Design (II)

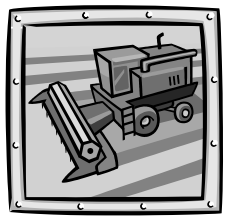
Peter Lo

Interface Design: Are of Concern

- Interface design focuses on 3 areas of concern:
  - Design of interfaces between software modules
  - Design of interfaces between software and other non-human producers and consumers of information
  - Design of the interface between a human and the computer

Internal Interface Design

- Also called **Inter-modular Interface Design**
- Concerned on which data must flow between modules

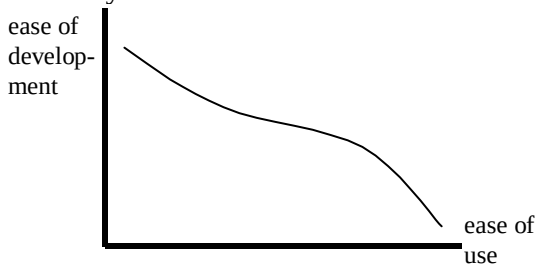


External Interface Design

- Begins with an evaluation of each external object represented by the DFD in the analysis model
- Data and control requirements of each external entity are determined and appropriate external interfaces are designed

User Interface Design

- User Interface Design addresses the study of people and how they react to technology issues.
- A good user interface is not easy to create, since the user interface development effort is inversely proportional to the ease of use finally obtained in the software.



Features of Good User Interface

- **Functionality:**
  - Help the user accomplish the required task
- **Flexibility:**
  - Able to perform slightly different tasks than the user interface was specified for.
- **Intelligence:**
  - In case of ambiguity, the user interface should do the right thing
- **Pain-free Usability:**
  - User should be comfortable interacting with the user interface.

Basic Types of User Interface

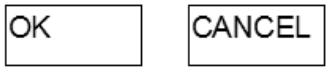
- **Limited Interfaces**
  - Such as those on a telephone (buttons + switch-hook) or an ATM machine (buttons + cash dispenser)
- **Text Interfaces**
  - Such as MS-DOS and UNIX
- **Graphical User Interfaces (GUI)**
  - Such as the Macintosh, Windows etc.

Interface Design Considerations

- Interface Consistency
  - An inconsistent interface is often the source of frustration to the user.
- Predictability
  - Related to the user model and the consistency of the interface, but emphasizes the principle of least surprise.
- Error Handling
  - A good User Interface minimizes the chances of an error occurring, and provides a way to recover from any error that does occur.

Interface Design Considerations

- Ease of Use
  - A User Interface should have on-line help and user guidance, which are accessible when user has trouble.
- Clarity
  - Errors can be minimized if the User Interface communication between the User Interface and the user is as clear as possible.
    - E.g.: Cancel print?



Interface Design Considerations

- Feedback
  - Feedback to the user is important in reducing frustration and provides reassurance that the user task is actually being carried out.

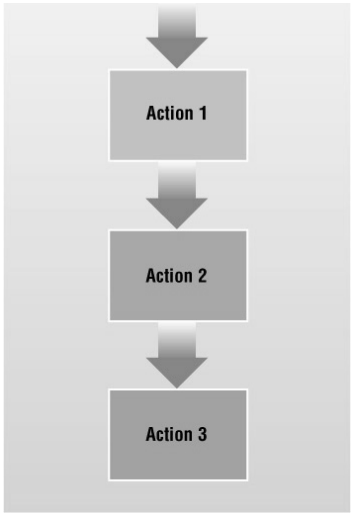
Procedural Design

- To transform structural components into a procedural description of the software.
- Procedural details can be represented in different ways:
  - Graphical Design Notation
  - Tabular Design Notation
  - Program Design Language

