Unit – V

Software Architecture and Design

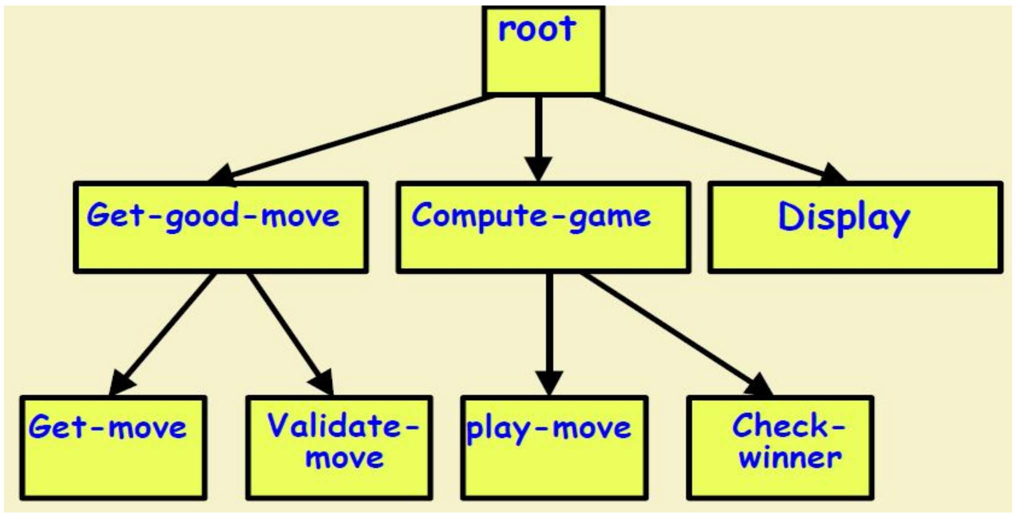
5.1 Role of Software Architecture

- What is achieved during design phase?
- Typically, we transform SRS document to design document. And the design can easily be implemented in some programming language.
- Design is usually classified into two types;

  1. High level design (also called Software Architecture)

  2. Detailed design
- Generally speaking, architecture of a system provides a very high level view of the parts of the system and how they are related to form the whole system.
- Formally we define software architecture as

  The software architecture of a system is the structure of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.
- For example, consider the Tic-tac game. Its architecture can be shown below.



- Some of the important uses that software architecture play are;

1. **Understanding and Communication** – There are several stakeholders for a software system – users (who use the system),

clients (who pay for the system), analysts (who analyze the requirements of the system), designers (who design), testers (who test), document writers (who prepare user manuals) etc. Through the architecture, the stakeholders gain an understanding of the system. In addition, it acts as a vehicle for providing communication between the stakeholders.

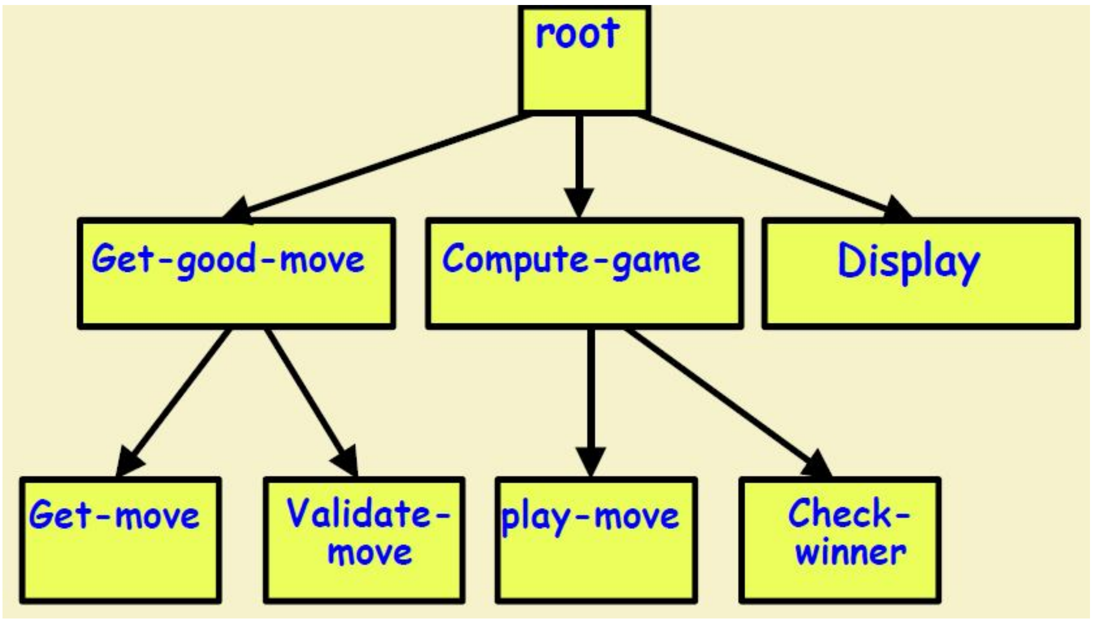
2. **Reuse** – For a long time, the software engineering world is been working toward a discipline where software can be assembled from parts that are developed by different people and are available for others to use. For example, the components like date, money, APIs etc are the reusable components which are used in many software systems. The architecture facilitates the analysts to decide which components can be reused and which can be developed. This is a crucial decision which speeds up the development of the software system.
3. **Construction and Evolution** – As we know that, the software architecture partitions the system into parts that are relatively independent to each other. This allows that different development teams to work on simultaneously on different parts.
4. **Analysis** – One of the important properties of the architecture is that it can be used to analyze the system before the system is actually built. It allows the designers to analyze the system in terms of its cost, reliability, performance, etc. For example, while building a website for shopping, it is possible to analyze the response time or throughput for a proposed architecture, given some assumptions about the request load and hardware. It can then be decided whether the performance is satisfactory or not, and if not, what new capabilities should be added (for example, a different architecture or a faster server for the back end) to improve it to a satisfactory level.

