# Structuring the Program: Good Modularity

1. Goals ........................................................................................................... 2  
2. Cohesion ...................................................................................................... 2  
   2.1. Coincidental cohesion ........................................................................ 3  
   2.2. Logical cohesion ............................................................................... 3  
   2.3. Temporal cohesion ........................................................................... 4  
   2.4. Procedural cohesion .......................................................................... 4  
   2.5. Communicational cohesion ............................................................... 5  
   2.6. Informational cohesion ...................................................................... 5  
   2.7. Functional cohesion ........................................................................... 6  
3. Coupling ....................................................................................................... 7  
   3.1. Content Coupling .............................................................................. 8  
   3.2. Common Coupling ............................................................................ 9  
   3.3. Control Coupling ............................................................................... 10  
   3.4. Stamp Coupling ............................................................................... 11  
   3.5. Data Coupling ................................................................................... 11  
4. Formal Definitions ......................................................................................... 12
1. Goals

- **Maximize** interaction within each (cohesion) and **minimize** interaction between modules (coupling)

2. Cohesion

- the degree to which the internals of a module are related

- A module has high cohesion if all of its elements are strongly related: elements are grouped together for a logical reason, not just by chance. They cooperate to achieve the common goal of the module.

- High cohesion ➔ well-designed reusable module

- There are seven cohesion levels:
  1. Coincidental cohesion ➔ BAD
  2. Logical cohesion
  3. Temporal cohesion
  4. Procedural cohesion
  5. Communication cohesion
  6. Informational cohesion
  7. Functional cohesion ➔ GOOD
2.1. Coincidental cohesion

- A module has coincidental cohesion if it performs completely unrelated actions.

- Ex.

  Module(p1,p2,p3)
  Begin
      Update_item_record (p1);
      Delete_orders(p2);
      Insert_new_customer_info(p3);
  End

- Problem: Not reusable.

2.2. Logical cohesion

- A module has logical cohesion when it performs a series of related actions, one of which is selected by the calling module.

- Ex.

  ➔ New_operation(function_code,p1,p2,p3)
  /* p1 and p3 are not used when this function is called with a function_code > 7 and less than 20 */

  ➔ A module performing all I/O operations: disk, tape, printers, etc.
• Problems:
  ➔ Module interface is difficult to understand
  ➔ Difficult to maintain

2.3. Temporal cohesion

• A module has temporal cohesion when it performs a series of related actions in time.

• Ex.
  ➔ An initialization module:
    Initial several unrelated objects:
    customer_table, item_table, etc.

  ➔ Modules to manipulate these objects are located in other modules

• Problems: Maintenance and Reusability

2.4. Procedural cohesion

• A module has procedural cohesion if it performs a series of actions related by the sequence of steps to be followed by the product.

• Ex.
  Read(part_number, part_table);
Update(repair_record)
maintain(customer_table)

- Better than temporal cohesion

- Problem: Reusability: actions are weakly related.

2.5. **Communicational cohesion**

- A module has communicational cohesion if it performs a series of actions related by the sequence of steps to be followed by the product and if all actions are performed on the same data.

- Better than procedural cohesion: actions of the module are closely related.

- Ex.
  
  ```
  Update_record(R,table);
  Write_record(R,audit_table);
  ```

2.6. **Informational cohesion**

- A module has informational cohesion if it performs a number of actions each with its own entry point, with independent code for each action, all performed on the same data structure.

- All actions in an informational cohesion level are closely related.
• Informational cohesion is optimal for OO.

• Ex: ADT

```
<table>
<thead>
<tr>
<th>Definition of item_table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization of item_table</td>
</tr>
<tr>
<td>Update item_table</td>
</tr>
<tr>
<td>Print item_table</td>
</tr>
</tbody>
</table>
```

2.7. Functional cohesion

• A module that performs one action and achieves one single goal has functional cohesion.

