specified during system design.
- Specifying the logic will required developing an algorithm.
- Principles for designing algorithms.
- An algorithm is to be an unambiguous procedure for solving a problem
- A procedure is a finite sequence of well-defined steps or operations, each of which require a finite amount of memory &

Roll No. _____

time to complete.

- There are a number of steps:-

(1) The starting step in the design of algorithms is statement of the problem

(2) The next step is development of a mathematical model for the problem.

(3) The next step is the design of the algorithm.
During this step the data structure & program structure are decided
- once the algorim is design, its correctness should be verified

(4) The most common method for designing algorithms for the logic for a module is to use the stepwise refiniement technique.

- The Step wise refinement technique breaks the logic design problem

into a series of steps.
- The process starts by converting the specifications of the module into an abstract description of an algorithm containing a few abstract statements.
- In each step, one or several statements in the algorithm developed so far are decomposed into more detailed instructions.
- The successive refinement terminates when all instructions are sufficiently precise that they can easily be converted into programming language statements.
- Detailed design is not specified using formal programming languages, but using languages that have formal programming language like outer structures.

Roll No. _____

State Modeling of classes:-

- An object of a class has some state & many operations on it.
- A method to understand the behaviour of a class is to view it as a finite state automaton, which consist of states & transition between states.
- When modeling an object, the state is the value of its attributes, and an event is the performing of an operation on the object.
- A state diagram attempts to represent only the logical states of the object.
- eg: push will add an element, pop will remore one. etc.
- The state representing an empty stack is different as the behaviour of top & pop operations are different now.
- similarly, state representing a full stack is different.

as shown in figure 4.23.

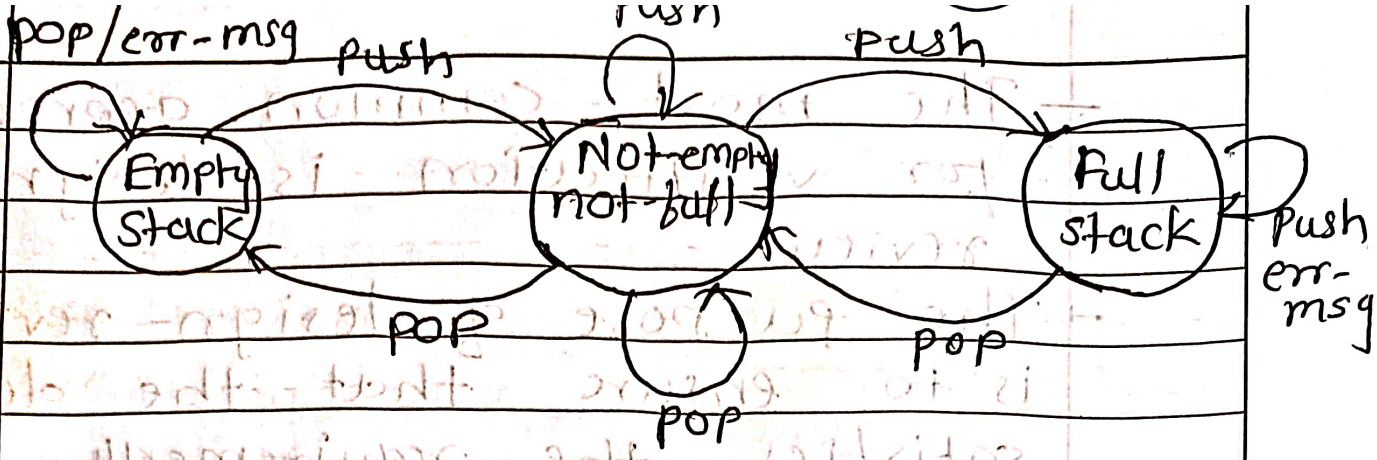


fig. FSA model of a stack.

- A state model can be used for generating test cases & for validation.

4.5. Verification :-

- If the design is expressed in some formal notation for which analysis tools are available, then through tools it can be checked for internal consistency (e.g. those modules used by another are defined, the interface of a module is consistent with the way others use it, data usage is consistent with declaration, etc)

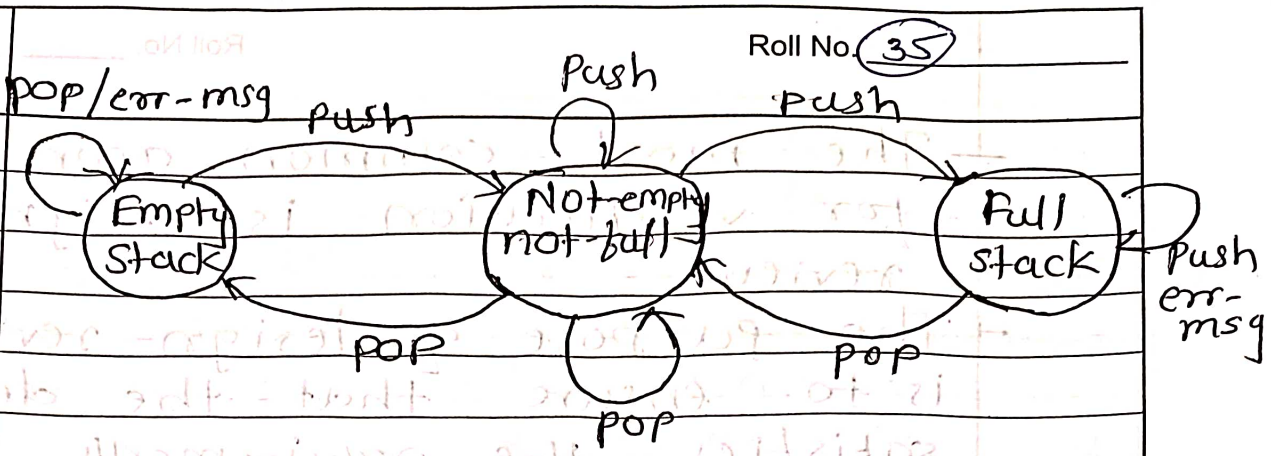


fig. FSA model of a stack.