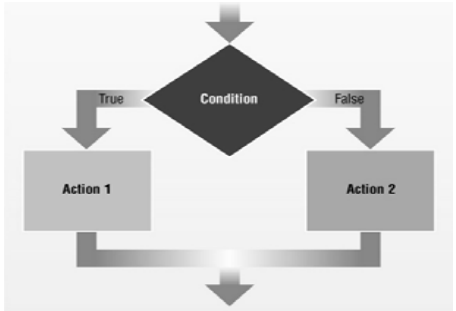
Graphical Notation

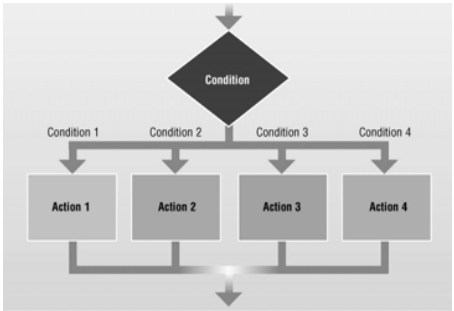
- The most widely used notation is the flowchart. Some notation used in flowcharts are
  - Boxes to indicate processing steps;
  - Diamond to indicate logical conditions;
  - Arrows to indicate flow of control;
  - Two boxes connected by a line of control will indicate a Sequence

Sequence Structure

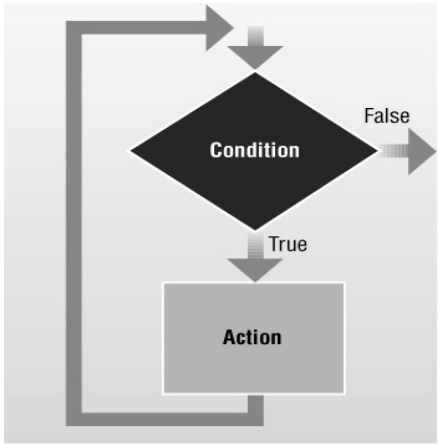


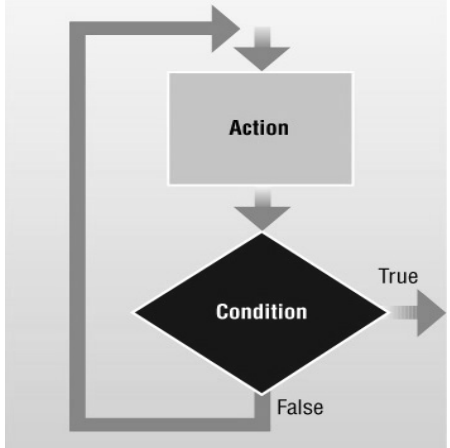
Selection Structure





Iteration Structure





Tabular Notation

		1	2	3	4	5	6	7	8
CONDITIONS	Background check results (S = Satisfactory, U = Unsatisfactory)	S	S	S	S	U	U	U	U
	References furnished?	Y	Y	N	N	Y	Y	N	N
	Passed credit check?	Y	N	Y	N	Y	N	Y	N
ACTIONS	Vendor approved	X							
	Vendor not approved					X	X	X	X
	Waiting for additional information		X	X	X				

Tabular Notation

- Provides a notation that translates actions and conditions (described in a processing narrative) into a tabular form.
  - The upper left-hand section contains a list of all conditions.
  - The lower left-hand section lists all actions that are possible based on the conditions.
  - The right-hand sections form a matrix that indicates condition combinations and the corresponding actions that will occur for a specific combination.

Program Design Language

- Program Design Language (PDL) is also called **Structured English**, or **Pseudocode**.

```
UPLOADING VENDOR INFORMATION
For each item containing vendor information, perform the following steps:
        If the item is not a computer file then
                Use the scanner to convert it into a file format.
        Copy the file into the Vendor Information folder on your hard disk.
Zip all new files in the Vendor Information folder into a single file.
Save the zipped file in a Web folder.
E-mail the Webmaster with the name of the zipped file.
```


