

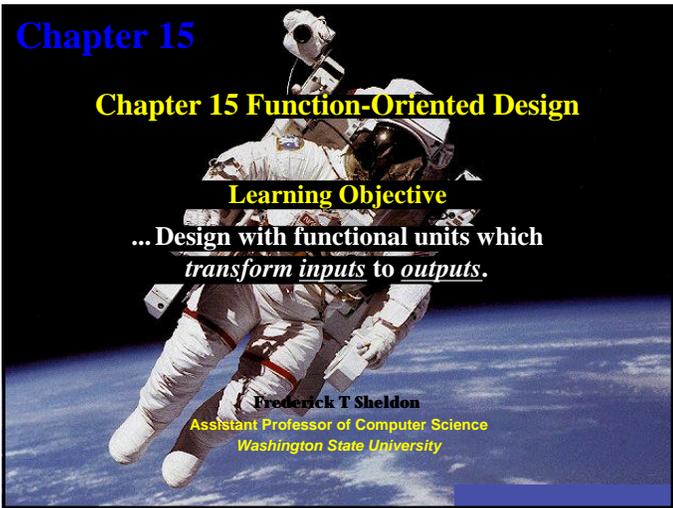
Chapter 15

Chapter 15 Function-Oriented Design

Learning Objective

... Design with functional units which transform inputs to outputs.

Frederick T Sheldon
Assistant Professor of Computer Science
Washington State University



Objectives

- To explain how a software design may be represented as a set of functions that share state
- To introduce notations for function-oriented design
- To illustrate the function-oriented design process by example
- To compare sequential, concurrent and object-oriented design strategies

CS-422 Software Engineering Principles
From Software Engineering by I. Sommerville, 1996.

Chapter 15
Slide 2

Topics covered

- Data-flow design
- Structural decomposition
- Detailed design
- A comparison of design strategies

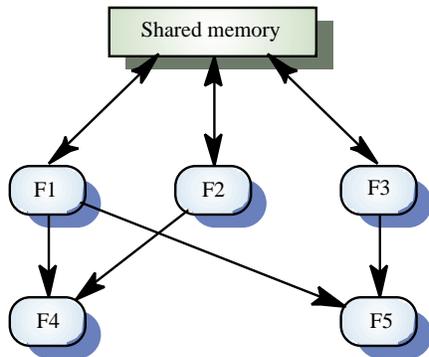
CS-422 Software Engineering Principles
From Software Engineering by I. Sommerville, 1996.

Chapter 15
Slide 3

Function-oriented design

Practiced informally since programming began
Thousands of systems have been developed using this approach
Supported directly by most programming languages
Most design methods are functional in their approach
CASE tools are available for design support

A function-oriented view of design



Natural functional systems

Some systems are naturally function-oriented
Systems which maintain minimal state information
i.e. where the system is concerned with processing
independent actions whose outcomes are not affected
by previous actions
Information sharing through parameter lists
Transaction processing systems fall into this category.
Each transaction is independent

An ATM system design

```
loop
loop
Print_input_message (" Welcome - Please enter your card" );
exit when Card_input ;
end loop ;
Account_number := Read_card ;
Get_account_details (PIN, Account_balance, Cash_available) ;
if Validate_card (PIN) then
loop
Print_operation_select_message ;
case Get_button is
when Cash_only =>
Dispense_cash (Cash_available, Amount_dispensed) ;
when Print_balance =>
Print_customer_balance (Account_balance) ;
when Statement =>
Order_statement (Account_number) ;
when Check_book =>
Order_checkbook (Account_number) ;
end case ;
Eject_card ;
Print ("Please take your card or press CONTINUE") ;
exit when Card_removed ;
end loop ;
Update_account_information (Account_number, Amount_dispensed)
else
Retain_card ;
end if ;
end loop ;
```

Functional and object-oriented design

For many types of application, object-oriented design is likely to lead to a more *reliable* and *maintainable system*

