# CS342 Software Engineering

Dr. Ismail Hababeh
German Jordanian University

## Lecture 15
## DESIGN WORKFLOW

*Adapted from Software Engineering, by Dr. Paul E. Young*
*& slides by Dr. Mohammad Daoud*

# The Design Workflow

- The main objective is to refine the analysis workflow.
- Puts the product processes in a form that can be implemented.
- Many nonfunctional requirements should be finalized including
  - ✓Choice of programming language
  - ✓Reuse issues
  - ✓Portability issues

# The Design Workflow

Two types of design workflow:

- Classical Design
- Object-Oriented Design

# Classical Design

- Architectural design
  - Divide the product into modules
- Detailed design: design each module
  - ✓ Data structures
  - ✓ Algorithms

# Object-Oriented Design

- Classes are extracted during the object-oriented analysis workflow and designed during the design workflow.
- Architectural design corresponds to part of the object-oriented analysis workflow
- Detailed design corresponds to part of the object-oriented design workflow

# Classical Design Activities

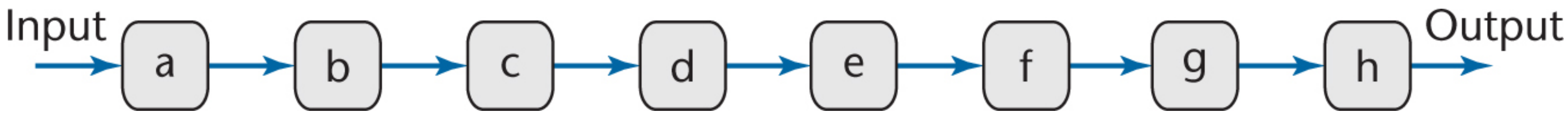
- Classical design activities:
  - Architectural design
    - Input: Specifications
    - Output: Modular decomposition
  - Detailed design
      - Specific algorithms, data structures
  - Design Testing

# Actions and Data

- Two aspects of a classical design
  - Actions that operate on data
  - Data on which actions operate
- The two basic ways of designing a product
  - Operation-oriented design
  - Data-oriented design
- Third way
  - Hybrid methods (for example, object-oriented design)

# Operation-Oriented Design

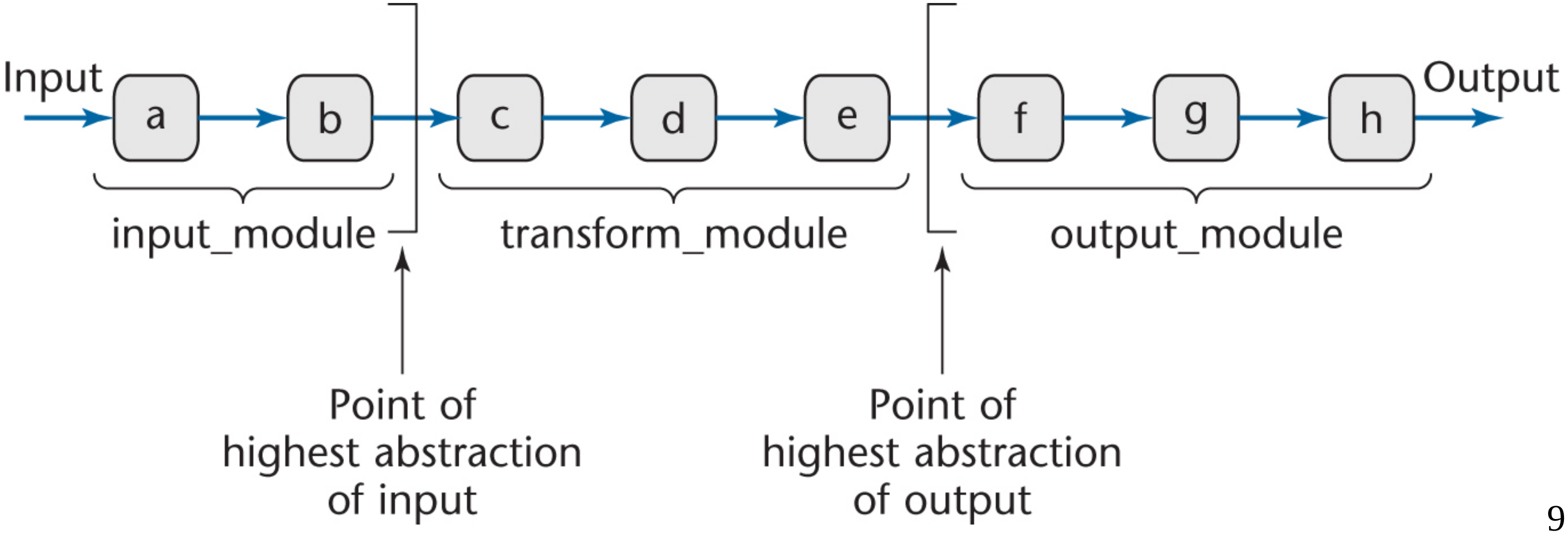
- Key point: We have detailed action information from the DFD



- Data Flow Analysis (DFA)
  - A classical design technique for achieving modules with high cohesion.
  - Used with most specification methods

# Data Flow Analysis - Example

- Consider the following data flow diagram. Every product transforms input into output, therefore determine:
  - "Point of highest abstraction of input": the point at which the input losses its quality of being input and becomes internal data
  - "Point of highest abstraction of output": the first point at which data can be identified as an output