- A state model can be used for generating test cases & for validation.

4.5. Verification :-

- If the design is expressed in some formal notation for which analysis tools are available, then through tools it can be checked for internal consistency (e.g. those modules used by another are defined, the interface of a module is consistent with the way others use it, data usage is consistent with declaration, etc)

Roll No. _____

- The most common approach for verification is design review.
- The purpose of design reviews is to ensure that the design satisfies the requirements & is of good quality.
- If error ~~msg~~ are made during the design process, they will ultimately reflect themselves in the code of the final system.
- Detecting errors in design is the purpose of design reviews.
- The system design review process is similar to the inspection process in that a group of people get together to discuss the design with the aim of revealing design errors or undesirable properties.
- The review group must include a member of both the system design team & the detailed design team, the author of

the requirements document, the author responsible for maintaining the design document, and an independent software quality ~~design~~ engineer.

- As with any review, it should be kept in mind that the aim of the meeting is to uncover design errors, not to try to fix them, fixing ~~to~~ is done later.
- The most important design errors, however, is that the design does not fully support some requirements.
- For design quality, modularity is the main criterion. However, efficiency is another key property for which a design is evaluated.

Roll No. _____

4.6. Metrics :-

- Size is always a product metric of interest.
- For size of a design, the total number of modules is a commonly used metric. (By using an average size of a module, from this metric the final size in LOC can be estimated & compared with project estimates).
- Another metric of interest is complexity.
- A possible use of complexity metrics at design time is to improve the design by reducing the complexity of the modules that have been found to be most complex.
- This will directly improve the testability & maintainability.

Roll No. 37

* Complexity Metrics for Function-oriented Design:-

Network Metrics :-

- Network Metrics is a complexity metrics that tries to capture how "good" the structure chart is.
- A Good structure is considered one that has exactly one caller.
- That is, the call graph structure is Simplest if it is a pure tree.
- Deviation of the tree is then defined as the graph impurity of the design.
- Graph impurity is defined as :-

Graph impurity = n - e - 1

where n is the number of nodes in the Structure chart & e is

Roll No. _____

the number of edges.
- As in a pure tree the total number of nodes is one more that the no. of edges, the graph impurity for a tree is 0.
- Each time a module has Fan-in of more than one, the graph impurity increases.

Information flow metrics :-

- As the graph impurity increases, the coupling increases.
- The information flow metrics attempts to define the complexity in terms of the total information flowing through a module.
- The intramodule complexity is approximately by the size of the module in lines of code
- The intermodule complexity of a module depends on the total information flowing in the module (inflow) and the

~~the~~

total information flowing out of the module (outflow).
- The module design complexity, Dc, is defined as

$$Dc = size * (inflow * outflow)^2$$

- The term (inflow * outflow) refers to the total number of combinations of input source & output destination.
- The modules with more interacti connections are harder to test or modify.
- A varient of this was proposed based on the hypothesis that the module complexity depends not only on the information flowing in & out, but also on the number of modules to or from which it is flowing.
- Complexity Dc, for a module defined as

$$Dc = fan\_in * fan\_out + inflow * outflow.$$

Roll No. _____

where fan in represents the number of modules that call this module & fan-out is the number of module this module calls.

Let avg_complexity be the average complexity of the modules in the design being evaluated, & let std_deviation be the standard deviation in the design complexity of the modules of the system.

Complexity Metrics for OO design:-

As design of classes is the central issue in OOD and the major output of any OOD methodology is the class definition, these metrics focus on evaluating classes

Roll No. 39

Weighted Methods per Class (WMC):-

- The number of methods the class has & the complexity of the methods.
- Hence, a complexity metrics that combines the number of methods & the complexity of methods can be useful in estimating the overall complexity of the class.
- WMC is defined as,

$$WMC = \sum_{i=1}^{i=n} c_i$$

- If the complexity of each method is considered to be 1, WMC gives the total number of methods in the class.

Roll No. _____

* Depth of Inheritance Tree (DIT)

- Inheritance is the unique features of the object-oriented paradigm.
- Inheritance increases coupling, which makes changing a class harder.
- The DIT of a class C in an inheritance hierarchy is the depth from the root class in the inheritance tree.
- In other words, it is the length of the shortest path from the root of the tree to the node representing C

* coupling between classes (CBC)

- coupling between classes is a metric that tries to qualify coupling that exist between classes.
- The CBC value for a class C is the total number of other

classes to which the class is coupled.

Response for a class (RFC):-

- It does not quatify the "strength" of interconnects.
- In other words, it does not explain the degree. of connection of methods of a class with other classes.
- Response for a class (RFC) tries to quatify this by capturing the total number of methods that can be invoked from an object of this class
- The RFC value for a class C is the cardinality of the response set for a class
- It is clear that even if the CBC value of a class is 1.
- ~~RFC~~ RFC value may be quite high, including that the "volume" of interaction between the two classes is very high