5.2. Architecture Views

- In general, there is no unique architecture of a system. For example, consider the construction of a multi-storied building. In order to construct it, several plans like floor plans, plumbing plan, electricity plan, fire safety plan, elevation plan etc are created. These drawings are not independent of each other - they are all about the same building. However, each drawing provides a different view of the building.
- Similarly, in software development also several plans are created. These different plans are called as views of the system. A view represents the system as composed of different components of the system and their relationships.
- Many types of views have been proposed. Most of the proposed views generally belong to one of these three types;
    1. Module view
    2. Component and Connector view
    3. Allocation view

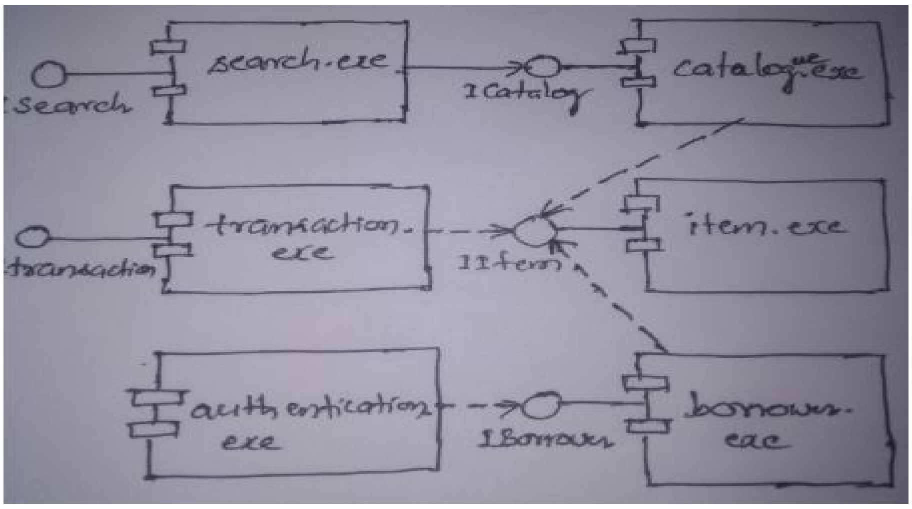
Module View

- In a module view, the system is viewed as a collection of modules and their relationships.
- Each module performs a well defined functionality. For example, the module view of Tic-tac software is shown below.



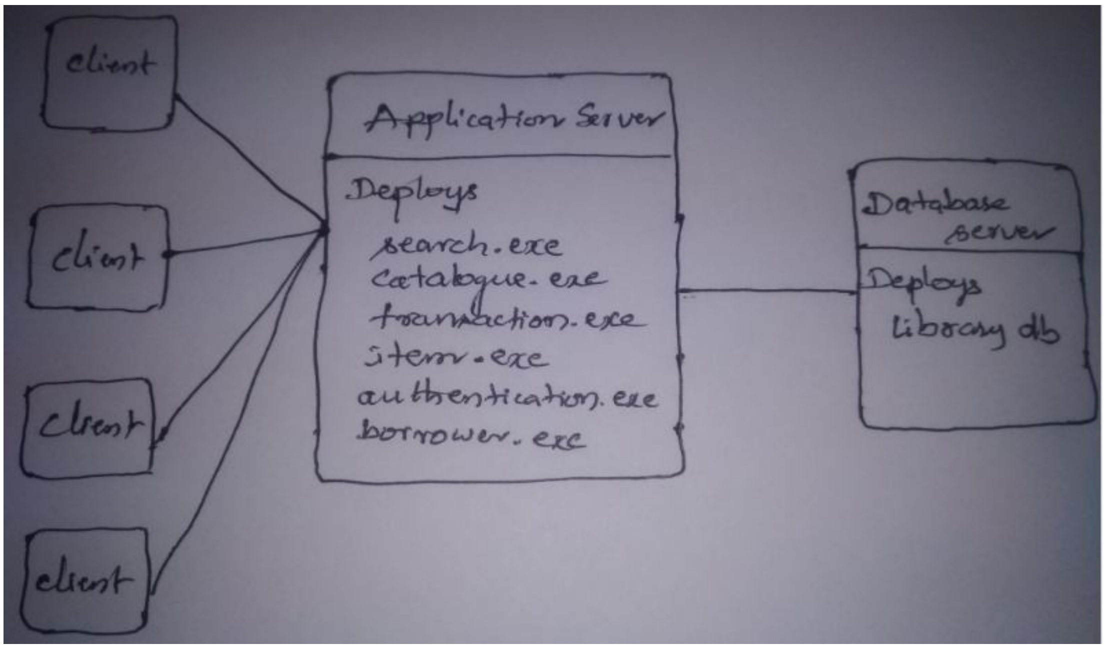
Component and Connector View

- In a component and connector (C&C) view, the system is viewed as a collection of runtime entities called components.
- A component is a unit which has an identity in the executing system. Objects (not classes), a collection of objects, and a process are examples of components.
- While executing, components need to interact with others to support the system services. Connectors provide means for this interaction. Examples of connectors are interfaces, pipes and sockets.
- For example, the component and connector view of library system is shown below.



Allocation View

- An allocation view focuses on how the different software units are allocated to resources like the hardware, file systems, and people.
- That is, an allocation view specifies the relationship between software elements and the environments in which the software system is executed.
- They expose structural properties like which processes run on which processor, and how the system files are organized on a file system.
- For example, the allocation view of library system is shown below.



5.5 Function Oriented Design

- In this, the system is looked upon as a set of functions.
- That is, each function specified in the SRS document is decomposed into more detailed functions.
- That is why it is also called as the top-down approach.
- Functions are then mapped to a module structure.
- For example, the function **create-new-library-member** can be decomposed into sub functions as
  - creates the record for a new member,
  - assigns a unique membership number

  - prints a bill towards the membership
- Function oriented design can be graphically represented using a structure chart.

