Module 4

Software Design Issues

Lesson 9

An Overview of Current Design Approaches

Specific Instructional Objectives

At the end of this lesson the student will be able to:

- State what cohesion means.
- Classify the different types of cohesion that a module may possess.
- State what coupling means.
- Classify the different types of coupling between modules.
- State when a module can be called functionally independent of other modules.
- State why functional independence is the key factor for a good software design.
- State the salient features of a function-oriented design approach.
- State the salient features of an object-oriented design approach.
- Differentiate between function-oriented and object-oriented design approach.

Cohesion

Most researchers and engineers agree that a good software design implies clean decomposition of the problem into modules, and the neat arrangement of these modules in a hierarchy. The primary characteristics of neat module decomposition are high cohesion and low coupling. Cohesion is a measure of functional strength of a module. A module having high cohesion and low coupling is said to be functionally independent of other modules. By the term functional independence, we mean that a cohesive module performs a single task or function. A functionally independent module has minimal interaction with other modules.

Classification of cohesion

The different classes of cohesion that a module may possess are depicted in fig. 4.1.

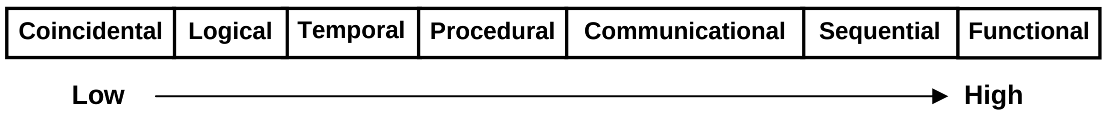

Fig. 4.1: Classification of cohesion

Coincidental cohesion: A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely, if at all. In this case, the module contains a random collection of functions. It is likely that the functions have been put in the module out of pure coincidence without any thought or design. For example, 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.

Logical cohesion: A module is said to be logically cohesive, if all elements of the module perform similar operations, e.g. error handling, data input, data output, etc. An example of logical cohesion is the case where a set of print functions generating different output reports are arranged into a single module.

Temporal cohesion: When a module contains functions that are related by the fact that all the functions must be executed in the same time span, the module is said to exhibit temporal cohesion. The set of functions responsible for initialization, start-up, shutdown of some process, etc. exhibit temporal cohesion.

Procedural cohesion: 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, e.g. the algorithm for decoding a message.

Communicational cohesion: A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, e.g. the set of functions defined on an array or a stack.

Sequential cohesion: A module is said to possess sequential cohesion, if the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next. For example, in a TPS, the get-input, validate-input, sort-input functions are grouped into one module.

Functional cohesion: Functional cohesion is said to exist, if different elements of a module cooperate to achieve a single function. For example, a module containing all the functions required to manage employees' pay-roll exhibits functional cohesion. Suppose a module exhibits functional cohesion and we are asked to describe what the module does, then we would be able to describe it using a single sentence.

Coupling

Coupling between two modules is a measure of the degree of interdependence or interaction between the two modules. A module having high cohesion and low coupling is said to be functionally independent of other modules. If two modules interchange large amounts of data, then they are highly interdependent. The degree of coupling between two modules depends on their interface complexity.

The interface complexity is basically determined by the number of types of parameters that are interchanged while invoking the functions of the module.

Classification of Coupling

Even if there are no techniques to precisely and quantitatively estimate the coupling between two modules, classification of the different types of coupling will help to quantitatively estimate the degree of coupling between two modules. Five types of coupling can occur between any two modules. This is shown in fig. 4.2.

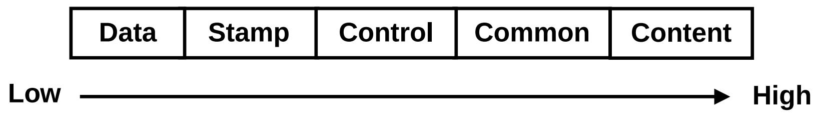

Fig. 4.2: Classification of coupling

Data coupling: Two modules are data coupled, if they communicate through a parameter. An example is an elementary data item passed as a parameter between two modules, e.g. an integer, a float, a character, etc. This data item should be problem related and not used for the control purpose.

