CS 390 – Lecture 21
Chapter 7: Modules to Objects

○ A module is a lexically contiguous sequence of program statements, bounded by boundary elements, with an aggregate identifier
  - "Lexically contiguous"
  - Adjoining in the code
  - "Boundary elements"
    - { ... } begin ... end
  - "Aggregate identifier"
    - A name for the entire module

Design of Computer (Figure 7.1)

○ A highly incompetent computer architect decides to build an ALU, shifter, and 16 registers with AND, OR, and NOT gates, rather than NAND or NOR gates

Design of Computer (Figure 7.2)

○ The architect designs three silicon chips
  - Obvious arrangement is one chip per function

Design of Computer (Figure 7.3)

○ Instead, redesign with one gate type per chip
  - Resulting “masterpiece”

Computer Design (4)

○ The two designs are functionally equivalent
  - The second design is
    - Hard to understand
    - Hard to locate faults
    - Difficult to extend or enhance
    - Cannot be reused in another product
  - Modules must be like the first design
    - Maximal relationships within modules, and
    - Minimal relationships between modules

Composite/Structured Design

○ A method for breaking up a product into modules to achieve
  - Maximal interaction within a module, and
  - Minimal interaction between modules
○ Module cohesion
  - Degree of interaction within a module
○ Module coupling
  - Degree of interaction between modules
Function, Logic, and Context of a Module

- In C/SD, the name of a module is its function
- Example:
  - A module computes the square root of double precision integers using Newton’s algorithm. The module is named `compute_square_root`

Cohesion

- The degree of interaction within a module
- Seven categories or levels of cohesion - non-linear scale (Figure 7.4)

<table>
<thead>
<tr>
<th>Level</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Coincidental cohesion</td>
<td><code>print_next_line</code>, <code>reverse_string_of_second_parameter</code>, <code>add_7_to_fifth_parameter</code>, <code>convert_fourth_parameter_to_floating_point</code></td>
</tr>
<tr>
<td>2. Logical cohesion</td>
<td><code>op_code = 7; new_operation(op_code, dummy_1, dummy_2, dummy_3);</code></td>
</tr>
<tr>
<td>3. Temporal cohesion</td>
<td><code>An object performing all input and output</code></td>
</tr>
<tr>
<td>4. Procedural cohesion</td>
<td><code>One version of OS/VS2 contained a module with logical cohesion performing 13 different actions. The interface contains 21 pieces of data</code></td>
</tr>
<tr>
<td>5. Communicational cohesion</td>
<td></td>
</tr>
<tr>
<td>6. Functional cohesion</td>
<td></td>
</tr>
<tr>
<td>7. Informational cohesion</td>
<td></td>
</tr>
</tbody>
</table>

Coincidental Cohesion

- A module has coincidental cohesion if it performs multiple, completely unrelated actions
- Example:
  - `print_next_line`, `reverse_string_of_second_parameter`, `add_7_to_fifth_parameter`, `convert_fourth_parameter_to_floating_point`
- Such modules arise from rules like
  - “Every module will consist of between 35 and 50 statements”

Why Is Coincidental Cohesion So Bad?

- It degrades maintainability
- A module with coincidental cohesion is not reusable
- The problem is easy to fix
  - Break the module into separate modules, each performing one task

Logical Cohesion

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

Logical Cohesion (2)

- Example 1:
  - `op_code = 7; new_operation(op_code, dummy_1, dummy_2, dummy_3);` // dummy_1, dummy_2, and dummy_3 are dummy variables, not used if op code is equal to 7
- Example 2:
  - An object performing all input and output
- Example 3:
  - One version of OS/VS2 contained a module with logical cohesion performing 13 different actions. The interface contains 21 pieces of data
Why Is Logical Cohesion So Bad?
- The interface is difficult to understand
- Code for more than one action may be intertwined
- Difficult to reuse

Temporal Cohesion
- A module has temporal cohesion when it performs a series of actions related in time
- Example:
  - open_old_master_file, new_master_file, transaction_file, and print_file;
  - initialize_sales_district_table, read_first_transaction_record, read_first_old_master_record
  (a.k.a. perform_initialization)

Why Is Temporal Cohesion So Bad?
- The actions of this module are weakly related to one another, but strongly related to actions in other modules
- Consider sales_region_table
- Operations update_sales_table, print_sales_region_table, ...
- Not reusable

Procedural Cohesion
- A module has procedural cohesion if it performs a series of actions related by the procedure to be followed by the product
- Example:
  - read_part_number_and_update_repair_record_on_master_file

Why Is Procedural Cohesion So Bad?
- The actions are still weakly connected, so the module is not reusable
- Solution is to break up actions into separate modules

Communicational Cohesion
- A module has communicational cohesion if it performs a series of actions related by the procedure to be followed by the product, but in addition all the actions operate on the same data
- Example 1:
  - update_record_in_database_and_write_it_to_audit_trail
- Example 2:
  - calculate_new_coordinates_and_send_them_to_terminal
Why Is Communicational Cohesion So Bad?
- Still lack of reusability
- Solution is to break up actions into separate modules

Why Is Functional Cohesion So Good?
- More reusable
- Corrective maintenance is easier
  - Fault isolation
  - Fewer regression faults
- Easier to extend a product

Why Is Informational Cohesion So Good?
- Essentially, this is an abstract data type
- Allows changes in a module implementation without changing users of the module
- Well-designed objects are modules with informational cohesion

Functional Cohesion
- A module with functional cohesion performs exactly one action
  - Example 1: `get_temperature_of_furnace`
  - Example 2: `compute_orbital_of_electron`
  - Example 3: `write_to_diskette`

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

Cohesion Example (Figure 7.7)