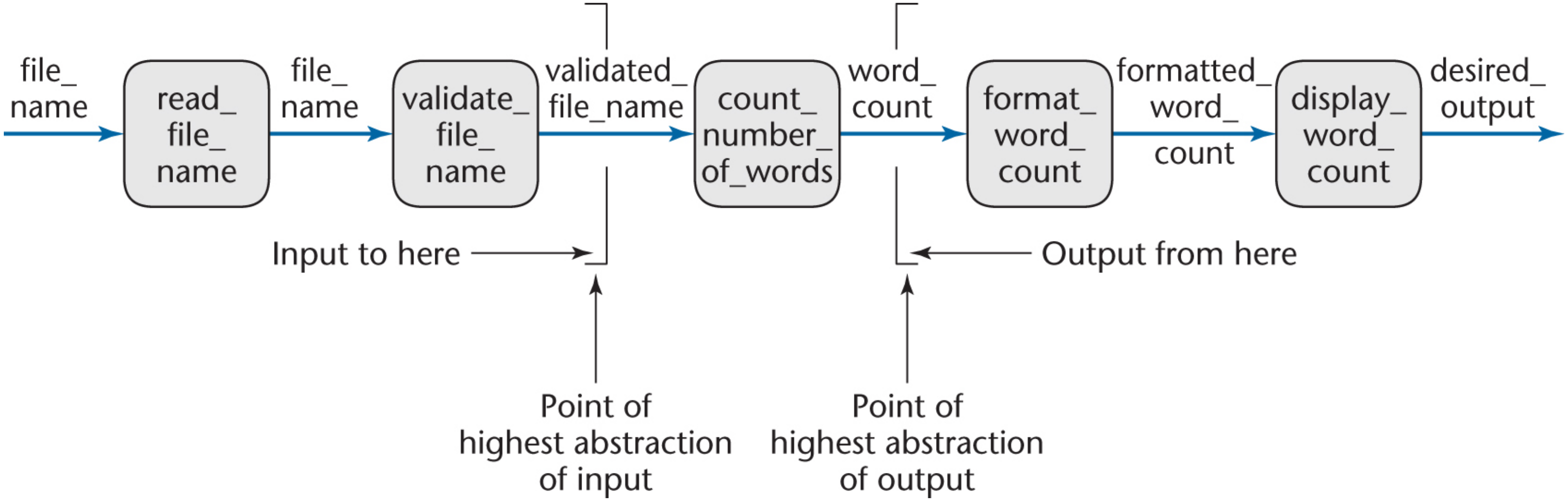

Input
a
b
c
d
e
f
g
h
Output
input_module
transform_module
output_module
Point of highest abstraction of input
Point of highest abstraction of output

# Data Flow Analysis

- Decompose the product into three modules:
  - Input module
  - Transform module
  - Output module
- Each module is taken in turn, its points of highest abstraction are found, and the module decomposition is performed again
- Repeat stepwise until each module has high cohesion:
  - The design consists of modules where each module preforms a single operation (has high cohesion)

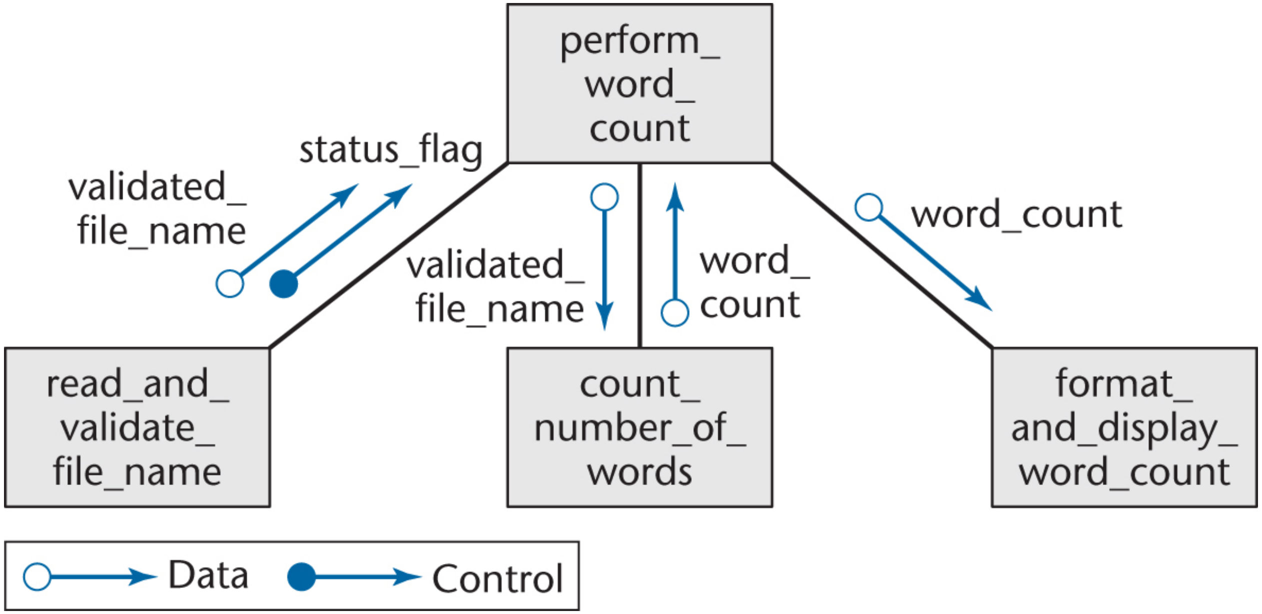
# Data Flow Analysis- Word Counting Example

Design a product which takes as input a text file and returns the number of words in that file (like UNIX *wc* )