Some applications maintain little state ---> hence function-oriented design is appropriate

Standards, methods and CASE tools for functional design are *well-established*

Existing systems must be maintained ---> *hence function-oriented design will be practiced well into the 21st century*

Functional design process

Data-flow design

Model the data processing in the system using data-flow diagrams

Structural decomposition

Model how functions are decomposed to sub-functions using graphical structure charts

Detailed design

The entities in the design and their interfaces are described in detail. These may be recorded in a data dictionary and the design expressed using a PDL (Program Description Language)

Data flow diagrams

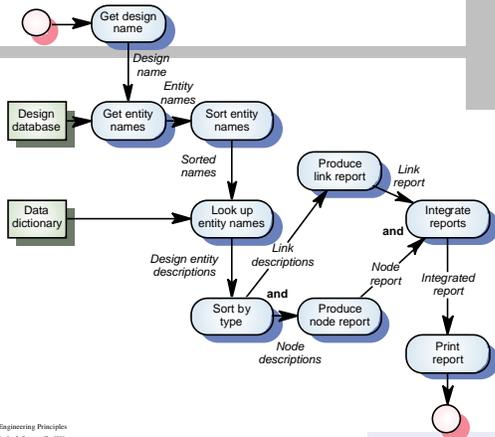
Show how an input data item is functionally transformed by a system into an output data item
Are an integral part of many design methods and are supported by many CASE systems
May be translated into either a sequential or parallel design:

- In a *sequential* design, processing elements are functions or procedures
- In a *parallel* design, processing elements are tasks or processes

DFD notation

- Rounded rectangle - function or transform
- Rectangle - data store
- Circles - user interactions with the system
- Arrows - show direction of data flow
- Keywords “and” / “or” used to link data flows

Design report generator



Structural decomposition

Structural decomposition is concerned with developing a model of the design which shows the dynamic structure i.e. function calls

This is not the same as the static composition structure

The aim of the designer should be to derive design units which are highly cohesive and loosely coupled

In essence, a data flow diagram is converted to a structure chart

Decomposition guidelines

For business applications, the top-level structure chart may have four functions namely input, process, master-file-update and output

Data validation functions should be subordinate to an input function

Coordination and control should be the responsibility of functions near the top of the hierarchy

Decomposition guidelines

The aim of the design process is to identify *loosely coupled, highly cohesive* functions. Each function should therefore do one thing and one thing only

Each node in the structure chart should have between two and seven subordinates

Process steps

Identify system processing transformations

Transformations in the DFD which are concerned with processing rather than input/output activities. Group under a single function in the structure chart

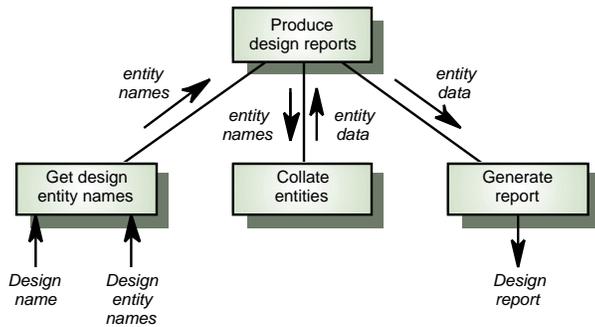
Identify input transformations

Transformations concerned with reading, validating and formatting inputs. Group under the input function

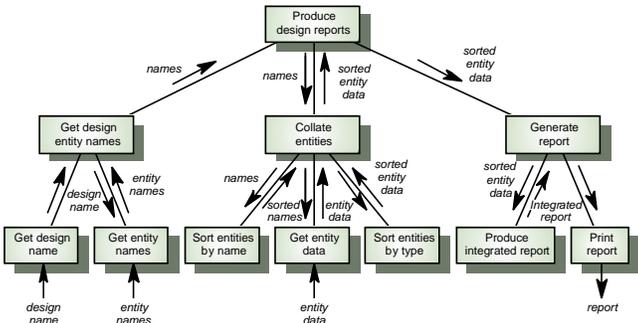
Identify output transformations

Transformations concerned with formatting and writing output. Group under the output function