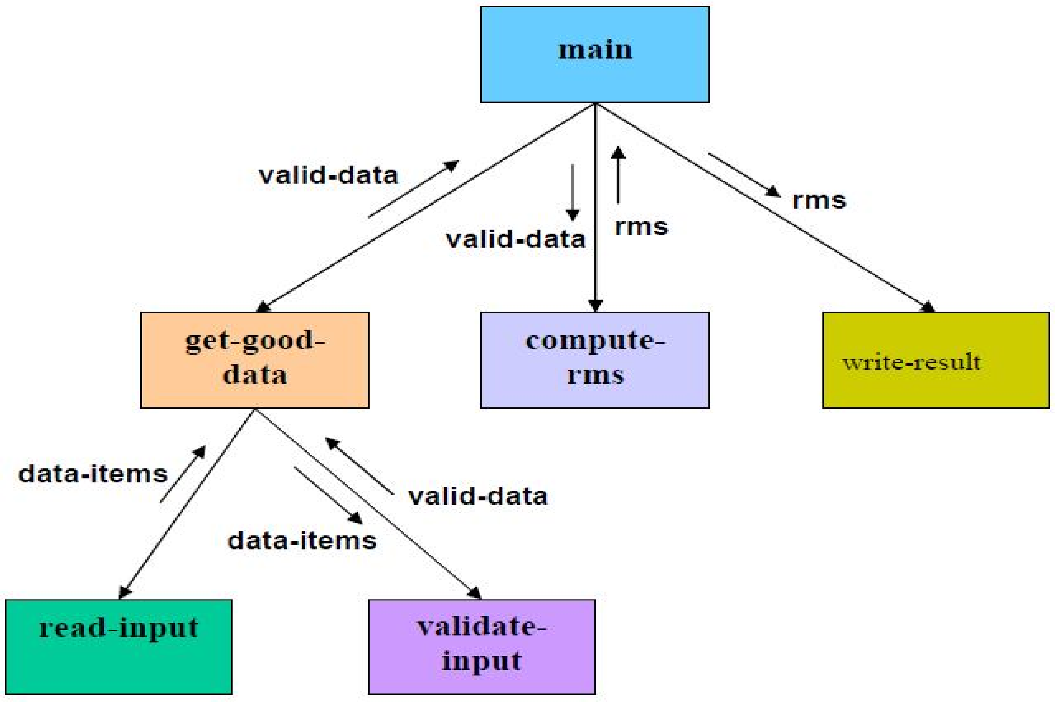
5.5.1 Structure Chart

- A structure chart represents the software architecture, i.e. the various modules making up the system, the dependency (which module calls which other modules), and the parameters that are passed among the different modules.
- Hence, the structure chart representation can be easily implemented using some programming language.
- Since the main focus in a structure chart representation is on the module structure of the software and the interactions among different modules, the procedural aspects (e.g. how a particular functionality is achieved) are not represented.
- The basic building blocks which are used to design structure charts are the following:
  - **Rectangular boxes:** Represents a module.
  - **Module invocation arrows:** Control is passed from one module to another module in the direction of the connecting arrow.
  - **Data flow arrows:** Arrows are annotated with data name; named data passes from one module to another module in the direction of the arrow.
  - Library modules: Represented by a rectangle with double edges.
  - **Selection:** Represented by a diamond symbol.
  - Repetition: Represented by a loop around the control flow arrow.
- For example, the following figure shows the structure chart for RMS (Root Mean Square) software. RMS is a kind of average used by statisticians and engineers. The RMS of a set of numbers can be computed as follows;
  - SQUARE all the values
  - Take the average of the squares
  - Take the square root of the average

Formula: $$\text{RMS} = \sqrt{\frac{a_1^2 + a_2^2 + a_3^2 + \cdots + a_n^2}{n}} \quad \text{or} \quad \sqrt{\frac{\sum_{i=1}^{n} a_i^2}{n}}$$

Example: For the numbers 4 and 9,

$$\text{RMS} = \sqrt{\frac{4^2 + 9^2}{2}} \approx 6.96$$



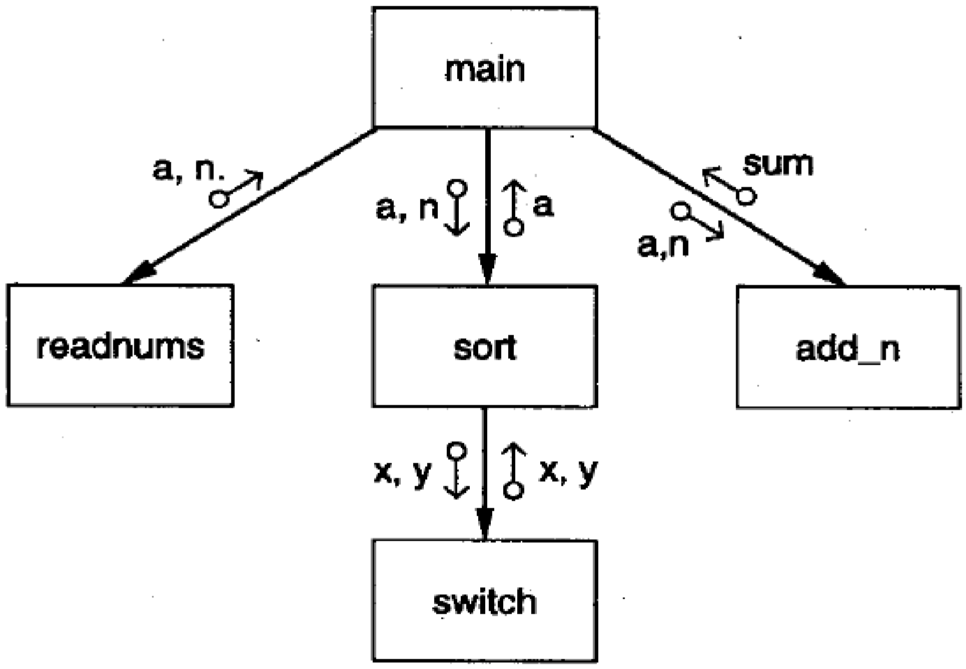
- In a structure chart, an arrow from module A to module B represents that module A invokes module B.
- As another example, consider the following program, whose structure chart is shown below.

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

readnums(a, N)
int a[], *N;
{
     :
}

sort(a, N)
int a[], N;
{
     :
     if (a[i] > a[t]) switch(a[i], a[t]);
     :
}