# Data Flow Analysis- Word Counting Example

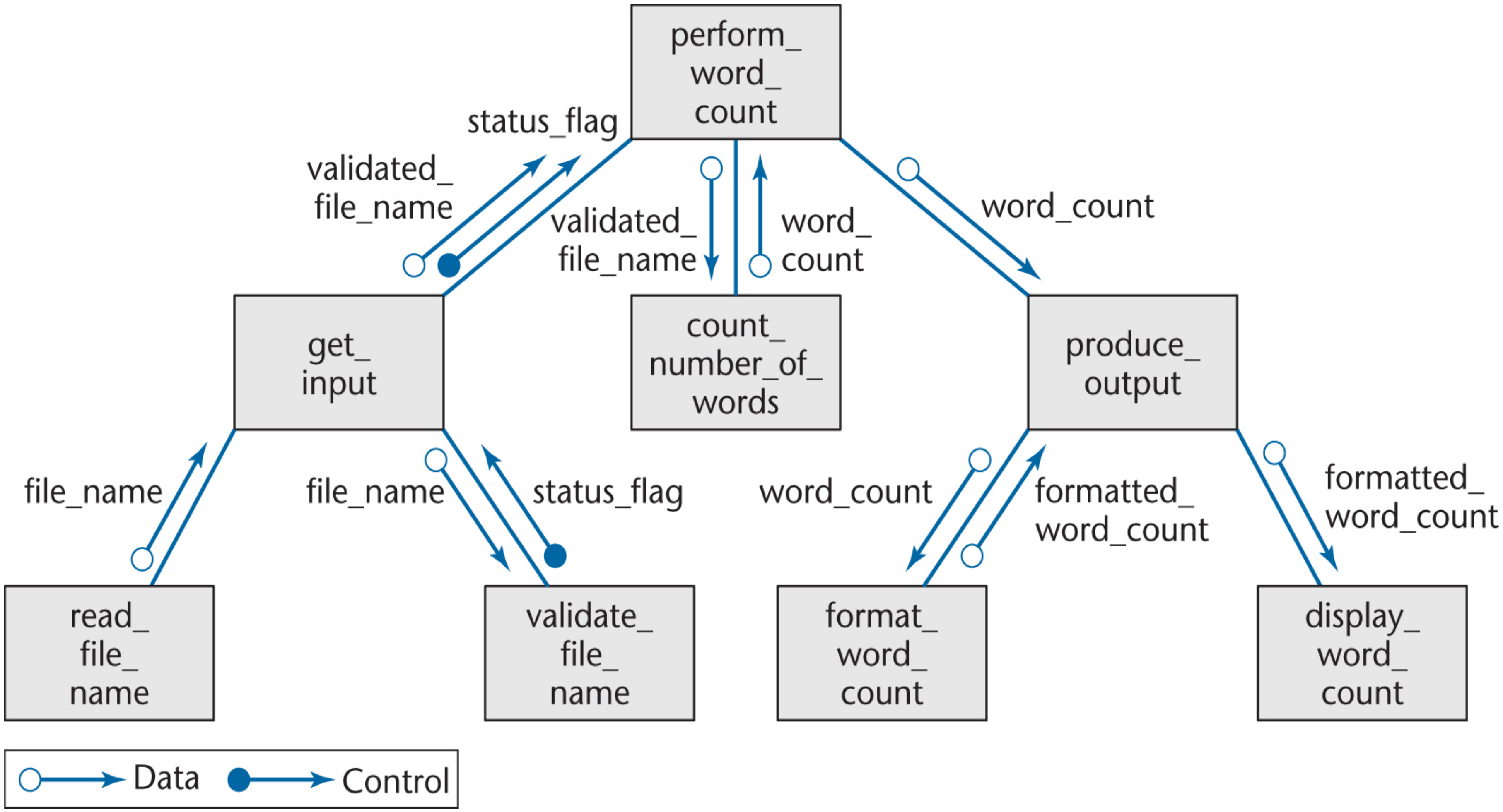
- First refinement



- Now refine the two modules of communicational cohesion (read_and_validate_file_name and format_and_display_word_count).
- A module has communication cohesion if it performs a series of operations on the same data.