Stamp coupling: Two modules are stamp coupled, if they communicate using a composite data item such as a record in PASCAL or a structure in C.

Control coupling: Control coupling exists between two modules, if data from one module is used to direct the order of instructions execution in another. An example of control coupling is a flag set in one module and tested in another module.

Common coupling: Two modules are common coupled, if they share data through some global data items.

Content coupling: Content coupling exists between two modules, if they share code, e.g. a branch from one module into another module.

Functional independence

A module having high cohesion and low coupling is said to be functionally independent of other modules. By the term functional independence, we mean that a cohesive module performs a single task or function. A functionally independent module has minimal interaction with other modules.

Need for functional independence

Functional independence is a key to any good design due to the following reasons:

- **Error isolation:** Functional independence reduces error propagation. The reason behind this is that if a module is functionally independent, its degree of interaction with the other modules is less. Therefore, any error existing in a module would not directly effect the other modules.

- **Scope of reuse:** Reuse of a module becomes possible. Because each module does some well-defined and precise function, and the interaction of the module with the other modules is simple and minimal. Therefore, a cohesive module can be easily taken out and reused in a different program.

- **Understandability:** Complexity of the design is reduced, because different modules can be understood in isolation as modules are more or less independent of each other.

Function-oriented design

The following are the salient features of a typical function-oriented design approach:

1. A system is viewed as something that performs a set of functions. Starting at this high-level view of the system, each function is successively refined into more detailed functions. For example, consider a function create-new-library-member which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge. This function may consist of the following sub-functions:

    - assign-membership-number
    - create-member-record
    - print-bill

Each of these sub-functions may be split into more detailed subfunctions and so on.

2. The system state is centralized and shared among different functions, e.g. data such as member-records is available for reference and updation to several functions such as:

- create-new-member
- delete-member
- update-member-record

Object-oriented design

In the object-oriented design approach, the system is viewed as collection of objects (i.e. entities). The state is decentralized among the objects and each object manages its own state information. For example, in a Library Automation Software, each library member may be a separate object with its own data and functions to operate on these data. In fact, the functions defined for one object cannot refer or change data of other objects. Objects have their own internal data which define their state. Similar objects constitute a class. In other words, each object is a member of some class. Classes may inherit features from super class. Conceptually, objects communicate by message passing.

Function-oriented vs. object-oriented design approach

The following are some of the important differences between function-oriented and object-oriented design.

- Unlike function-oriented design methods, in OOD, the basic abstraction are not real-world functions such as sort, display, track, etc, but real-world entities such as employee, picture, machine, radar system, etc. For example in OOD, an employee pay-roll software is not developed by designing functions such as update-employee-record, get-employee-address, etc. but by designing objects such as employees, departments, etc. Grady Booch sums up this difference as “identify verbs if you are after procedural design and nouns if you are after object-oriented design”

- In OOD, state information is not represented in a centralized shared memory but is distributed among the objects of the system. For example, while developing an employee pay-roll system, the employee data such as the names of the employees, their code numbers, basic salaries, etc. are usually implemented as global data in a traditional programming system; whereas in an object-oriented system these data are distributed among different employee objects of the system. Objects communicate by message passing. Therefore, one object may discover the state information of another object by interrogating it. Of course, somewhere or other the real-world functions must be implemented. In OOD, the functions are usually associated with specific real-world entities (objects); they directly access only part of the system state information.

- Function-oriented techniques such as SA/SD group functions together if, as a group, they constitute a higher-level function. On the other hand, object-oriented techniques group functions together on the basis of the data they operate on.

To illustrate the differences between the object-oriented and the function-oriented design approaches, an example can be considered.

Example: Fire-Alarm System

The owner of a large multi-stored building wants to have a computerized fire alarm system for his building. Smoke detectors and fire alarms would be placed in each room of the building. The fire alarm system would monitor the status of these smoke detectors. Whenever a fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire condition has occurred and then sound the alarms only in the neighboring locations. The fire alarm system should also flash an alarm message on the computer console. Fire fighting personnel man the console round the clock. After a fire condition has been successfully handled, the fire alarm system should support resetting the alarms by the fire fighting personnel.