/* Add the first n numbers of a */
add_n(a, n)
int a[], n;
{
     :
```



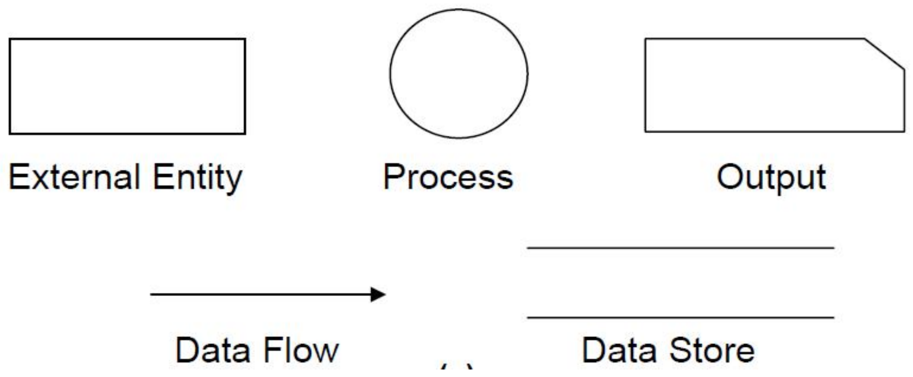
5.5.2 Structured Design Methodology

- The basic principle behind the structured design methodology is problem partitioning.

- That is, the system is decomposed into modules, modules are then decomposed into sub modules, sub modules are then decomposed into sub-sub modules and so on until the modules are small enough to handle them easily. This forms the hierarchy of modules.
- There are four major steps in the methodology.
    1. Restate the problem as a data flow diagram
    2. First-level factoring
    3. Identify the input and output data elements
    4. Factoring of input, output, and transform branches

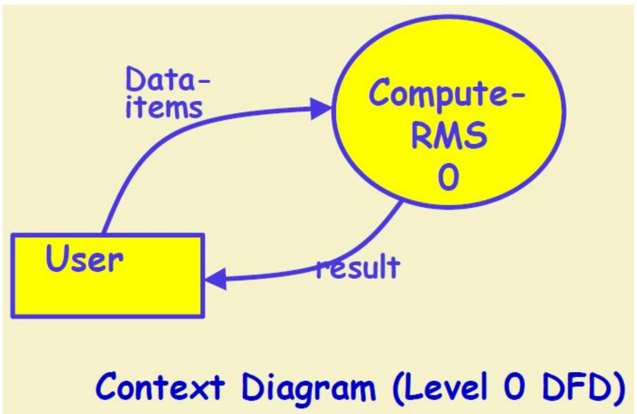
1. Restate the problem as a data flow diagram

- To use this methodology, the first step is to construct the data flow diagram for the problem.
- The DFD (also known as a bubble chart) is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data exchanged among these functions.
- Each function is considered as a processing station (or process) that consumes some input data and produces some output data.
- The system is represented in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system.
- A DFD model uses a very limited number of primitive symbols as shown below.



Example 1: RMS calculating software

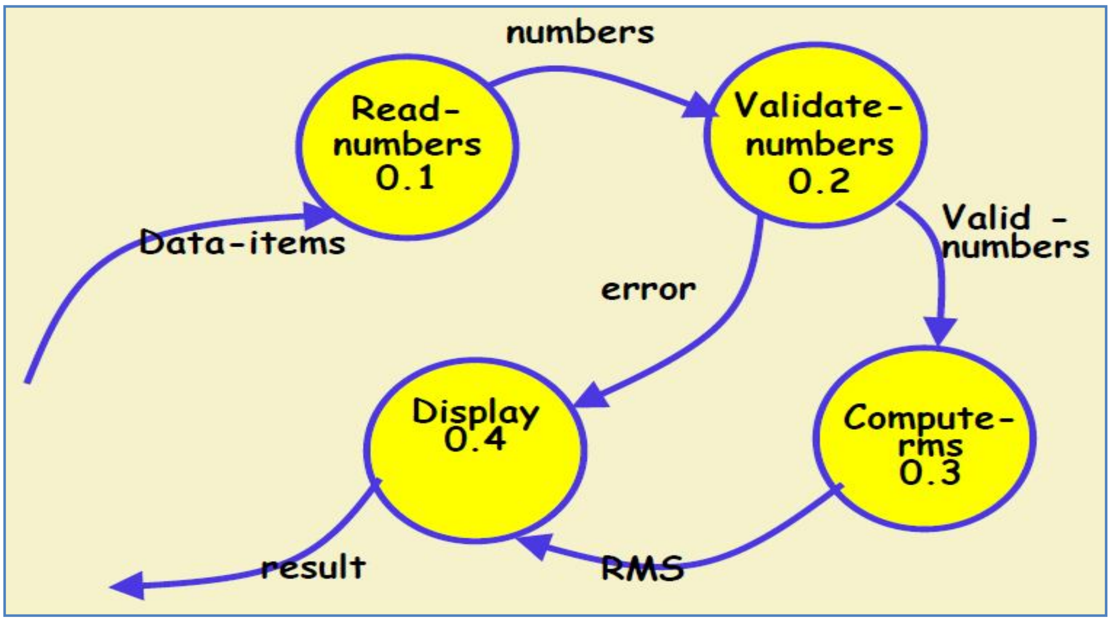
The RMS calculating software would read three integer numbers from the user in the range of -1000 and +1000 and then determine the root mean square (RMS) of the three input numbers and display it. In this example, the context diagram (shown here) is simple to draw. The system accepts three integers from the user and returns the result to him.



Context Diagram (Level 0 DFD)

2. First-level Factoring

Each bubble in the DFD represents a function performed by the system. The bubbles are decomposed into sub-functions at the successive levels of the DFD. Decomposition of a bubble is also known as factoring a bubble. For example, in the above DFD *Compute-RMS* can be factored as shown below leading to level 1 DFD.