# Word Counting Case Study

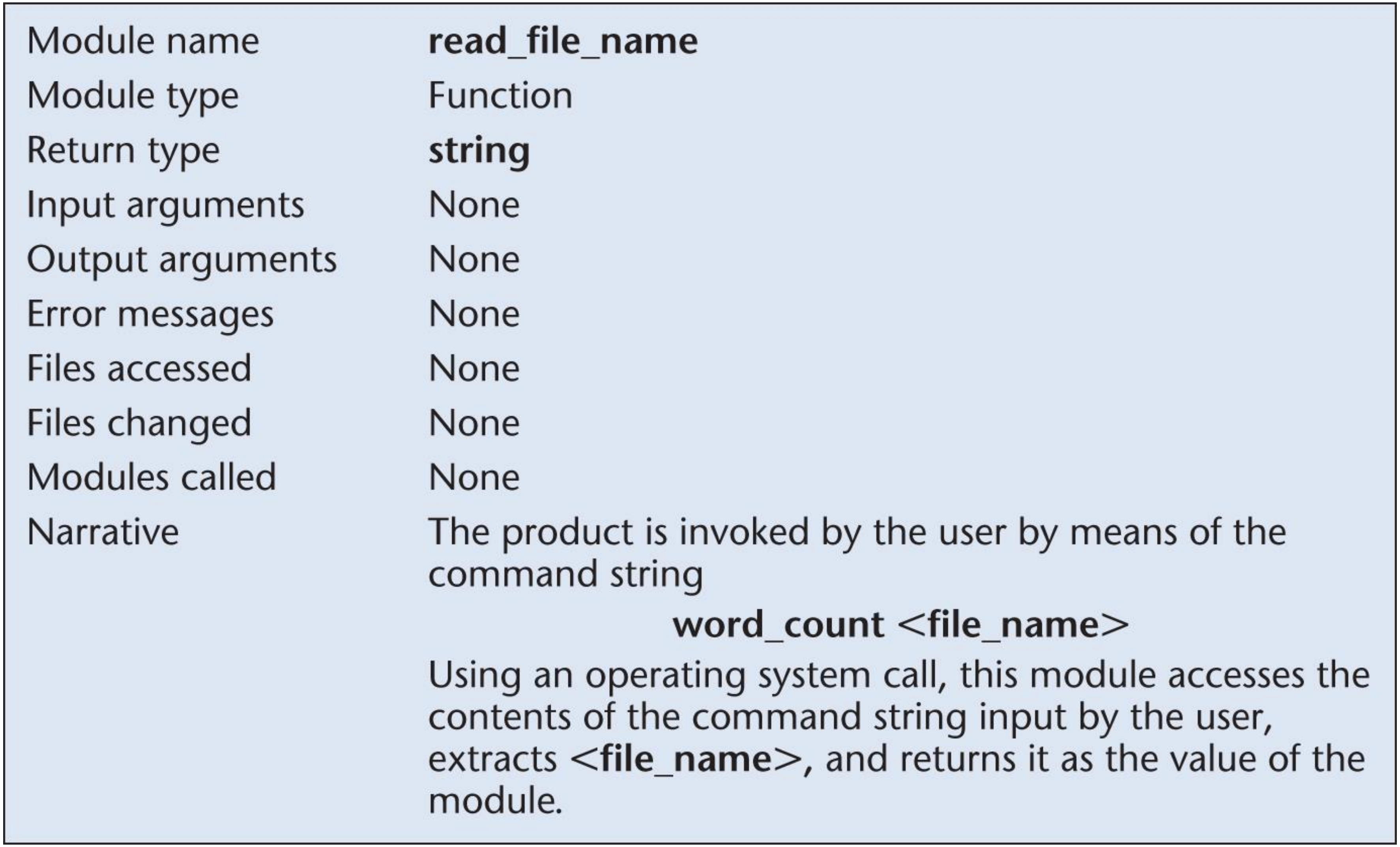
- Second refinement



# Data Flow Analysis- Word Counting Example

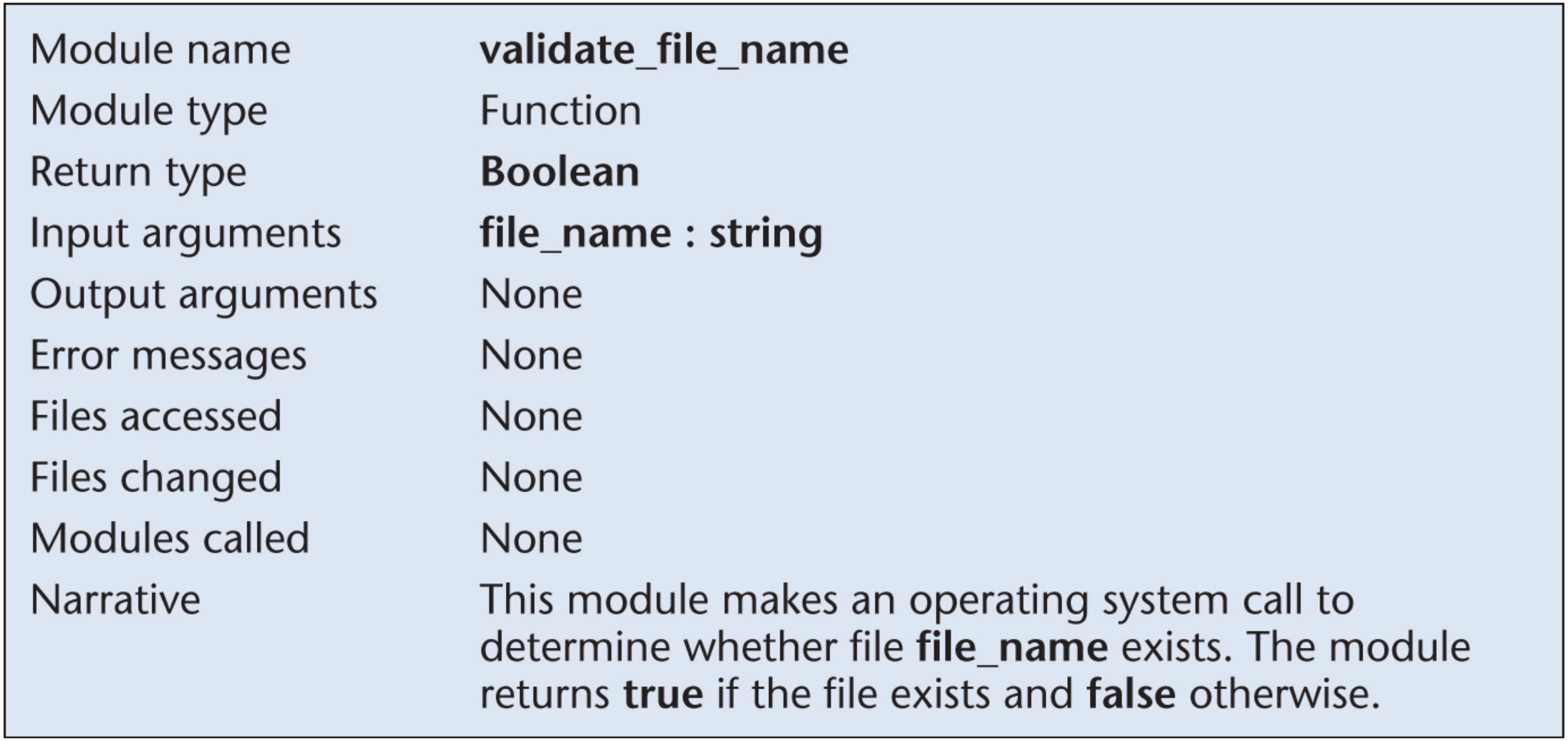
- Once the architectural design is completed, proceed to the detailed design.
- Two types of design format for representing the detailed design:
  - Tabular
  - Pseudocode (PDL — Program Design Language)

