Cohesion and coupling

Cohesion and coupling are two factors that measures functional interdependence of a module.

Coupling is the measure of the degree of interdependence between the modules. Good software will have **low coupling**.

Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have **high cohesion**.

There are seven types of cohesions we will discuss them one by one:

1. **Coincidental cohesion (worst)**

   Coincidental cohesion is when parts of a module are grouped arbitrarily; the only relationship between the parts is that they have been grouped together. Elements contribute to activities with no meaningful relationship to one another. Similar to logical cohesion, except the activities may not even be the same type.

   Difficult to understand and maintain, with strong possibilities of causing 'side effects' every time the module is modified.

   Examples:

   - Utility Class: This class contains a collection of public and static functions and they are accessible from different classes. Changes in Utility class will effect on other classes that are accessing functions of the utility class.
   - Transaction Processing System: In a transaction processing system (TPS), the get-input, print-error, and summarize-members functions are grouped into one module. The grouping does not have any relevance to the structure of the problem.

2. **Logical cohesion**

   Logical cohesion is when parts of a module are grouped because they are logically categorized to do the same thing, even if they are different by nature (e.g. grouping all mouse and keyboard input handling routines).

   Examples:

   - Print Functions: An example of logical cohesion is the case where a set of print functions generating different output reports are arranged into a single module.
   - Input handling routines: Grouping all mouse and keyboard handling routines

3. **Temporal cohesion**

   Temporal cohesion is when parts of a module are grouped by when they are processed – the parts are processed at a particular time in program execution.

   Examples:

   - A function which is called after catching an exception which closes open files, creates an error log, and notifies the user.
   - The set of functions responsible for initialization, start-up, shutdown of some process, etc. exhibit temporal cohesion.

4. **Procedural cohesion**

   Procedural cohesion is when parts of a module are grouped because they always follow a certain sequence of execution and commonly found at the top of hierarchy such as the main program. This cohesion is similar to sequential cohesion except that elements are unrelated.

   OR

   A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective.

   Examples:

   - A function which checks file permissions then opens the file.
   - The algorithm for decoding a message.

5. **Communicational/informational cohesion**

   A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure. Elements contribute to activities that use the same input or output data. Not flexible, for example, if we need to focus on some activities and not the others.

   Possible links that cause activities to affect each other. Better to split to functional cohesive ones.

   Examples:

   - The set of functions defined on an array or a stack.
   - Module determines customer details like use customer account no to find and return customer name and loan balance.

6. **Sequential cohesion**

   Sequential cohesion is when parts of a module are grouped because the output from one part is the input to another part like an assembly line (e.g. a function which reads data from a file and processes the data).

   Usually has good coupling and is easily maintained. Not so readily reusable because activities that will not in general be useful together.

   Examples:

   - Module format and cross-validate record like use and format raw record and cross- validate in raw record. Then return formatted cross-validated record.
   - In a TPS, the get-input, validate-input, sort-input functions are grouped into one module.

7. **Functional cohesion (best)**

   Functional cohesion is when parts of a module are grouped because they all contribute to a single well-defined task of the module (e.g. lexical analysis of an XML string).Focused (strong, single minded purpose) and no element doing unrelated activities

   Examples:

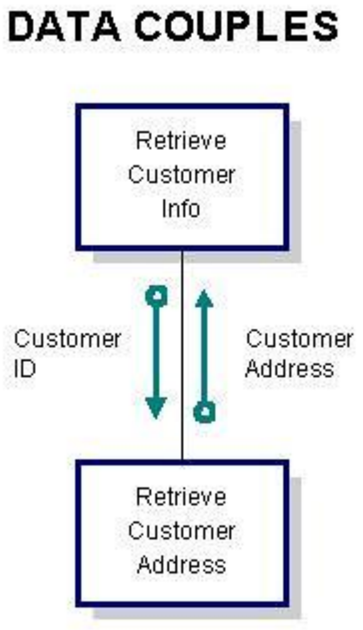
   - Lexical analysis of an XML
   - Compute cosine of angle
   - Read transaction record
   - Assign seat to airline passenger

Below are the different levels of coupling listed from best (data coupling) to worst (content coupling).