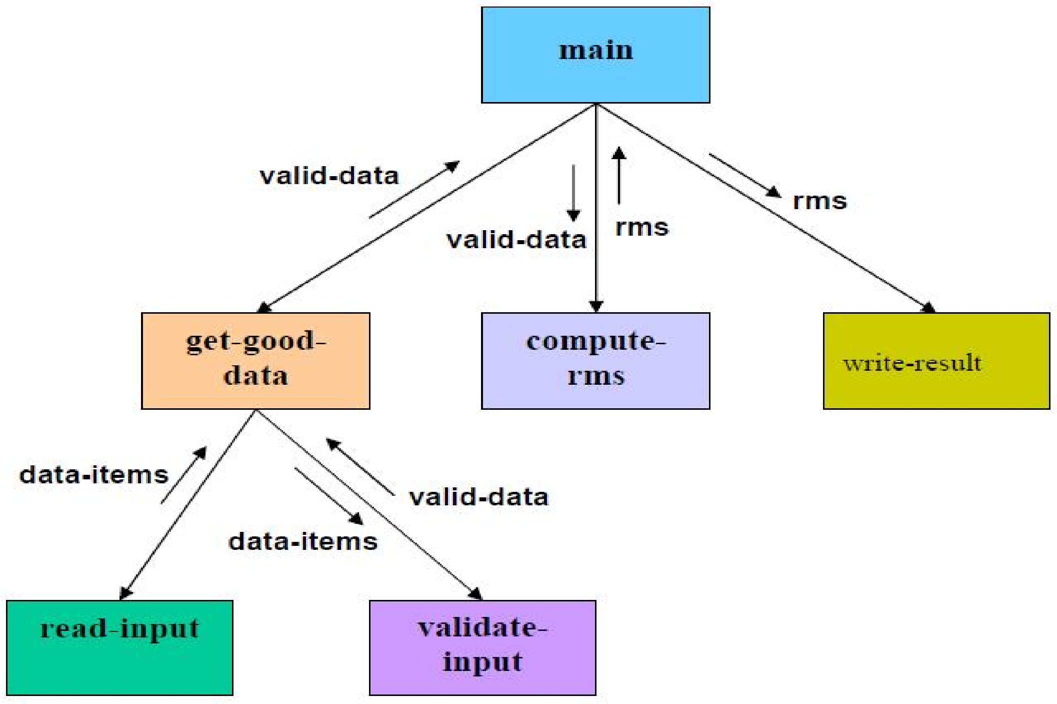
3. Identify the input and output data elements

- After factoring the DFD, the next step is to identify the input and output elements in the DFD.
- An input element in a DFD represents a process or processes which convert input data from physical to logical form. For example reading characters from the terminal and storing them in the internal table list is considered as an input element.
- In the above DFD, Read-numbers is the input element.
- Similarly, an output element in a DFD is one which transforms output data from logical form to physical form. Ex. From list or array into output characters.
- In the above DFD, Display is the output element.
- In addition to input and output elements it is also required to identify the transformation elements in the DFD. For example, in the above DFD Compute-rms is a transformation element.

4. Factoring of input, output, and transform branches

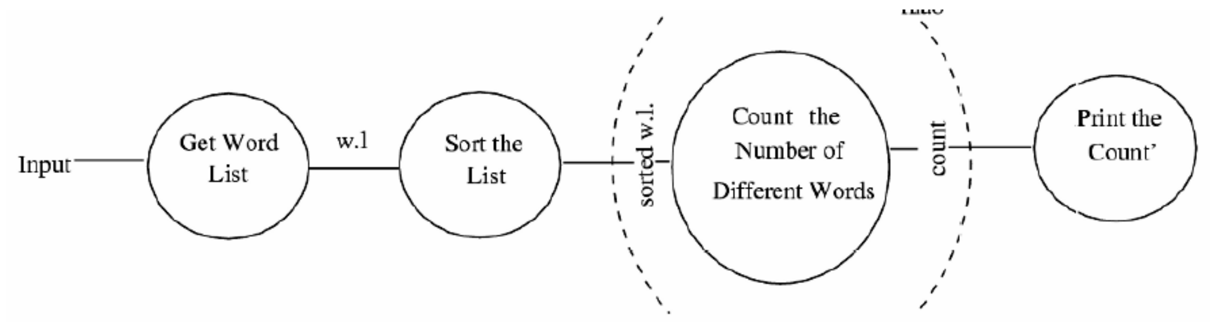
After identifying the input, output and transformation elements in the DFD the next step is to factor them further. That is, decompose the input element into sub input elements and so on.

The output of the structured design methodology is a structure chart. For example, the structure chart for Compute RMS software is shown below.



Example 2

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

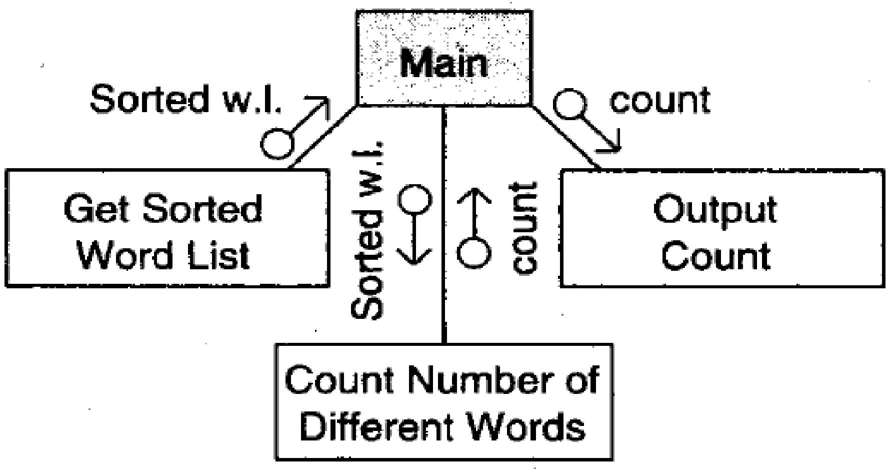


Input element – Get Word List

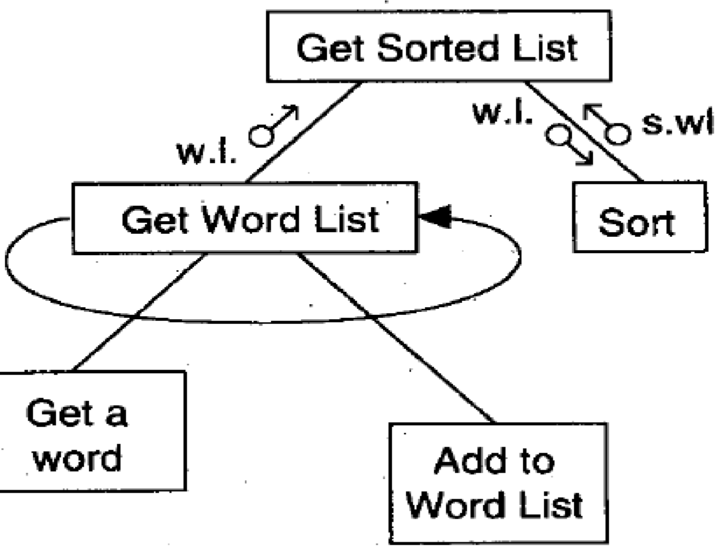
Output element – Print Count

Transformation Element(s) – Sort List and Count Number of Words

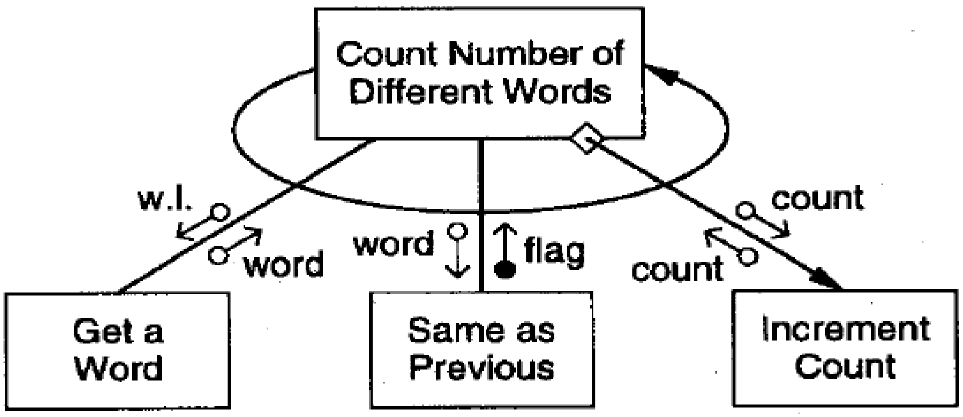
First level factoring



Factoring of Input module



Factoring of Transform module



5.6 Object-Oriented Design

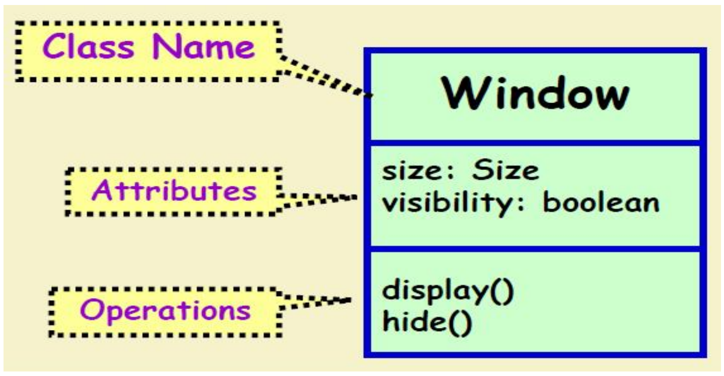
- There are typically two types of designs
    1. Function-oriented design
    2. Object-oriented design

Function-oriented design

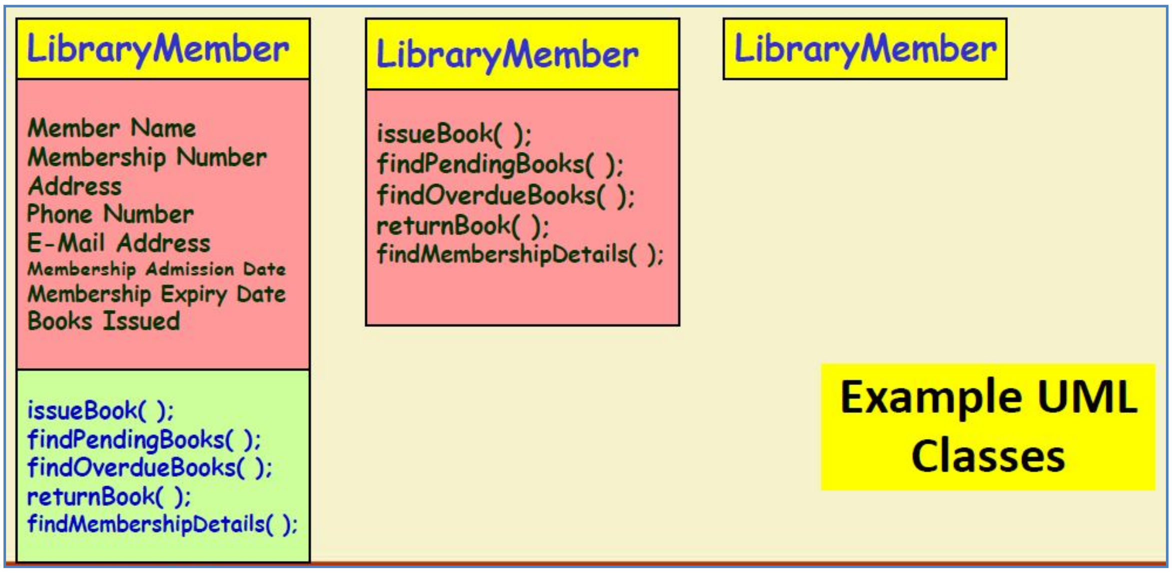
- It is the traditional design.
- In function-oriented design, the system is decomposed into set of functions called modules forming a hierarchical structure of the system.
- Here, the main building block is function or module.