# Detailed Design: Tabular Format



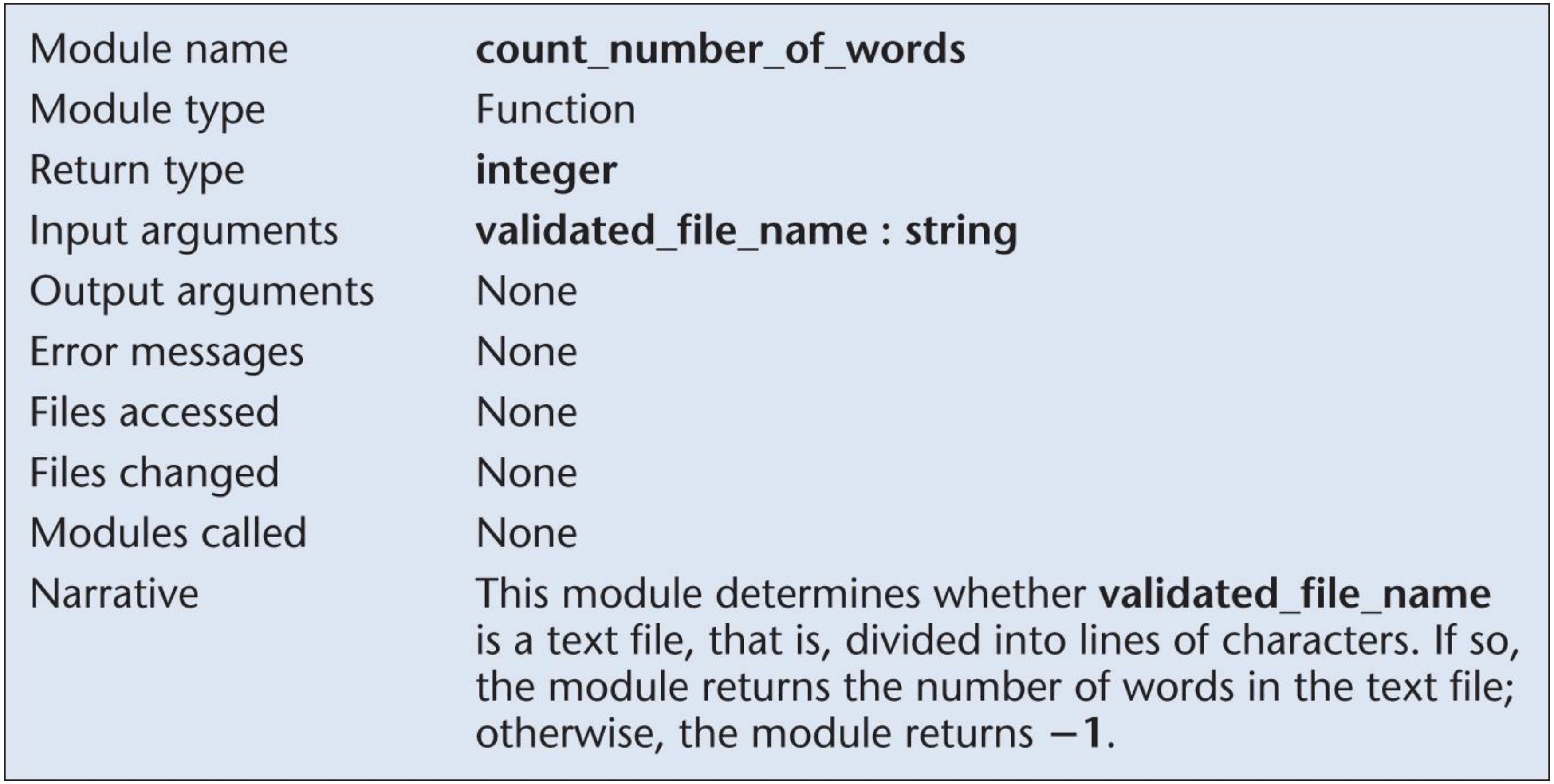
| | |
|---|---|
| Module name | **read_file_name** |
| Module type | Function |
| Return type | **string** |
| Input arguments | None |
| Output arguments | None |
| Error messages | None |
| Files accessed | None |
| Files changed | None |
| Modules called | None |
| Narrative | The product is invoked by the user by means of the command string<br>**word_count <file_name>**<br>Using an operating system call, this module accesses the contents of the command string input by the user, extracts **<file_name>**, and returns it as the value of the module. |

# Detailed Design: Tabular Format



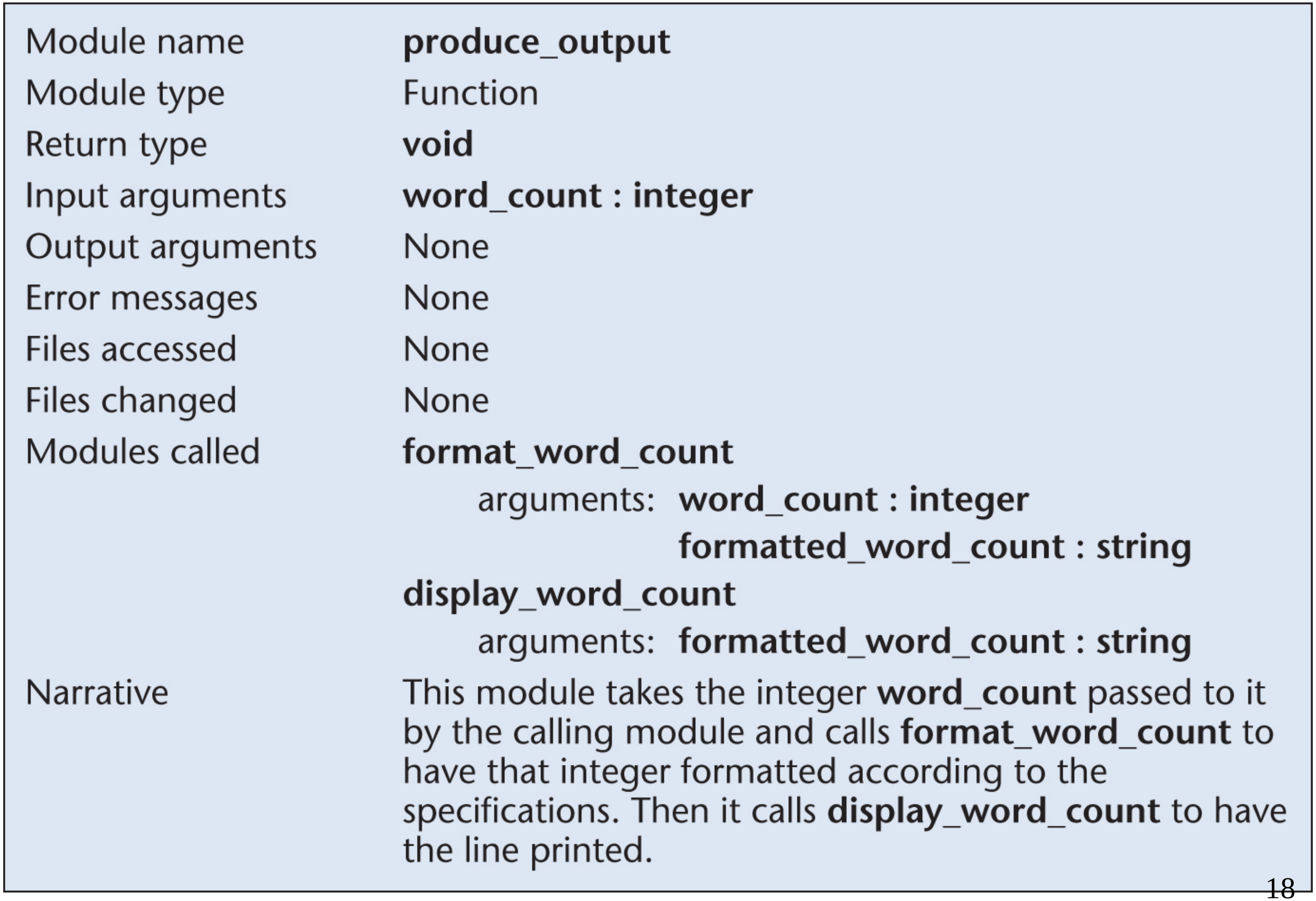
| | |
|---|---|
| Module name | **validate_file_name** |
| Module type | Function |
| Return type | **Boolean** |
| Input arguments | **file_name : string** |
| Output arguments | None |
| Error messages | None |
| Files accessed | None |
| Files changed | None |
| Modules called | None |
| Narrative | This module makes an operating system call to determine whether file **file_name** exists. The module returns **true** if the file exists and **false** otherwise. |

