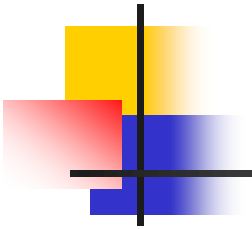
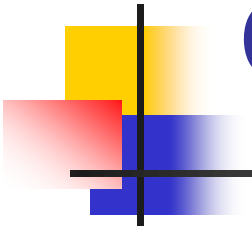


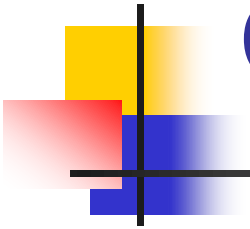
Lecture 7

Software Design



Contents

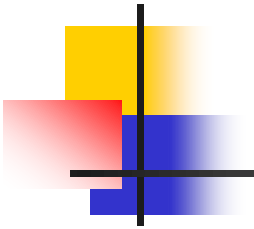
- Software Design
- Design Principle
- Design Models
- Design Issues
- Architectural Design
- Advantages of Explicit Architecture
- Architectural Design Process
- Distinction Between Sub-systems and Modules
- Architectural Models
- Architecture Non-functional Attributes



Contents (Cont)

- Data and Actions
- Design and Abstraction
- Operation-oriented Design
- Data Flow Analysis
- Cohesion
- Coupling
- Mini Case Study: Word Counting
- Word Counting: Detailed Design
- Detailed Design: Tabular Format
- Detailed Design: PDL Format

Software Design



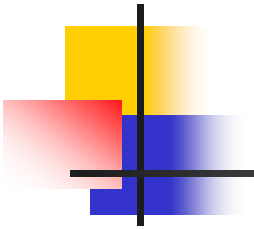
- Sits at the kernel of Software Engineering (SE) and is applied regardless of the software process that is used.
- Once software requirements have been analysed and specified, software design is the first of the three technical activities – design, code generation and test.
- Design is important. It is the place where quality is fostered in SE. Design is the only way to translate a customer's requirements into a finished software product.

Design Principle

- The design process is a sequence of steps that enable the designer to describe all aspects of the software to be built.
  - Creative skill, past experience, a sense of what make good software and an overall commitment to quality to achieve good design.
- The design model is created to provide a variety of different views of the computer software.

Design Models

- **Data design** – transforms the information domain model during system analysis into the data structure.
- **Architectural design** – defines the relationship between major structural of the software.
- **Interface design** – describes how the software communications within itself, with systems that interoperate with it and with the humans who use it.
- **Component-level design** – transforms structural elements of the software architecture into a procedural description of software components.



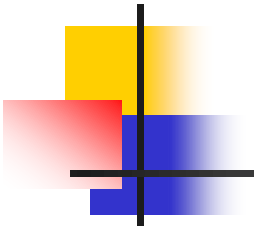
Design Issues

- Large systems always decomposed into sub-systems.
- The design process for identifying the sub-systems that make up a system and the framework for sub-system control and communication is known as *Architectural Design*.
- The output of this design process is a description of the *Software Architecture*.

Architectural Design

- An early stage of the system design process.
  - Establishing a basis structural framework for a system.
- Represents the link between specification and design processes.
- Often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.

Advantages of Explicit Architecture

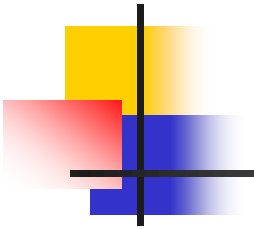


- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems.

Architectural Design Process

1. System structuring
   - The system is decomposed into several principal sub-systems and communications between these sub-systems are identified.
2. Control modelling
   - A model of the control relationships between the different parts of the system is established.
3. Modular decomposition
   - The identified sub-systems are decomposed into modules.

Distinction Between Sub-systems and Modules

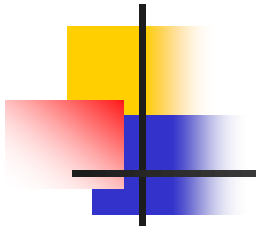


- A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems.
- A module is a system component that provides services to other components but would not normally be considered as a separate system.

Architectural Models

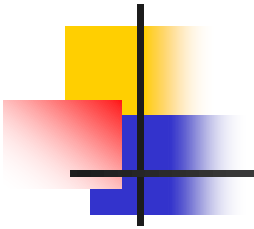
- Different architectural models may be produced during the design process.
- Each model presents different perspectives of the architecture.
- Static structural model shows the major system components.