Function-Oriented Approach:

```
/* Global data (system state) accessible by various
functions */

BOOL detector_status[MAX_ROOMS];
int detector_locs[MAX_ROOMS];
BOOL alarm_status[MAX_ROOMS];
/* alarm activated when status is set */
int alarm_locs[MAX_ROOMS];
/* room number where alarm is located */
int neighbor-alarm[MAX_ROOMS][10];
/* each detector has at most 10 neighboring locations
*/
```

The functions which operate on the system state are:

```
interrogate_detectors();
get_detector_location();
determine_neighbor();
ring_alarm();
reset_alarm();
report_fire_location();
```

Object-Oriented Approach:

```
class detector
attributes:

     status, location, neighbors

operations:

     create, sense_status, get_location,
     find_neighbors

class alarm
attributes:

     location, status

operations:

     create, ring_alarm, get_location,
     reset_alarm
```

In the object oriented program, an appropriate number of instances of the class detector and alarm should be created. If the function-oriented and the object-oriented programs are examined, it can be seen that in the function-oriented program, the system state is centralized and several functions accessing this central data are defined. In case of the object-oriented program, the state information is distributed among various sensor and alarm objects.

It is not necessary an object-oriented design be implemented by using an object-oriented language only. However, an object-oriented language such as C++ supports the definition of all the basic mechanisms of class, inheritance, objects, methods, etc. and also support all key object-oriented concepts that we have just discussed. Thus, an object-oriented language facilitates the implementation of an OOD. However, an OOD can as well be implemented using a conventional procedural language – though it may require more effort to implement an OOD using a procedural language as compared to the effort required for implementing the same design using an object-oriented language.

Even though object-oriented and function-oriented approaches are remarkably different approaches to software design, yet they do not replace each other but complement each other in some sense. For example, usually one applies the top-down function-oriented techniques to design the internal methods of a class, once the classes are identified. In this case, though outwardly the system appears to have been developed in an object-oriented fashion, but inside each class there may be a small hierarchy of functions designed in a top-down manner.

The following questions have been designed to test the objectives identified for this module:

1. Identify at least five important items developed during software design phase.

Ans.: - For a design to be easily implementable in a conventional programming language, the following items must be designed during this phase.

- Different modules required to implement the design solution.
- Control relationship among the identified modules. The relationship is also known as the call relationship or invocation relationship among modules.
- Interface among different modules. The interface among different modules identifies the exact data items exchanged among the modules.
- Data structures of the individual modules.
- Algorithms required to implement the individual modules.

2. State two major design activities.

Ans.: - The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. A good software design is seldom arrived by using a single step procedure but rather through several iterations through a series of steps. Design activities can be broadly classified into two important parts:

- Preliminary (or high-level) design and
- Detailed design.

3. Identify at least two activities undertaken during high-level design.

Ans.: - High-level design means identification of different modules and the control relationships among them and the definition of the interfaces among these modules. The outcome of high-level design is called the program structure or software architecture.

4. Identify at least two activities undertaken during detailed design.

Ans.: - During detailed design, the data structure and the algorithms of the different modules are designed. The outcome of the detailed design stage is usually known as the module-specification document.

5. Identify at least three reasons in favor of why functional independence is the key factor for a good software design.

Ans.: - Functional independence is a key to any good design primarily due to the following reason:

- **Error isolation:** Functional independence reduces error propagation. The reason behind this is that if a module is functionally independent, its degree of interaction with the other modules is less. Therefore, any error existing in a module would not directly effect the other modules.

- **Scope of reuse:** Reuse of a module becomes possible. Because each module does some well-defined and precise function and the interaction of the module with the other modules is simple and minimal. Therefore, a cohesive module can be easily taken out and reused in a different program.

- **Understandability:** Complexity of the design is reduced, because different modules can be understood in isolation as modules are more or less independent of each other.