# Detailed Design: Tabular Format



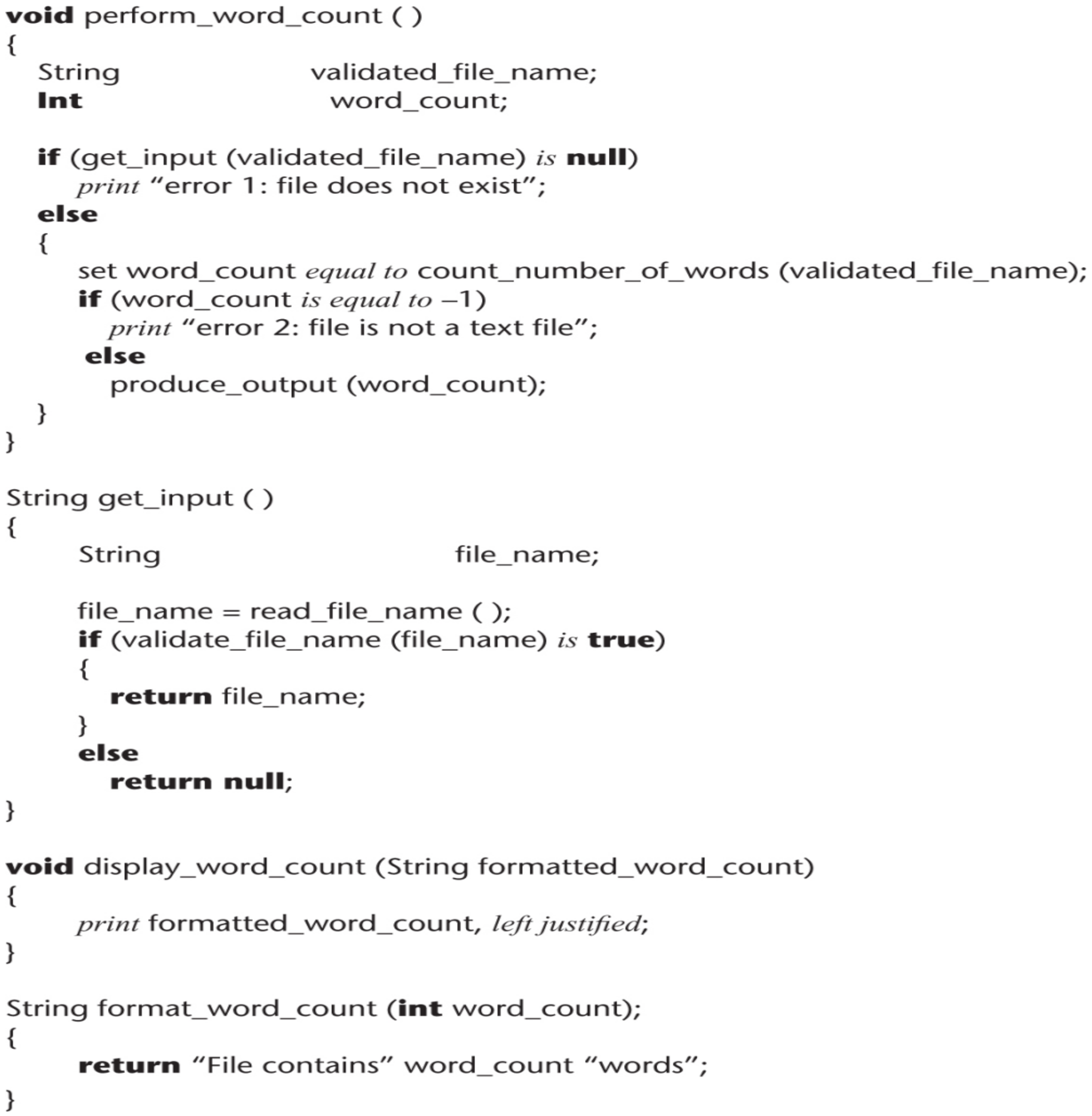
| | |
|---|---|
| Module name | **count_number_of_words** |
| Module type | Function |
| Return type | **integer** |
| Input arguments | **validated_file_name : string** |
| Output arguments | None |
| Error messages | None |
| Files accessed | None |
| Files changed | None |
| Modules called | None |
| Narrative | This module determines whether **validated_file_name** is a text file, that is, divided into lines of characters. If so, the module returns the number of words in the text file; otherwise, the module returns **−1**. |

# Detailed Design: Tabular Format



| | |
|---|---|
| Module name | **produce_output** |
| Module type | Function |
| Return type | **void** |
| Input arguments | **word_count : integer** |
| Output arguments | None |
| Error messages | None |
| Files accessed | None |
| Files changed | None |
| Modules called | **format_word_count**<br>arguments: **word_count : integer**<br>**formatted_word_count : string**<br>**display_word_count**<br>arguments: **formatted_word_count : string** |
| Narrative | This module takes the integer **word_count** passed to it by the calling module and calls **format_word_count** to have that integer formatted according to the specifications. Then it calls **display_word_count** to have the line printed. |