Architectural Models (Cont)

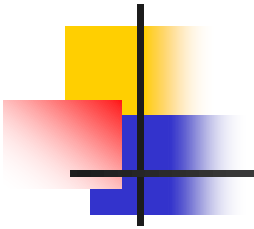


- Dynamic process model shows the process structure of the system.
- Interface model defines sub-system interfaces.
- Relationships model such as a data-flow model.

Architecture Non-functional Attributes



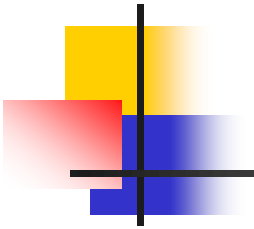
- **Performance**
  - Localise operations to minimise sub-system communication.
- **Security**
  - Use a layered architecture with critical assets in inner layers.
- **Safety**
  - Isolate safety-critical components.
- **Availability**
  - Include redundant components in the architecture.
- **Maintainability**
  - Use fine-grain, self-contained components.



Data and Actions

- Two aspects of a product:
  - 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.

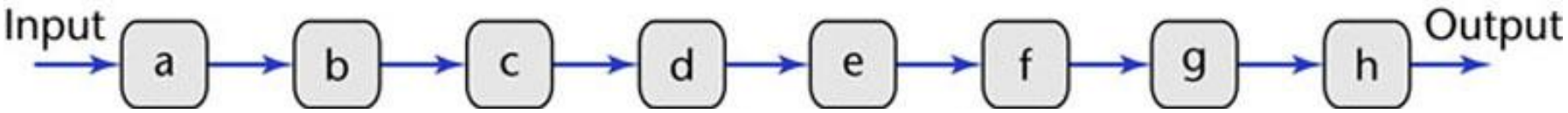
Design and Abstraction



- Classical design activities
  - **Architectural design**
  - **Detailed design**
  - **Design testing**
- Architectural design
  - **Input: Specifications**
  - **Output: Modular decomposition**
- Detailed design
  - **Each module is designed.**
    - Specific algorithms, data structures.

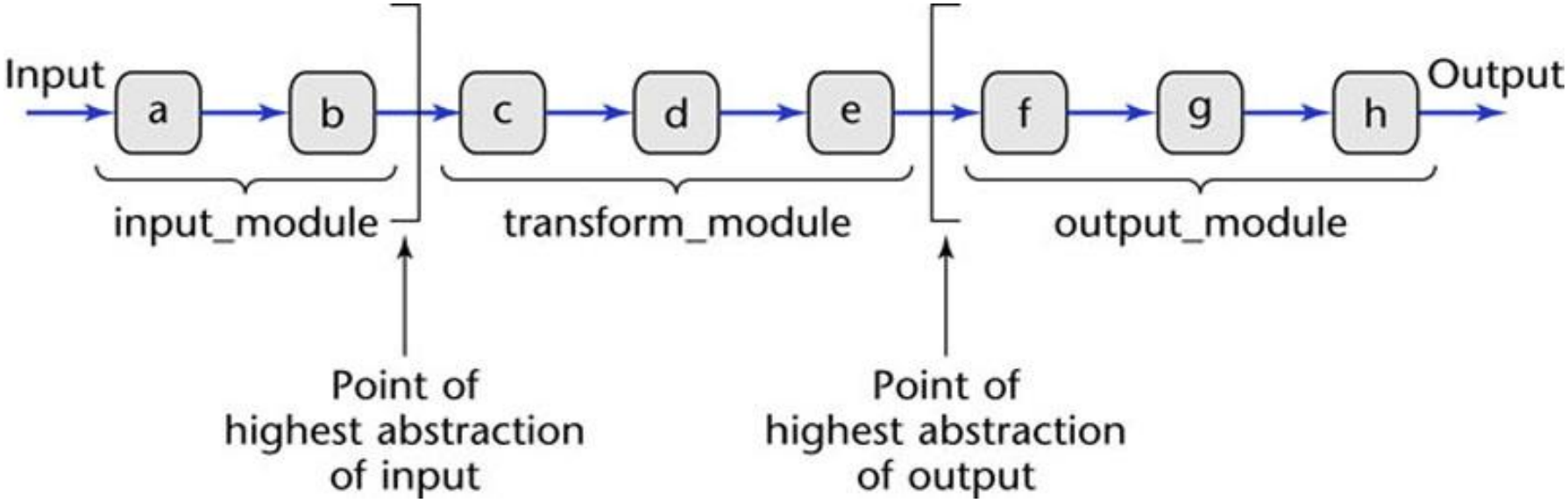
Operation-oriented Design

- Data flow analysis
  - Use it with most specification methods. (Structured Systems Analysis)
- Key point: the detailed action information is from the DFD.