- It follows top-down approach (i.e. the system is decomposed into modules, modules are then decomposed into sub modules, sub modules are then decomposed into sub-sub modules and son).
- Each module can be coded in a procedural language like C.

Object-oriented design

- It is the modern style of design.
- In object-oriented design, the main building block is an object or a class.
- It follows bottom-up approach (i.e. we start with the objects, objects are then combined into components, components are then combined to form the entire system).
- In this, the system is coded in an object oriented language like Java.
- An object is an entity or thing in the real world. For example, in Library Management system we could find set of objects such as Student, Book, Professor etc.
- A class is a set of objects that share the same attributes and operations.
- Object oriented design is best expressed in UML (Unified Modelling Language).
- In UML, a class is represented as a rectangle with three compartments as shown below.



- The following figure shows alternative representations of a class.



- Object-oriented design is concerned with finding and describing the software objects and how they collaborate to full fill the requirements.
- For example, in Library Management system the *Student* object collaborates with the *Book* object to full fill the requirement *Issue book*.

Example 1

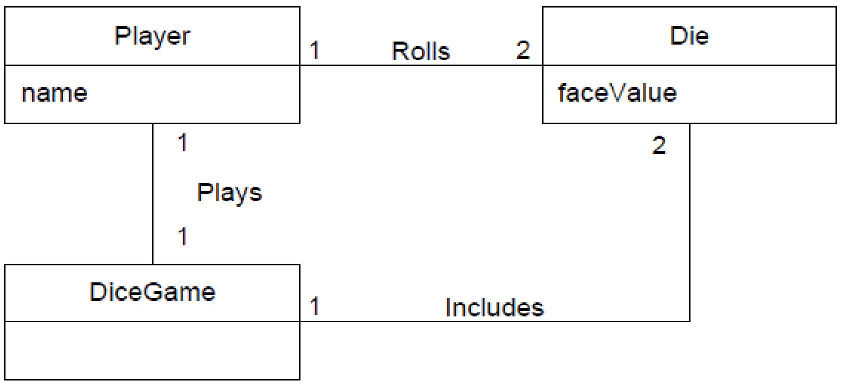
Let us we discuss the object-oriented design using a simple example - Dice game example in which a player picks up and rolls the dice. If the dice face value total seven, they win; otherwise, they lose.

After investigating this problem, we can find the following classes to solve the problem.

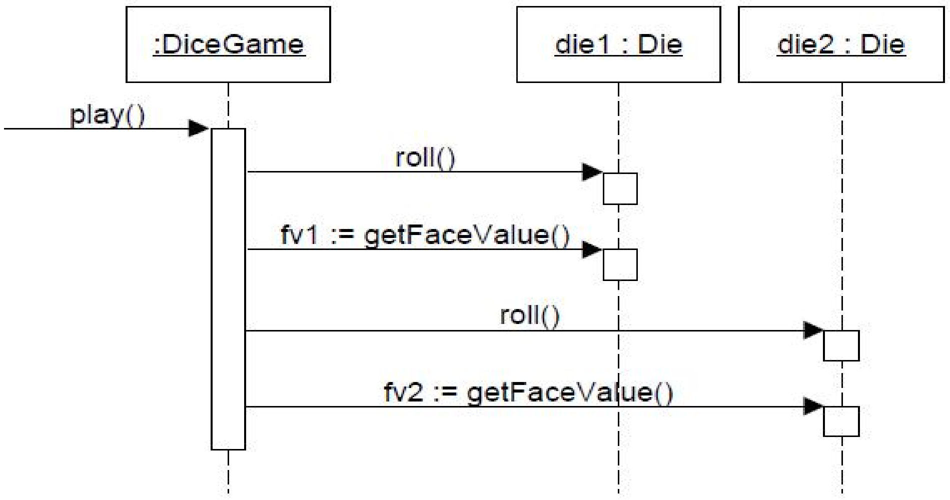
- o Player

- o DiceGame and Die

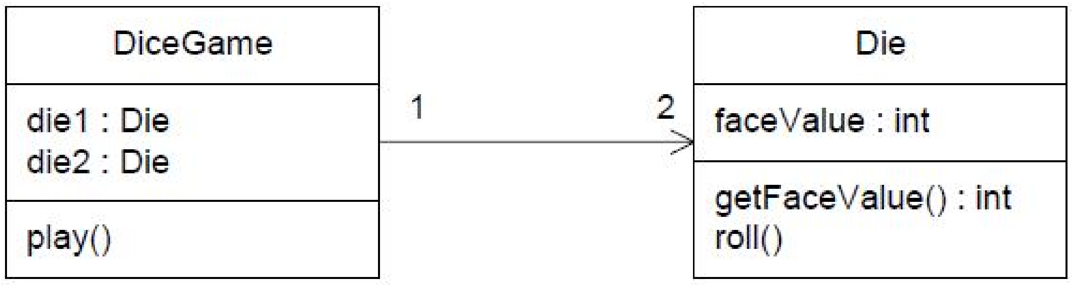
With these classes, now we can able to create the initial class diagram as shown below.



In order to find the operations of these classes, we need to create the interaction diagram for the Dice Game. The interaction diagram is shown below.



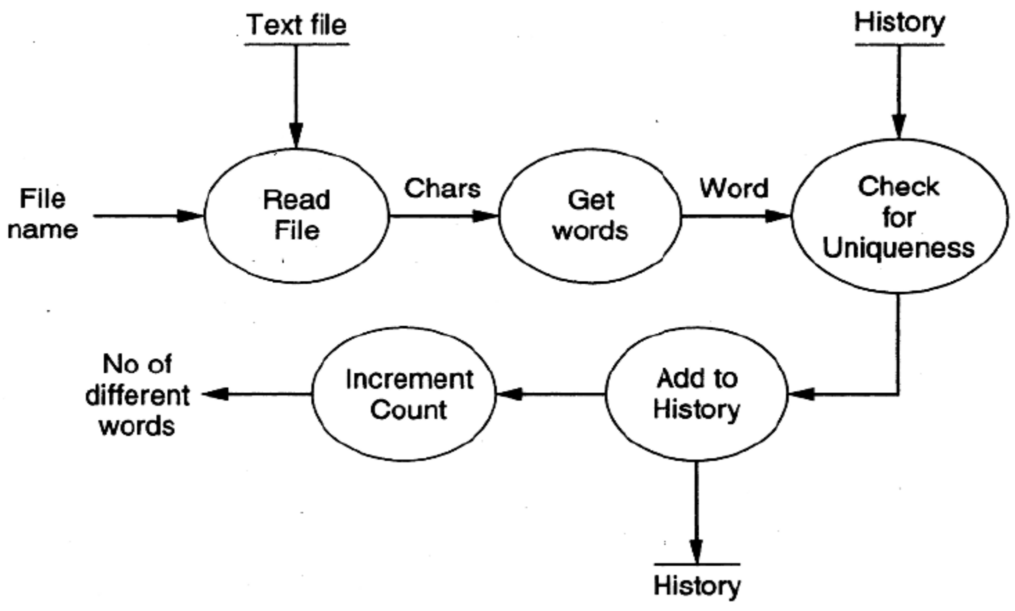