• Ex.

```
Compute(sales_commission);
```

• Communicational cohesion: reusable, easy to maintain and to understand.
3. Coupling

- Originally designed to support programming in the large
  - Modules
  - Encapsulation
  - Interfaces

- Coupling measures the degree to which the modules of a design are related.

- Ex. Module A calls a routine provided by B or access an object of module B.

  Module A and B have high coupling if they depend on each other heavily.
• low coupling is a desirable feature. Modules are:

  ➞ Modules are Reusable
  ➞ Modules are easily tested and modified
  ➞ Reduction of the maintenance cost

• There are five coupling levels:

  1. Data coupling ➞ GOOD
  2. Stamp coupling
  3. Control coupling
  4. Common coupling
  5. Content coupling ➞ BAD

3.1. Content Coupling

• Two modules are content-coupled if one directly references the contents of the other.

• Ex.

  ➢ Module p modifies a statement of module q
  ➢ Module p refers to local data of module q
  ➢ Module p refers to a local label of module q

• Problems:

  ➢ Any change to q may require a change to p.

  ➢ It is impossible to reuse module p without reusing q
3.2. Common Coupling

- Two modules are common-coupled if they both have access to the same global data.

- Ex.

  ![](image)

  ➢ Both module p and module q read and write a record of a database. Read-only is not a common coupling.

- Problems:

  ➢ Unreadable code:

```c
  While(global_variable==0)
  { 
    if(local_variable>val)
      function_1();
    else
      function_2();
  }
```
> global_variable may be changed by function_1() or function_2().

> Modification of the declaration of the global_variable may yield a modification of every module that accesses the global variable.

> Common-coupled are not reusable

> They depend on modules where global variables are declared.

3.3. Control Coupling

- Two modules are control-coupled if one passes an element of control to the other module; one module explicitly the logic of the other.

- Ex.

  If p calls q and q passes back a flag to p that says, "I am unable to perform the requested action Ai, please write the message number 1020.

  q informs p as what to do ➔ Control-coupling
3.4. Stamp Coupling

- Two modules are stamp-coupled if a whole data structure is passed as an argument but the called module operates on only some of the individual components of that data structure.

- Problems:
  - Data access cannot be controlled: More data is passed than needed.

- Optimization:
  - Passing different variables \( \rightarrow \) slower
  - Passing one single record \( \rightarrow \) faster

  - Knuth's first law: Don't!!

  - Required optimization \( \rightarrow \) Leave it to the experts

  - Knuth's second law: Not yet!!

3.5. Data Coupling

- Two modules are data-coupled if all arguments are homogeneous data items.
- It is a desirable goal.

- Data-coupled modules are easier to maintain.
4. Formal Definitions

- A software system can be modeled as a graph. Each node (Fi) in the graph corresponds to a method (function or procedure) call of a module and each edge corresponds to an interaction (dependency) between methods in the system:

![Graph Diagram]

- Cohesion:
  - The cohesion of a module is the extent to which its individual methods are needed to perform some tasks. Cohesion of a module is defined in terms of the ratio of internal relationships to the total number of relationships. The cohesion of a module mi is:

\[
CH(mi) = \frac{R_i}{R_i + Re}
\]
Where Ri is the number of internal links and Re is the number of external links.

- Example:

The cohesion of M1 and M2 in Figure 1 are:

\[
\begin{align*}
CH(M1) &= 2/3 \\
CH(M2) &= 1/2 \\
\end{align*}
\]

- Example: The cohesion of a software system is the average cohesion of all its n modules:

\[
CH = \frac{1}{n} \sum \text{CH}(mi) \text{ for all mi of the system.}
\]

- Example: The cohesion of the system in Figure 1 is:

\[
CH = 7/6
\]

• Coupling

- The coupling of a module is the ratio of the number of external links to the total number of links:

\[
CP(mi) = \frac{Re}{Ri + Re}
\]

- The coupling of a system is the average coupling of all its n modules:

\[
CP = \frac{1}{n} \sum CP(mi) \text{ for all mi of the system.}
\]