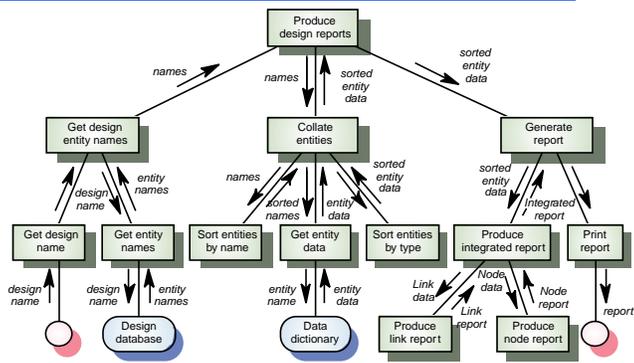
Initial structure chart



Expanded structure chart



Final structure chart



CS 422 Software Engineering Principles

From Software Engineering by I. Sommerville, 1996.

Chapter 15

Slide 19

Detailed design

Concerned with producing a short design specification (mini-spec) of each function.

This should describe the processing, inputs and outputs

These descriptions should be managed in a data dictionary

From these descriptions, detailed design descriptions, expressed in a PDL or programming language, can be produced

CS 422 Software Engineering Principles

From Software Engineering by I. Sommerville, 1996.

Chapter 15

Slide 20

Data dictionary entries

Entity name	Type	Description
Design name	STRING	The name of the design assigned by the design engineer.
Get design name	FUNCTION	<i>Input:</i> Design name <i>Function:</i> This function communicates with the user to get the name of a design that has been entered in the design database. <i>Output:</i> Design name
Get entity names	FUNCTION	<i>Input:</i> Design name <i>Function:</i> Given a design name, this function accesses the design database to find the names of the entities (nodes and links) in that design. <i>Output:</i> Entity names
Sorted names	ARRAY of STRING	A list of the names of the entities in a design held in ascending alphabetical order.

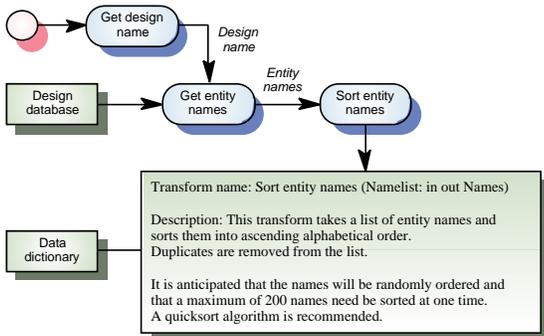
CS 422 Software Engineering Principles

From Software Engineering by I. Sommerville, 1996.

Chapter 15

Slide 21

Design entity information



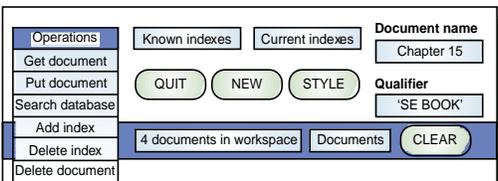
A comparison of design strategies

An example of an office information retrieval system (OIRS) is used to compare different design strategies

Functional design, concurrent systems design and object-oriented design are compared

The OIRS is an office system for document management. Users can file, maintain and retrieve documents using it

OIRS user interface



Function-oriented design is an approach to software design where the design is decomposed into a set of interacting units where each unit has a clearly defined function. By comparison with object-oriented design, the design components in this approach are cohesive around a function whereas object-oriented cohesion is around some abstract data entity.

Function-oriented design has probably been practised informally since programming began but it was only in the late 1960s and early 1970s that it

Interface description

Operation field.

Pull-down menu allowing an operation to be selected.

Known and current indexes fields

Pull-down menus of indexes

Document name.

Name under which the document is to be filed.