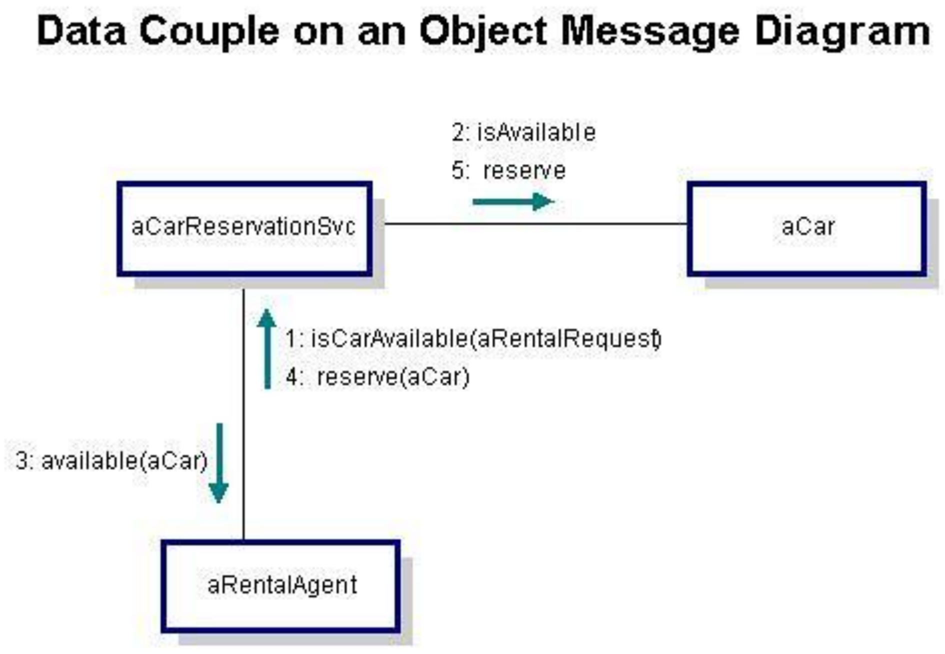
DATA COUPLING

Data coupling occurs between two modules when data are passed by parameters using a simple argument list and every item in the list is used.

An example of data coupling is illustrated as a module which retrieves customer address using customer id.



An example of object-oriented data coupling is shown in the object message diagram. It illustrates messages sent between a Rental Agent object and a CarReservationService object to reserve a Car (the Car object is an argument passed in the messages shown in the object message diagram).



Use data coupling when the argument list is small. As a rule of thumb, limit the argument list to three items.

Strengths of data coupling are:

- A module sees only the data elements it requires.
- It is the best (i.e., loosest) form of coupling.

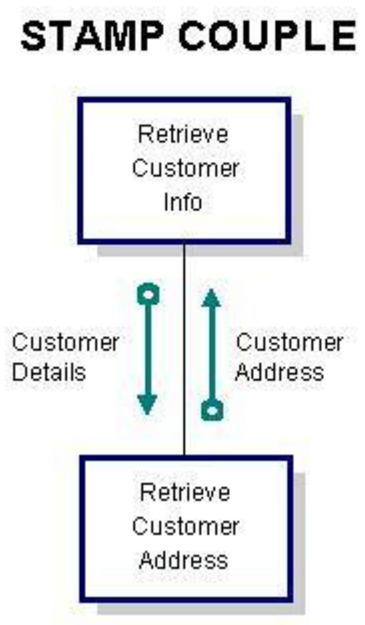
Weakness of data coupling is:

- A module can be difficult to maintain if many data elements are passed. Too many parameters can also indicate that a module has been poorly partitioned.

STAMP COUPLING

Stamp coupling occurs between modules when data are passed by parameters using a data structure containing fields which may or may not be used.

An example of stamp coupling illustrates a module that retrieves customer address using only customer id which is extracted from a parameter named customer details.



As a rule of thumb, never pass a data structure containing many fields to a module that only needs a few.

Some of the practitioners of structured design do not make a distinction between the passing of parameters in an unstructured format (as described under data coupling) and the passing of parameters in a data structure (stamp coupling). The distinction between data and stamp coupling is not relevant in object-oriented systems.

Strengths of stamp coupling are:

- It is a loose form of coupling.
- It simplifies interfaces between modules.