6. Identify four characteristics of a good software design technique.

Ans.: - A few desirable characteristics that every good software design for general application must possess are as follows:

- **Correctness:** A good design should correctly implement all the functionalities identified in the SRS document.

- **Understandability:** A good design is easily understandable.

- **Efficiency:** It should be efficient.

- **Maintainability:** It should be easily amenable to change.

7. Identify at least two salient features of a function-oriented design approach.

Ans.: - The following are the salient features of a typical function-oriented design approach:

1. A system is viewed as something that performs a set of functions. Starting at this high-level view of the system, each function is successively refined into more detailed functions. For example, consider a function create-new-

library member which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge. This function may consist of the following sub-functions:

- assign-membership-number
- create-member-record
- print-bill

Each of these subfunctions may be split into more detailed subfunctions and so on.

2. The system state is centralized and shared among different functions, e.g. data such as member-records is available for reference and updation to several functions such as:

    - create-new-member
    - delete-member
    - update-member-record

8. Identify at three least salient features of an object-oriented design approach.

Ans.: - In the object-oriented design approach, the system is viewed as collection of objects (i.e. entities). The state is decentralized among the objects and each object manages its own state information. For example, in a Library Automation Software, each library member may be a separate object with its own data and functions to operate on these data. In fact, the functions defined for one object cannot refer or change data of other objects. Objects have their own internal data which define their state. Similar objects constitute a class. In other words, each object is a member of some class. Classes may inherit features from super class. Conceptually, objects communicate by message passing.

9. Write down at least three differences between function-oriented and object-oriented design approach.

Ans.: - The following are some of the important differences between the function-oriented and object-oriented design.

- Unlike function-oriented design methods, in OOD, the basic abstraction are not real-world functions such as sort, display, track, etc, but real-world entities such as employee, picture, machine, radar system, etc. For example in OOD, an employee pay-roll software is not developed by designing functions such as update-employee-record, get-employee-address, etc. but by designing objects such as employees, departments, etc.

- In OOD, state information is not represented in a centralized shared memory but is distributed among the objects of the system. For example, while developing an employee pay-roll system, the employee data such as the names of the employees, their code numbers, basic salaries, etc. are usually implemented as global data in a traditional programming system; whereas in an object-oriented system these data are distributed among different employee objects of the system. Objects communicate by message passing. Therefore, one object may discover the state information of another object by interrogating it. Of course, somewhere or other the real-world functions must be implemented. In OOD, the functions are usually associated with specific real-world entities (objects); they directly access only part of the system state information.

- Function-oriented techniques such as SA/SD group functions together if, as a group, they constitute a higher-level function. On the other hand, object-oriented techniques group functions together on the basis of the data they operate on.

To illustrate the differences between the object-oriented and the function-oriented design approaches, an example can be considered.

Example: Fire-Alarm System

The owner of a large multi-stored building wants to have a computerized fire alarm system for his building. Smoke detectors and fire alarms would be placed in each room of the building. The fire alarm system would monitor the status of these smoke detectors. Whenever a fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire condition has occurred and then sound the alarms only in the neighboring locations. The fire alarm system should also flash an alarm message on the computer consol. Fire fighting personnel man the console round the clock. After a fire condition has been successfully handled, the fire alarm system should support resetting the alarms by the fire fighting personnel.

Function-Oriented Approach:

/* Global data (system state) accessible by various functions */

```
BOOL detector_status[MAX_ROOMS];
int detector_locs[MAX_ROOMS];
BOOL alarm_status[MAX_ROOMS];
/* alarm activated when status is set */
int alarm_locs[MAX_ROOMS];
/* room number where alarm is located */
int neighbor-alarm[MAX_ROOMS][10];
/* each detector has atmost 10 neighboring locations */
```

The functions which operate on the system state are:

```
interrogate_detectors();
get_detector_location();
determine_neighbor();
ring_alarm();
reset_alarm();
report_fire_location();
```

Object-Oriented Approach:

```
class detector

attributes
     status, location, neighbors

operations
     create, sense-status, get-location,
     find-neighbors


class alarm

attributes
     location, status

operations
     create, ring-alarm, get_location, reset-alarm
```

In the object oriented program, an appropriate number of instances of the class detector and alarm should be created. If the function-oriented and the object-oriented programs are examined, it can be seen that in the function-oriented program, the system state is centralized and several functions accessing this central data are defined. In case of the object-oriented program, the state information is distributed among various sensor and alarm objects.

