1. Explain and compare cohesion and coupling.

The design of a system is correct if a system built precisely according to the design satisfies the requirements of that system. The goal during the design phase is to produce correct designs. A system is considered modular if it consists of discrete modules so that each module can be implemented separately, and a change to one module has minimal impact on other modules. For modularity, each module needs to support a well-defined abstraction and have a clear interface through which it can interact with other modules.

Three modularization criteria are:

1. Coupling

2. Cohesion

Coupling:

Coupling between modules is the strength of interconnections between modules or a measure of interdependence among modules. “Highly coupled” modules are joined by strong interconnections, while “loosely coupled” modules have weak interconnections. Two modules are considered independent if one can function completely without the presence of the other. Independent modules have no interconnections. To solve and modify a module separately, we would like the module to be loosely coupled with other modules.

Three factors affecting coupling are:

1. No.of interfaces per module – As no.of interconnections increases coupling increases.

2. Complexity of each interface - The more complex each interface is, the higher will be the degree of coupling.

3. 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. In general, interfaces

with only data communication result in the lowest degree of coupling, followed by interfaces that only transfer control data. Coupling is considered highest if the data is hybrid.

In OO systems, three different types of coupling exist between modules :

• Interaction coupling

• Component coupling

• Inheritance coupling

Interaction coupling occurs due to methods of a class invoking methods of other classes. coupling is lower if only data is passed, but is higher if control information is passed since the invoked method impacts the execution sequence in the calling method. Also, coupling is higher if the amount of data being passed is increased.

Component coupling refers to the interaction between two classes where a class has variables of the other class. Three clear situations exist as to how this can happen. A class C can be component coupled with another class C1, if C has an instance variable of type C1, or C has a method whose parameter is of type C1, or if C has a method which has a local variable of type C1. Component coupling is considered to be weakest (i.e. most desired) if in a class C, the variables of class C1 are either in the signatures of the methods of C, or are some attributes of C. If interaction is through local variables, then this interaction is not visible from outside, and therefore increases coupling.

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.

Cohesion

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 is. There are several levels of cohesion:

• Coincidental

• Logical

• Temporal

• Procedural

• Communicational

• Sequential

• Functional

Coincidental is the lowest level, and functional is the highest. Coincidental cohesion occurs when there is no meaningful relationship among the elements of a module.

Coincidental cohesion can occur if an existing program is “modularized” by chopping it into pieces and making different pieces modules.

A module has logical cohesion if there is some logical relationship between the elements of a module, and the elements perform functions that fall in the same logical class. A typical example of this kind of cohesion is a module that performs all the inputs or all the outputs.

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,” and “termination” are usually temporally bound.

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 separate module.

A module with communicational cohesion has elements that are related by a reference to the same input or output data. That is, in a communicationally bound module, the elements are together because they operate on the same input or output data. An example of this could be a module to “print and punch record.”

When the elements are together in a module because the output of one forms the input to another, we get sequential cohesion.

Functional cohesion is the strongest cohesion. In a functionally bound module, all the elements of the module are related to performing a single function.

Cohesion in object-oriented systems has three aspects :

- Method cohesion
- Class cohesion
- Inheritance cohesion

Method cohesion is the same as cohesion in functional modules. It focuses on why the different code elements of a method are together within the method.

Class cohesion focuses on why different attributes and methods are together in this class. Inheritance cohesion focuses on the reason why classes are together in a hierarchy. The two main reasons for inheritance are to model generalization specialization relationship, and for code reuse.

2. Explain the requirements engineering process with a neat sketch

requirements engineering The process of establishing what services are required and the constraints on the system's operation and development.

Requirements engineering process

Feasibility study

Checking whether it is technically and financially feasible to build the system?

Requirements elicitation and analysis

What do the system stakeholders require or expect from the system?

Requirements specification

Defining the requirements in detail

Requirements validation

Checking the validity of the requirements

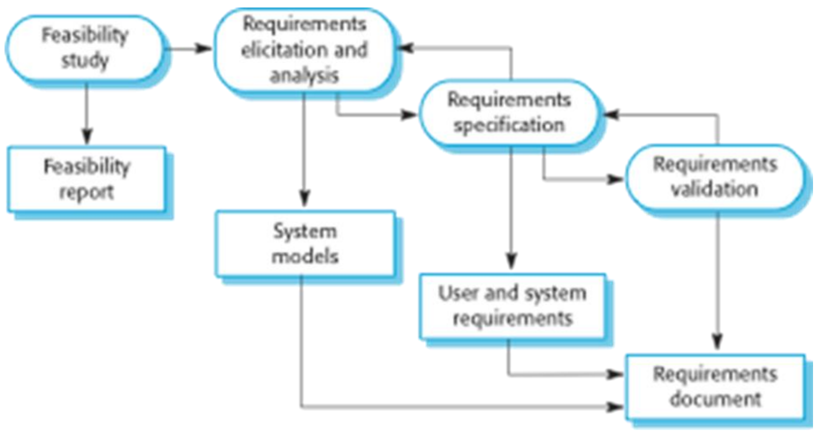
Requirements engineering is a particularly critical stage of the software process as errors at this stage inevitably lead to later problems in the system design and implementation.

The requirements engineering process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements.

Requirements are usually presented at two levels of detail.

User requirements:-End-users and customers need a high-level statement of the requirements;

System Requirements:-system developers need a more detailed system specification.



1. Explain the requirements engineering process with a neat sketch

requirements engineering The process of establishing what services are required and the constraints on the system's operation and development.

Requirements engineering process

Feasibility study

Checking whether it is technically and financially feasible to build the system?

Requirements elicitation and analysis

What do the system stakeholders require or expect from the system?

Requirements specification

Defining the requirements in detail

Requirements validation

Checking the validity of the requirements

Requirements engineering is a particularly critical stage of the software process as errors at this stage inevitably lead to later problems in the system design and implementation.

The requirements engineering process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements.

Requirements are usually presented at two levels of detail.

User requirements:-End-users and customers need a high-level statement of the requirements;

System Requirements:-system developers need a more detailed system specification.

1. Feasibility study

An estimate is made of whether the identified user needs may be

satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis.

2. Requirements elicitation and analysis

This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

3. Requirements specification

is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; system requirements are a more detailed description of the functionality to be provided.

4. Requirements validation

This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document

are inevitably discovered. It must then be modified to correct these problems.