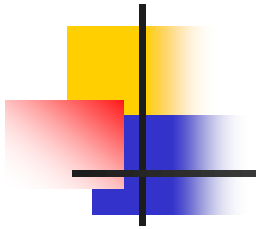


Data Flow Analysis

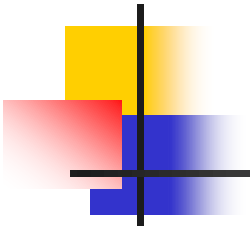
- Every product transforms input into output.
- Determine:
  - “Point of highest abstraction of input”.
  - “Point of highest abstract of output”.



Data Flow Analysis (Cont)



- Decompose the product into three modules.
- Repeat stepwise until each module has high cohesion.
  - Minor modifications may be needed to lower the coupling.

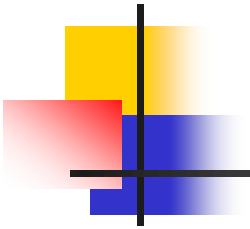


Cohesion

- The degree of interaction within a module.
- Seven categories or levels of cohesion:
  - Coincidental Cohesion (Bad)
  - Logical Cohesion
  - Temporal Cohesion
  - Procedural Cohesion
  - Communicational Cohesion
  - Functional Cohesion
  - Informational Cohesion (Good)

Cohesion (Cont)

- **Coincidental Cohesion: A module has coincidental cohesion if it performs multiple, completely unrelated operations.**
- **Logical Cohesion: A module has logical cohesion when it performs a series of related operations, one of which is selected by the calling module.**
- **Temporal Cohesion: A module has temporal cohesion when it performs a series of operations related in time.**
- **Procedural Cohesion: A module has procedural cohesion if it performs a series of operations related by the sequence of steps to be followed by the product.**
- **Communicational Cohesion: A module has communicational cohesion if it performs a series of operations related by the sequence of steps to be followed by the product and if all the operations are performed on the same data.**
- **Functional Cohesion: A module that performs exactly one operation or achieves a single goal has functional cohesion.**
- **Informational Cohesion: A module has informational cohesion if it performs a number of operations, each with its own entry point, with independent code for each operation, all performed on the same data structure.**



Coupling

- The degree of interaction between two modules.
- Five categories or levels of coupling:
  - Content Coupling (Bad)
  - Common Coupling
  - Control Coupling
  - Stamp Coupling
  - Data Coupling (Good)

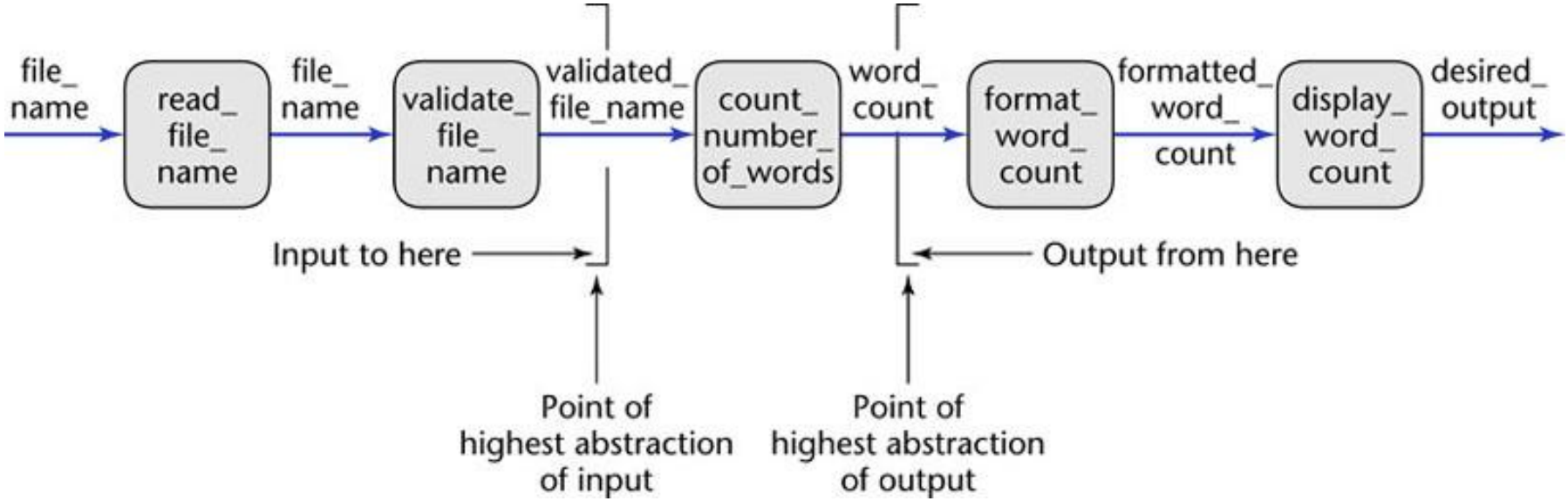
Coupling (Cont)