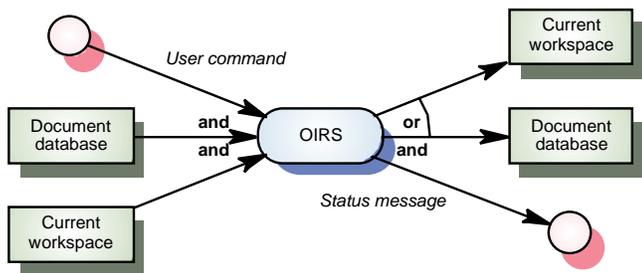
Qualifier field

Pattern used in retrieval.

Current workspace

Contains the documents currently being used. May be edited with word processor

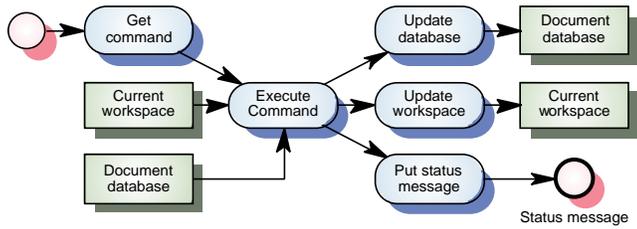
OIRS inputs and outputs



Fetch-execute model

```
procedure Interactive_system is  
begin  
  loop  
    Command := Get_command;  
    if Command = "quit" then  
      -- Make sure files etc. are closed properly  
      Close_down_system ;  
      exit ;  
    else  
      Input_data := Get_input_data ;  
      Execute_command (Command, Input_data, Output_data) ;  
    end if ;  
  end loop ;  
end Interactive_system ;
```

Top-level OIRS DFD



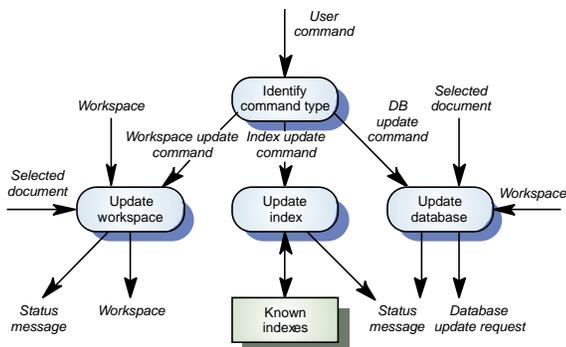
Design decisions

What strategy should be adopted in decomposing the *Execute* command?

Are the input and output data flows processed independently or are they inter-dependent. If independent, there should be a central transform for each processing unit

Is the central transform a series of transforms?
If so, each logical element in the series should be a single transformation

Execute command DFD



OIRS design description

```
procedure OIRS is
begin
  User := Login_user ;
  Workspace := Create_user_workspace (User) ;
  -- Get the users own document database using the user id
  DB_id := Open_document_database (User) ;
  -- get the user's personal index list;
  Known_indexes := Get_document_indexes (User) ;
  Current_indexes := NULL ;
  -- command fetch and execute loop
  loop
    Command := Get_command ;
    exit when Command = Quit ;
    Execute_command ( DB_id, Workspace, Command, Status) ;
    if Status = Successful then
      Write_success_message ;
    else
      Write_error_message (Command, Status) ;
    end if ;
  end loop ;
  Close_database (DB_id) ;
  Logout (User) ;
end OIRS ;
```

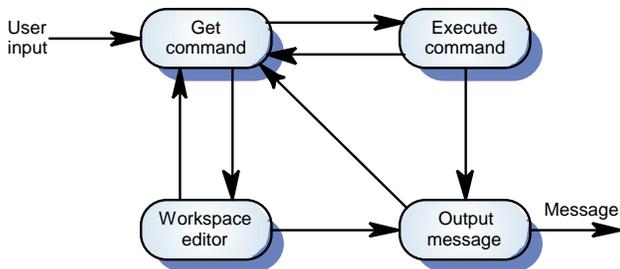
Concurrent systems design

Data flow diagrams explicitly exclude control information. They can be implemented directly as concurrent processes.