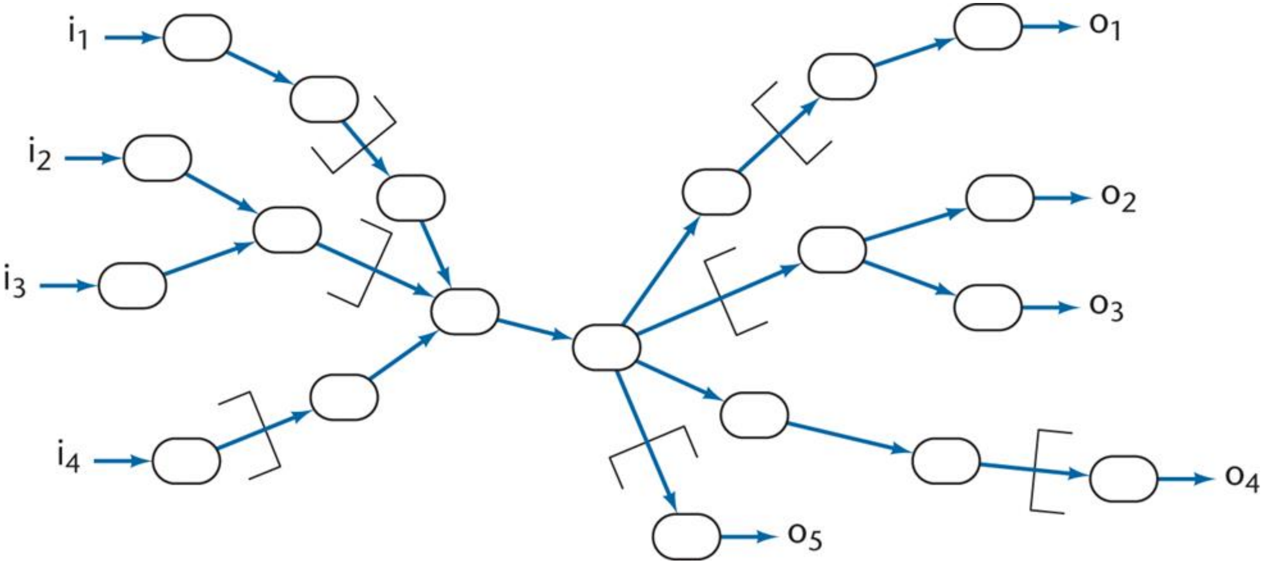
# Detailed Design: Pseudocode PDL



```
void perform_word_count ( )
{
   String                    validated_file_name;
   Int                         word_count;

   if (get_input (validated_file_name) is null)
       print "error 1: file does not exist";
   else
   {
       set word_count equal to count_number_of_words (validated_file_name);
       if (word_count is equal to –1)
          print "error 2: file is not a text file";
        else
          produce_output (word_count);
   }
}

String get_input ( )
{
       String                               file_name;

       file_name = read_file_name ( );
       if (validate_file_name (file_name) is true)
       {
          return file_name;
       }
       else
          return null;
}

void display_word_count (String formatted_word_count)
{
       print formatted_word_count, left justified;
}

String format_word_count (int word_count);
{
       return "File contains" word_count "words";
}
```

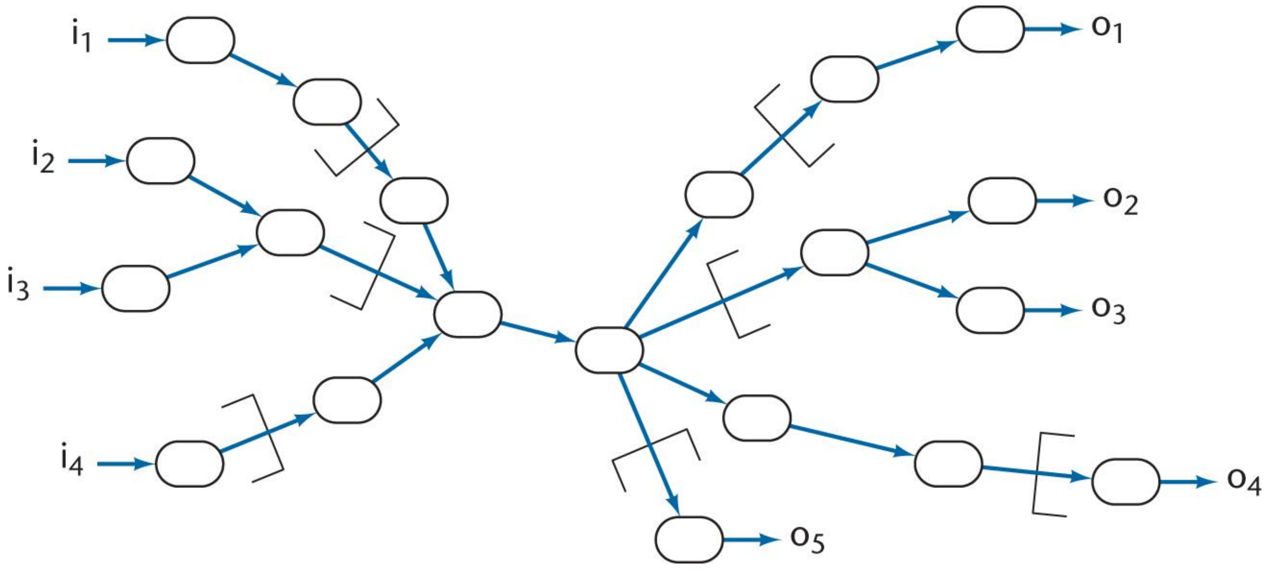
# Data Flow Analysis Extensions

- In real-world products, there are
  - – More than one input stream
  - – More than one output stream