- Two modules are content coupled if one directly references the contents of the other.
- Two modules are common coupled if both have access to the same global data.
- Two modules are control coupled if one passes an element of control to the other module; that is, one module explicitly controls the logic of the other.
- Two modules are stamp coupled if a data structure is passed as an argument, but the called module operates on only some of the individual components of that data structure.
- Two modules are data coupled if all arguments are homogeneous data items. That is, every argument is either a simple argument or a data structure in which all elements are used by the called module.

Mini Case Study: Word Counting

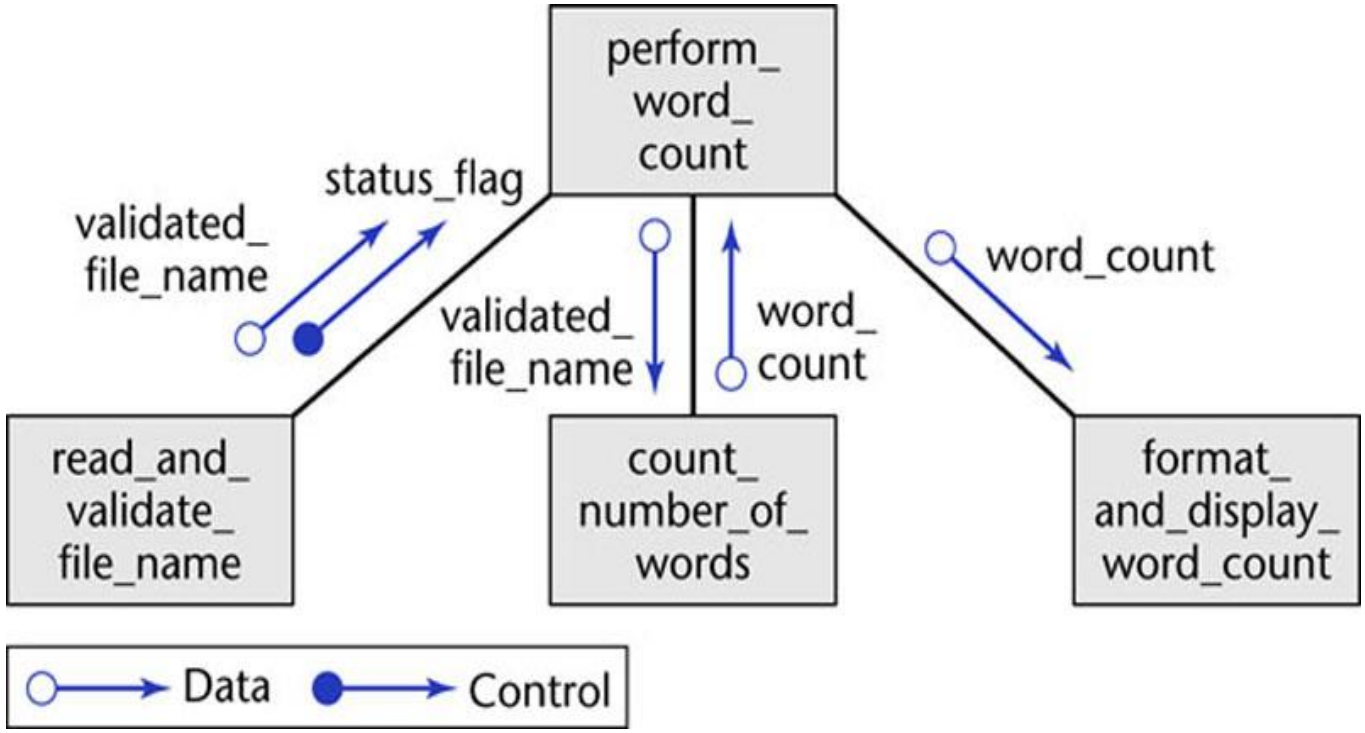
- Example:

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



Mini Case Study: Word Counting (Cont)