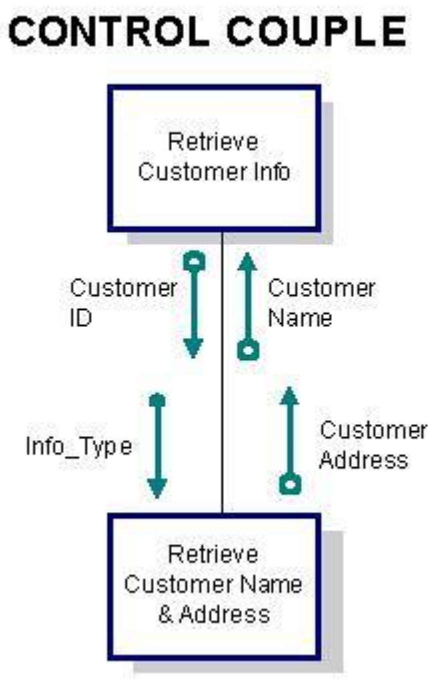
Weaknesses of Stamp Coupling

- It promotes the creation of artificial data structures (i.e., bundling of unrelated data elements together in a structure). Although this bundling is meant to reduce coupling between modules, it in fact increases the coupling between modules. Data structures are appropriate as long as the data bundled together is meaningful and related.

CONTROL COUPLING

Control coupling occurs between modules when data are passed that influence the internal logic of a module (e.g., flags and switches).

In an example of control coupling, a module that retrieves either a customer name or an address depending on the value of a flag is illustrated.



As a rule of thumb, use descriptive flags (i.e., a flag that describes a situation or condition, such as end-of-file or invalid-acct-num). Do not use control flags (i.e., a flag that tells a module what to do, such as write-err-message).

Object-Oriented Considerations

In object-oriented systems, control coupling occurs when an object receives a message from another object and the receiving object responds to the message differently based on control information contained in the message. Another type of control coupling is evident when a method returns control information to an object in responding to a message.

An example of object-oriented control coupling is a method that returns an error code with a value of zero when no errors are encountered handling a message.

Strength of Control Coupling

- It is a moderate form of coupling.

Weaknesses of Control Coupling

- Control flags passed down the hierarchy require the calling program to know about the internals of the called program (i.e., the called program is not a black box).
- Control flags passed up the hierarchy cause a subordinate module to influence the flow of control in the parent module (i.e., the subordinate module has an affect on a module it does not control).

COMMON COUPLING

Common coupling occurs when modules communicate using global data areas (i.e., universal common data areas). For example, C allows the developer to declare a data element as external, enabling it to be accessed by all modules.

Common coupling is also known as global coupling.

An example of a module using common coupling is an IBM environment using CICS as a teleprocessing monitor, which has programs that can access and update global variables through the Transaction Work Area (TWA).

Object-Oriented Considerations

In object-oriented systems, common coupling occurs when an object references a specific external object and has knowledge of the structure and implementation details of the external object. Generally one should avoid the use of data in the public interface of an object. However, it is sometimes necessary for objects to include items other than methods in their public interfaces. In these instances, the general guideline is to use constants (i.e., fixed values) rather than variables (i.e., values that vary throughout execution).

An example of object-oriented common coupling is an object whose public interface includes items with values that vary during execution and whose underlying structures and implementation details do not remain hidden.

Weaknesses of Common Coupling

- Modules are very tightly coupled. A fault in one module using global data may show up in another module because global data may be updated by any module at any time.
- Modules referencing global data are tied to specific data names, unlike modules that reference data through parameters. This greatly limits the module's re-usability.

- It may take considerable time for a maintenance programmer to determine what module updated the global data. The more global data used, the more difficult the job becomes to determine how modules obtain their data.
- It is more difficult to track the modules that access global data when a piece of global data is changed (e.g., a field size is changed).

CONTENT COUPLING (Worst)

Content coupling occurs between two modules if one refers to the internals of the other module. For example, if one module branches into another module, if one module refers to or changes data in another module, or if one module alters a statement in another module. In practice, only assembler language allows content coupling. Most procedural and object-oriented programming languages make it difficult to implement content coupling.

Content coupling is also known as pathological coupling.

Reference:

- https://weeklyitblogs.wordpress.com/tag/example-of-cohesion/
- https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/
- https://it.toolbox.com/blogs/craigborysowich/design-principles-coupling-data-and-otherwise-050307