# Data Flow Analysis Extensions

- Find the point of highest abstraction for each stream



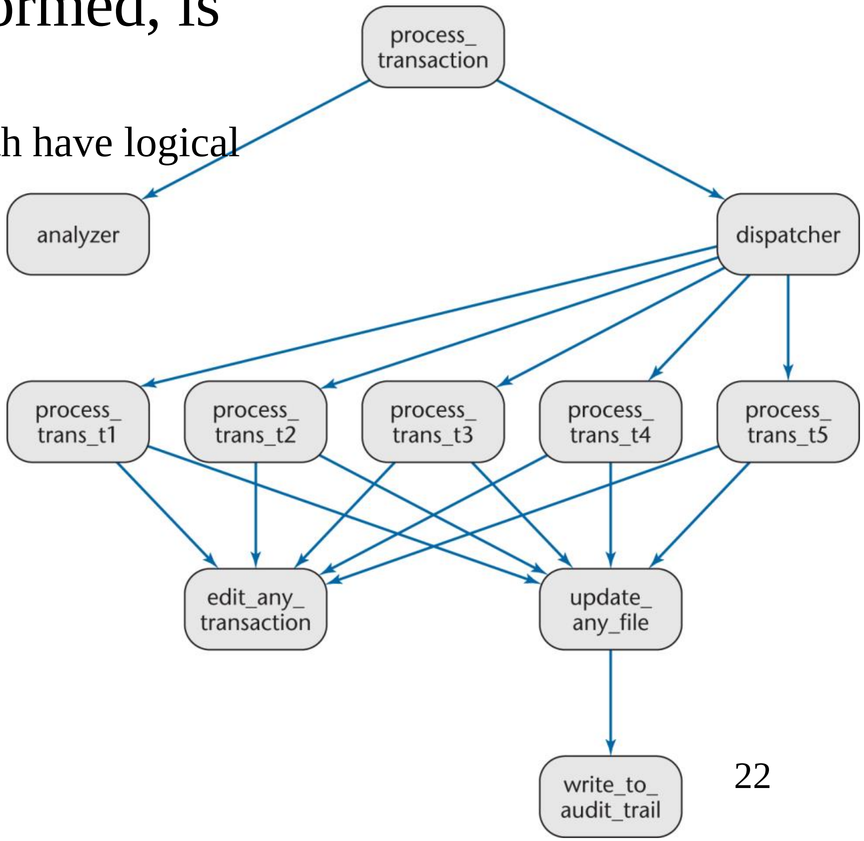
- Continue until each module (input, transform, output) has high cohesion, adjust the coupling if needed

# Transaction Analysis

- A transaction is an operation from the viewpoint of the user of the product, such as “process a request”
- Transaction-processing, in which number of related operations must be performed, is inappropriate for DFA

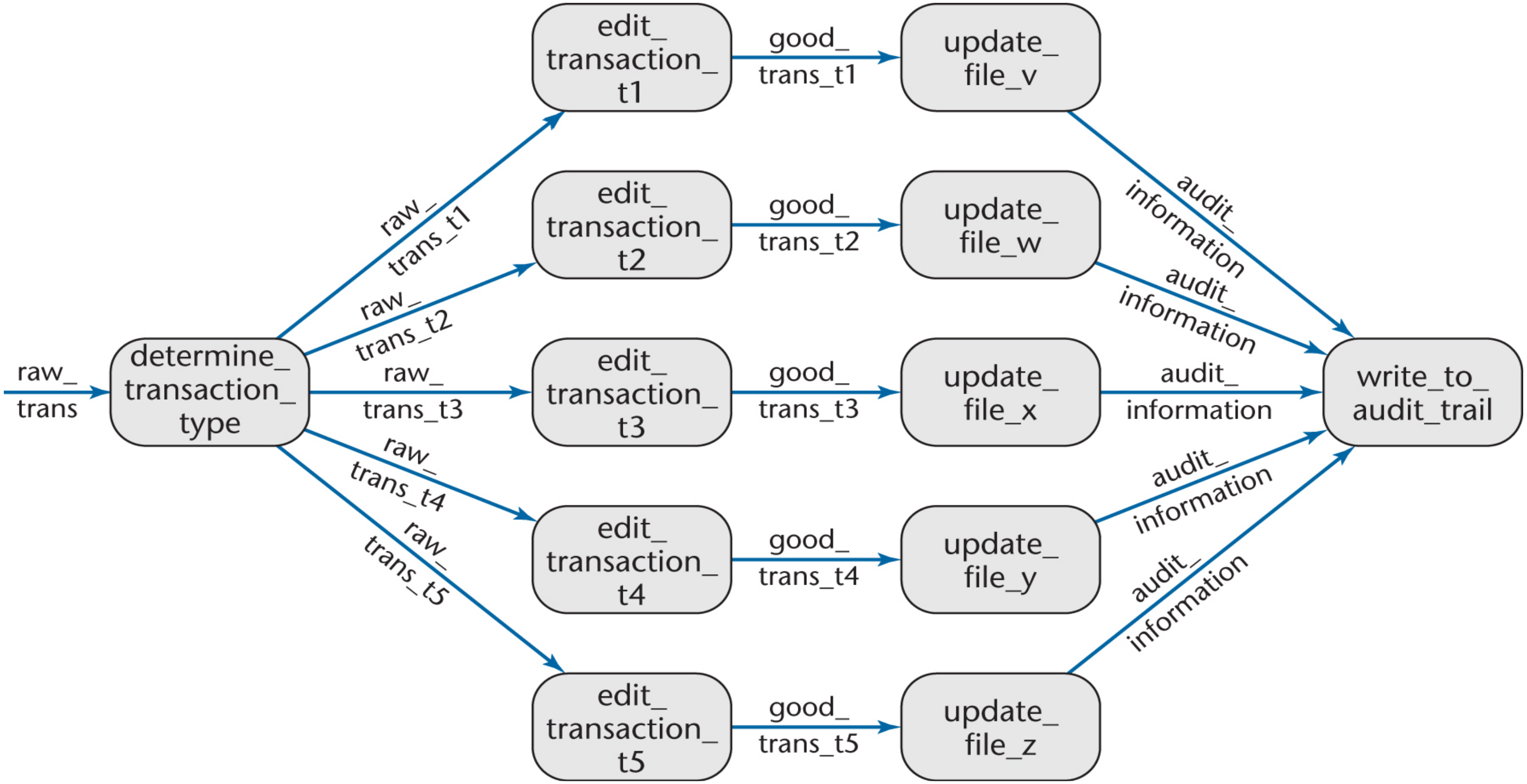
(Edit_any_transaction and write_to_audit_trail both have logical cohesion -> poor design).

- – Example: ATM Software
  - ○ Have one generic `edit` module, and one generic `update` module
  - ○ Instantiate them 5 times



# Corrected Design Using Transaction Analysis

We need to employ SW reuse: a basic edit module should be designed, coded, documented, tested, and then produce 5 versions. Each version is slightly different.



# Data-Oriented Design

- Basic principle
  - The structure of a product should be adapted to the structure of its data
- Three very similar methods
  - Michael Jackson [1975], Warnier [1976], Orr [1981]
- Data-oriented design
  - Not popular as action-oriented design
  - With the rise of Object-Oriented Design, data-oriented design has largely fallen out of fashion