It is not necessary an object-oriented design be implemented by using an object-oriented language only. However, an object-oriented language such as C++ supports the definition of all the basic mechanisms of class, inheritance, objects, methods, etc. and also support all key object-oriented concepts that we have just discussed. Thus, an object-oriented language facilitates the implementation of an OOD. However, an OOD can as well be implemented using a conventional procedural language – though it may require more effort to implement an OOD using a procedural language as compared to the effort required for implementing the same design using an object-oriented language.

Even though object-oriented and function-oriented approaches are remarkably different approaches to software design, yet they do not replace each other but complement each other in some sense. For example, usually one applies the top-down function oriented techniques to design the internal methods of a class, once the classes are identified. In this case, though outwardly the system appears to have been developed in an object-oriented fashion, but inside each class there may be a small hierarchy of functions designed in a top-down manner.

For the following, mark all options which are true.

1. The desirable characteristics that every good software design needs are

- □ Correctness
- □ Understandability
- □ Efficiency
- □ Maintainability
- □ All of the above √

2. A module is said to have logical cohesion, if

- □ it performs a set of tasks that relate to each other very loosely.
- □ all the functions of the module are executed within the same time span.
- □ all elements of the module perform similar operations, e.g. error handling, data input, data output etc. √
- □ None of the above.

3. High coupling among modules makes it

□ difficult to understand and maintain the product
□ difficult to implement and debug
□ expensive to develop the product as the modules having high coupling cannot be developed independently
□ all of the above √

4. Functional independence results in

□ error isolation
□ scope of reuse
□ understandability
□ all of the above √

Mark the following as either True or False. Justify your answer.

1. Coupling between two modules is nothing but a measure of the degree of dependence between them.

Ans.: - False.

Explanation: - Coupling between two modules is a measure of the degree of interdependence or interaction between the two modules.

2. The primary characteristic of a good design is low cohesion and high coupling.

Ans.: - False.

Explanation: - Neat module decomposition of a design problem into modules means that the modules in a software design should display high cohesion and low coupling. Conceptually it means that the modules in a design solution are more or less independent of each other.

3. A module having high cohesion and low coupling is said to be functionally independent of other modules.

Ans.: - True.

Explanation: - By the term functional independence, it is meant that a cohesive module performs a single task or function. A functionally independent module has minimal interaction with other modules.

4. The degree of coupling between two modules does not depend on their interface complexity.

   Ans.: - False.

   Explanation: - The degree of coupling between two modules depends on their interface complexity. The interface complexity is basically determined by the types of parameters that are interchanged while invoking the functions of the module.

5. In the function-oriented design approach, the system state is decentralized and not shared among different functions.

   Ans.: - False.

   Explanation: - In the function-oriented designed approach, the system state is centralized and shared among different functions. On the other hand, in the object-oriented design approach, the system state is decentralized among the objects and each object manages its own state information.

6. The essence of any good function-oriented design technique is to map the functions performing similar activities into a module.

   Ans.: - False.

   Explanation: - In a good design, the module should have high cohesion, when similar functions (e.g. print) are put into a module, it displays logical cohesion however functional cohesion is the best cohesion.

7. In the object-oriented design, the basic abstraction is real-world functions.

   Ans.: - False.

   Explanation: - In OOD, the basic abstraction are not real-world functions such as sort, display, track etc., but real-world entities such as employee, picture, machine, radar system, etc.

8. An OOD (Object-Oriented Design) can be implemented using object-oriented languages only.

   Ans.: - False.

   Explanation: - An OOD can also be implemented using a conventional procedural language – though it may require more effort to implement an

OOD using a procedural language as compared to the effort required for implementing the same design using an object-oriented language.