Notice that although in the real world a player rolls the dice, in the software design the DiceGame object "rolls" the dice. An inspection of the interaction diagram leads to the final class diagram shown below. Since a *play* message is sent to a *DiceGame* object, the *DiceGame* class requires a *play* method, while class *Die* requires a *roll* and *getFaceValue* method.



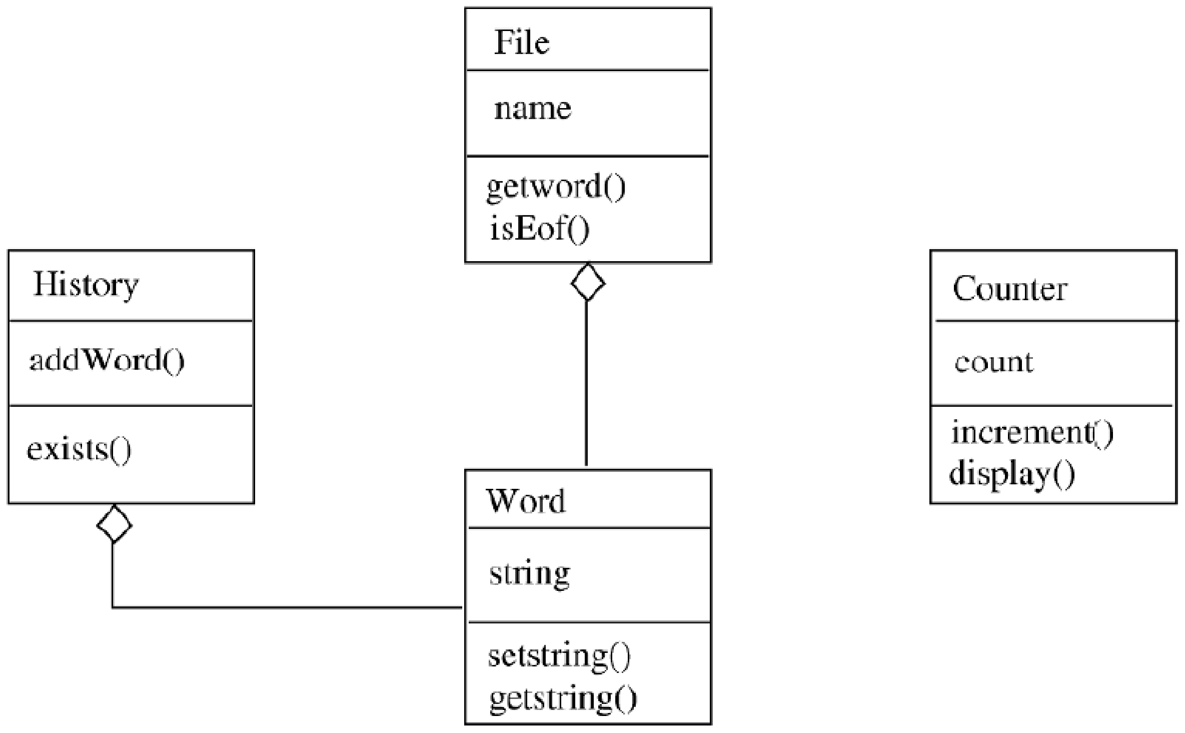
Example 2

The word counting problem - determine the number of different words in an input file.

The functional model of the problem is shown below.



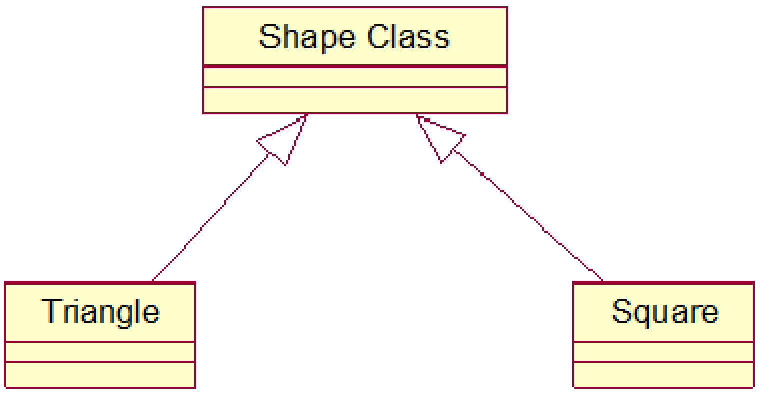
The class diagram for this problem is shown below.


File
name
getword()
isEof()
History
addWord()
exists()
Counter
count
increment()
display()
Word
string
setstring()
getstring()

UNIT V
Assignment-Cum-Tutorial Questions

Objective Questions

1) A component model defines standards for [ ]
   a) Properties b) Methods
   c) Mechanisms d) All of the mentioned
2) What makes a good architecture? [ ]
   a) The architecture may not be the product of a single architect or a small group
   b) The architect should have the technical requirements for the system and an articulated and prioritized list of qualitative properties
   c) The architecture may not be well documented
   d) All of the mentioned
3) To capture and access data from the store by various components we use [ ]
   a) Component Connector Structure
   b) Work-Allocation of Modules
   c) Component and Hardware Dependency
   d) Module Dependency Structure
4) Identify the architectural style which is most frequently used as web system backend [ ]
   a) Client Server Architectural Style
   b) Shared Data Style
   c) Peer to-Peer Style / Object Oriented Style
   d) Publish-Subscribe Style
5) Select the architectural style which is used for Events like Mouse Clicking, mouse drag and database events etc [ ]
   a) Peer-to-Peer Style
   b) Client server Style
   c) shared Data style
   d) Publish-Subscribe Style
6) Which of the following can be considered regarding client and server? [ ]
   a) Client and Server is an architectural style
   b) Client and Server may be considered as an architectural style