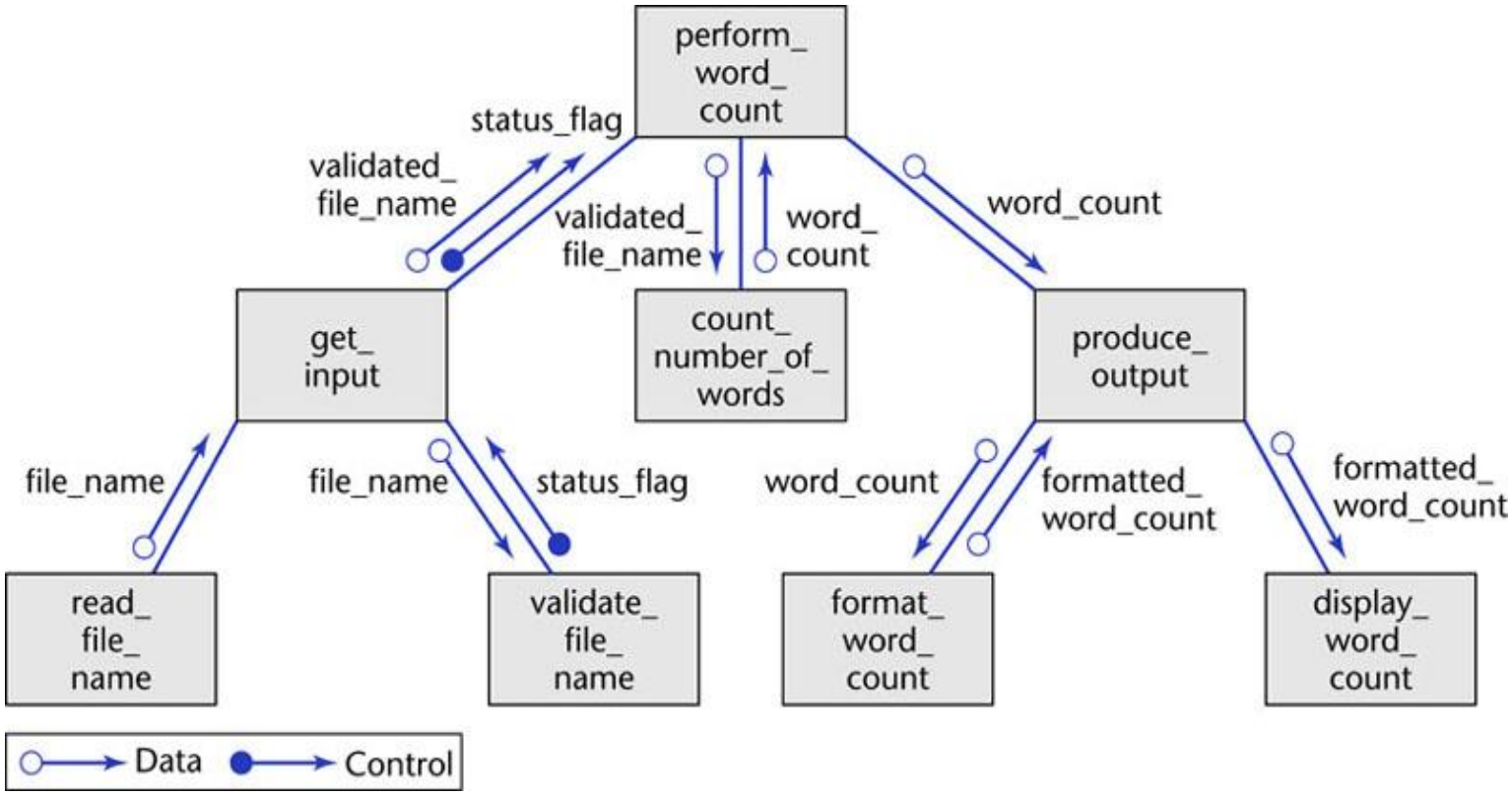
- First refinement.



- Now refine the two modules of communicational cohesion.

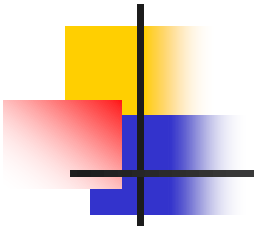
Mini Case Study: Word Counting (Cont)

- Second refinement.



- All eight modules now have functional cohesion.

Word Counting: Detailed Design



- The architectural design is complete.
  - Proceed to the detailed design.
- Two formats 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 **word_count <file_name>** 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 (Cont)

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 (Cont)

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 (Cont)

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** arguments: **word_count : integer** **formatted_word_count : string** **display_word_count** 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: PDL Format

```
void perform_word_count()
{
  String              validate_file_name;
  int                 word_count;

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

```
Boolean get_input (String validate_file_name)
{
  String                file_name;

  file_name = read_file_name ();
  if (validate_file_name (file_name) is true)
  {
    set validate_file_name equal to file_name;
    return true;
  }
  else
    return false;
}

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";
}
```