Name :- Pranov Mishra Registration No:- 12114762
Roll no:- RDOCO9A55 Assignment : 2 Course code :- CAP437

Teacher Name:- Dr. Yasir Iqbal Course Title :- Software Engineering

Ques 1:- What do you understand by cohesion. Discuss the type of cohesion in detail.

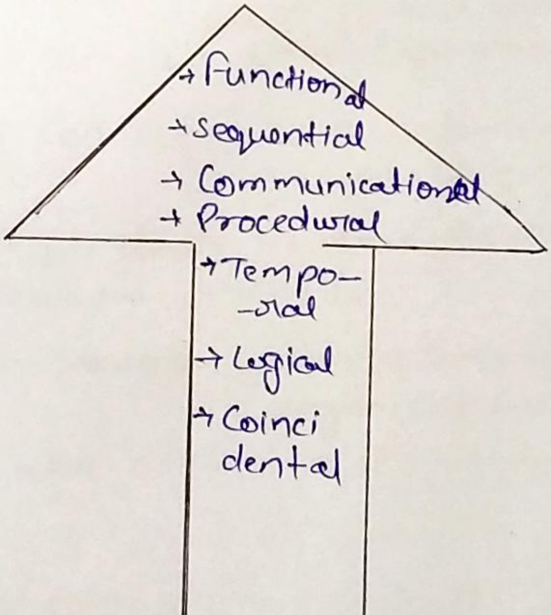
Ans :- Cohesion :- Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

"The measure of the strength of functional relatedness of elements with in a module."

Cohesion refers to the dependence with in and among a module's internal elements.

Cohesion is a natural extension of the informationg hiding concept.

Range of Cohesion :-



- → Functional Cohesion
- → Sequential Cohesion
- → Communicational Cohesion
- → Procedural Cohesion
- → Temporal cohesion
- → Logical cohesion
- → Coincidental Cohesion

There are seven types of cohesion are given below :-

Functional Cohesion :- It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well defined function. It can also be reused.

Sequential Cohesion: When elements of module are grouped because the output of ~~the~~ one element serves as input to another and so on, it is called sequential cohesion.

Communicational Cohesion :- When elements of module are grouped together, which are executed sequentially and work on same data it is called communicational cohesion.

Procedural Cohesion :- When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.

Temporal Cohesion :- When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.

Logical Cohesion :- When logically categorized elements are put together into a module, it is called logical cohesion.

Co-incidental Cohesion :- It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not accepted.

Ques: 2:- Explain class diagram. List and explain all the notations used in class diagram.

Ans:- Class diagram :- A class diagram in the Unified Modeling language (UML) is a type of static structure diagram that describes the structure of system by showing the system's classes, their attributes, operation (or methods) and the relationship among objects. Classes are arranged in hierarchies sharing common structure and behavior and are associated with other classes.

A class consits of its objects, and also it may inherit from other class. A class diagram is used for visualize, describe, document various different aspects of the system, and also construct executable software code.

It is a collection of classes, interface, associations, Collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams:-

→ Analysis and design of the static view of an application

→ Describe responsibilities of a system.

→ Base for component and deployment diagrams.

→ Forward and reverse engineering.

Benefits of Class Diagram:-

→ It can represent the object model for complex system.

→ It provides a general schematic of an application for better understanding.

→ It is helpful for the stake holders and the developers.

Usage of Class diagrams:-

→ To describe the static view of a system.

→ To show the collaboration among every intance in the static view.

→ To describe the functionalities performed by the system.

→ To construct the software application using object-oriented languages.

Notations used in class diagram are following:-

Classes:- Classes represent an abstraction of entities with common characteristics. Associations represent the relationship between classes.

Active Class:- Active classes initiate and control the flow of activity, while passive classes store data and serve other class es.

Visibility:- Use visibility markers to signity who can access the information contained within a class.

Associations:- Associations represent static relationship between class. Place association names above, on or below the associatio line.

Multiplicity (Cardinality):-

Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class.

Constraint:-

Place constraints inside curly braces {}.

Composition and Aggregation:-

Composition is a special type of aggregation that denotes a strong ownership between class A, the whole, and Class B, its part.

Generalization:-

Generalization is another name for inheritence or an "is a" relationship. It refers to a relationship between two class where one class is a specialized version of another.

A class notation consists of three part:-

Class Name:- The name of the class appears in the first partition.

Class Attribute:- Attribute are shown in the second partition.

→ The attribute type is shown after the colon.

→ The Attributes map onto member variables in code.

Class Operations (Methods):-

→ Operations are shown in the third partition.

→ Operations map onto class methods in code.

Ques:-3:- What are the characteristics of good programming language. Explain all the characteristics briefly:-

Ans:- Characteristics of good programming language:-

In computer science are some popular high-level programming language, while there are others that could not become so popular in-spite of being very power

There might be reasons for the success of a language but one obvious reasons is its characteristics.

A good programming language must be simple and easy to learn and use. It should provide a programmer with a clearn, simple and unified set of concepts that can be grasped easily. The overall simplicity of a this strongly affects the readability of the programs written in that language and programs that are easier to read and understand are easier to maintain.

It is also easy to develop and impliment a compiler or an interpreter for a simple language.

However, the power needed for the language should not be sacrified for simplicity.

All the characteristics are the following:-

Naturalness:- A good language should be natural for the application area for which it is designed. That is, it should provide appropriate operators, data structure, control structures and a natural syntax to facilitate programmers to code their problem easily and efficiently.

FORTRAN and COBOL are good examples of language possessing high degree of naturalness in scientific and business application areas, respectively.

Abstraction :- Abstraction means ability to define and them use complicated structures or operation is ways that allows many of the details to be ignored. The degree of abstraction allowed by a language directly affects its ease of programming.

For example :- Object oriented language support high degree of abstraction.

Efficiency :- Programs written in a good language are translated into machine code efficiently, are executed and require relatively less space in memory. A good programming language is supported with a good language language translator (a compiler or an Interpreter) that gives due consideration to space and time efficiency

Structured Programming Support :-

A good language should have necessary features to allow programmers to write their programs based on the concepts of structured programming. This property greatly affects the ease with which a program may be written, tested and maintained.

Compactness :-

In a good programming language should be able to express the intended operations concisely ~~with~~ with out loosing readability. Programmers generally do not like a verbose language because they need to write to much.

Locality :- A good programming language should be such that while writing a program, a programmer need not jump around the visually as the text of a program is prepare This allow the programmer to concentrate almost sobely on the part of the program around the statement currently being worked with.

Extensibility :- A good language should also allow extensions through a simply, natural and elegant mechanism. Almost all languages provide sub program definition mechanisms for the purpose, but some language are weak in this aspect.

Suitability to its Environment :-

Depending upon the type of application for which a programming language has been designed, the language must also be made suitable to its environments.