   c) Client and Server is not an architectural style
   d) None of the mentioned
7) Choose the option that does not define Function Oriented Software Design [ ]
   a) It consists of module definitions
   b) Modules represent data abstraction
   c) Modules support functional abstraction
   d) None of the above
8) What type of relationship is represented by Shape class and Square ? [ ]



a) Realization b) Generalization
c) Aggregation d) Dependency

9) Which diagram in UML shows a complete or partial view of the structure of a modelled system at a specific time? [ ]

a) Sequence Diagram b) Collaboration Diagram
c) Class Diagram d) Object Diagram

10) Which design defines the logical structure of each module and their interfaces that is used to communicate with other modules. [ ]

a) High level design b) Architectural Design
c) Detailed design d) All mentioned above

SECTION-B

Descriptive Questions

1) What is a Software Architecture? Explain important uses of software architecture?
2) Write about the Architecture views in detail.
3) Briefly explain about Architecture Styles in detail.
4) Identify first level factoring activities for design methodology and apply that for ATM.
5) Differentiate the Component and Connector views.
6) Illustrate architecture diagram for Student Survey System.
7) Apply the suitable style for course scheduling.
8) Illustrate the Authentication and cache management in the Architecture of survey system.
9) Illustrate structure chart of the sort program for
   a. Representation of different types of Modules.
   b. Iteration and decision representation.

10) What are the metrics that can be used to study complexity of an object-oriented design.
11) Draw DFDs for
    a. ATM and
    b. Word Count problem.

C) Previous GATE/UGC NET Questions:

1) ____________________________ of a system is the structure or structures of the system which comprise software elements, the externally visible properties of these elements and the relationship amongst them. **[UGC NET JUNE 2013]**
    a) Software construction
    b) Software evolution
    c) **Software architecture**
    d) Software reuse

*****