Logical groups of transformations can also be implemented as concurrent processes e.g. input data collection and checking

The OIRS system can be implemented as a concurrent system with command input, execution and status reporting implemented as separate tasks

OIRS process decomposition



Detailed process design

```
procedure Office_system is
  task Get_command ;
  task Process_command is
    entry Command_menu ;
    entry Display_indexes ;
    entry Edit_qualifier ;

    -- Additional entries here. One for each command

  end Process_commands ;
  task Output_message is
    entry Message_available ;
  end Output_message ;
  task Workspace_editor is
    entry Enter ;
    entry Leave ;
  end Workspace_editor ;
```

Detailed *process design*

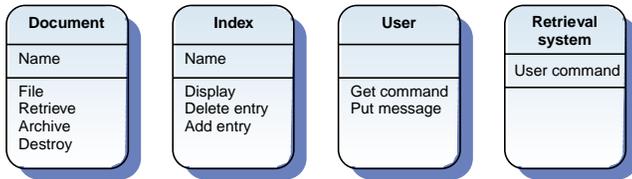
```
task body Get_command is
  begin
    -- Fetch/execute loop
    loop
      loop
        Cursor_position := Get_cursor_position ;
        exit when cursor positioned in workspace or
          (cursor positioned over menu and button pressed)
        Display_cursor_position ;
        end loop ;
        if in_workspace (Cursor_position) then
          Workspace_editor.Enter ;
        elsif In_command_menu (Cursor_position) then
          Process_command.Command_menu ;
        elsif In_Known_indexes (Cursor_position) then
          Process_command.Display_indexes ;
        elsif In_Current_indexes (Cursor_position) then
          ...
          Other commands here
          ...
        end loop ; -- Fetch/execute
      end Get_command ;
      -- other task implementations here
    end Office_system ;
```

Object-oriented design

An object-oriented design focuses on the entities in the system rather than the data processing activities
Simplified OOD here which illustrates a different decomposition

The initial decomposition was introduced in Chapter 14 in the discussion of object identification

Preliminary object identification



New objects required

Workspace

Corresponds to the user's workspace and provides operations to add and remove documents from the workspace

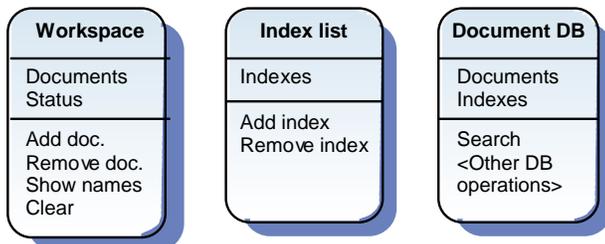
Index list

Provides facilities to manage a list of indexes

Document database

Corresponds to the database of documents, provides search and retrieval operations

Additional OIRS objects



Object refinement

Retrieval system does not provide services. It coordinates other objects. It has only attributes

Documents and indexes are explicitly named

The individual command components have been bundled into a single attribute User command in Retrieval system

The User object has been replaced by the Display object

Modified OIRS objects

Display

Command list
Buttons
Known indexes
Current indexes
Doc. name
Doc. list
Qualifier
WSpace status

Get command
Put message

Retrieval system

User command
Workspace
Known indexes
Current indexes

Key points

Function-oriented design relies on identifying functions which transform inputs to outputs

Many business systems are transaction processing systems which are naturally functional

The functional design process involves identifying data transformations, decomposing functions into sub-functions and describing these in detail

Key points

Data-flow diagrams are a means of documenting end-to-end data flow. Structure charts represent the dynamic hierarchy of function calls

Data flow diagrams can be implemented directly as cooperating sequential processes

Functional and object-oriented design result in different system decompositions. However, a heterogeneous approach to design is often necessary