Characteristics of Program Design Language

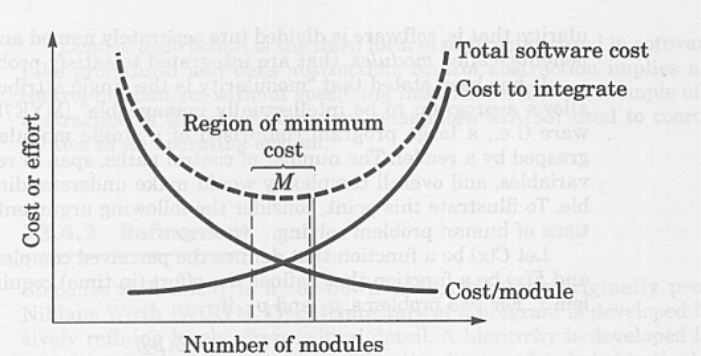
- A fixed syntax of keywords that provide for all structured constructs, data declaration, and modularity characteristics
- A free syntax of natural language that describes processing features
- Data declaration facilities that should include both simple (scalar, array) and complex (linked list or tree) data structures
- Subprogram definition and calling techniques that support various methods of interface description

Design Concepts

- Modularity
- Abstraction
- Control Hierarchy
- Data Structure
- Information Hiding
- Functional Independence

Modularity

- Software is divided into separately named an addressable components, called ***Modules***, that are integrated to satisfy problem requirements.



Modularity

- The graph shown refers to the relationship of Modularity and Software Cost, and is a useful tool to consider when modularity is to be implemented.
- The cost or effort drops as more modules are considered, but the cost to interface modules increases as we have more modules.
- We must take care to stay in the region of minimum cost
  - i.e. a balance between the number of modules and the cost or effort.

Modularity

- A module is defined as a sub-program that is invoked by another module.
- The statement are collectively referred to by a descriptive name called the module name
- A module must return to its caller
  - i.e. have a single entry and exit
- The module should be relatively small in size
- It should be easy to read, modify and use
- A module should preferably have a single function

Rationale for Modularity

- Allow large program to be written by several or different people
- Encourage creation of commonly used routines to be placed in library and/or be used by other programs
- Simplify overlay procedure of loading large program into main storage
- Provide more check point to measure progress
- Simplify design, making program easy to modify and reduce maintenance costs
- Provide a framework for more complete testing, easier to test
- Produces well-designed and more readable program

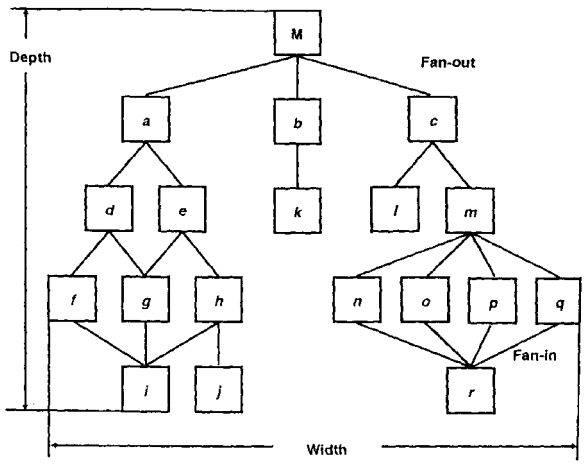
Rationale against Modularity

- Execution time, compilation/loading time and storage size requirements may be increase
  - This is a result of generally higher overhead costs in a software that has been broken into multiple parts
- Inter-module communication problems may be increased, due to the larger number of interfaces between parts of the software
- Demands more initial design time, since greater time would need to be spent in the architectural phase of design

Abstraction

- Abstraction permits one to concentrate on a problem at some level of generalization without regard to irrelevant low level details
- Use of abstraction also permits one to work with concepts and terms that are familiar, in the problem environment without having to transform them to an unfamiliar structure

Control Hierarchy



Control Hierarchy

- Control Hierarchy, represents the hierarchical organization of program components (modules) and implies a hierarchy of control.
  - Fan-Out
    - A measure of the Number of modules that are directly controlled by another module;
  - Fan-in
    - Indicates How Many modules directly control a given module.

Control Hierarchy

- Depth
  - Number of levels of control
- Width
  - Overall span of control
- Superordinate
  - A module that controls another module
- Subordinate
  - A module that is controlled by another module

Data Structure

- Data structure is a representation of the logical relationship among individual elements of data.

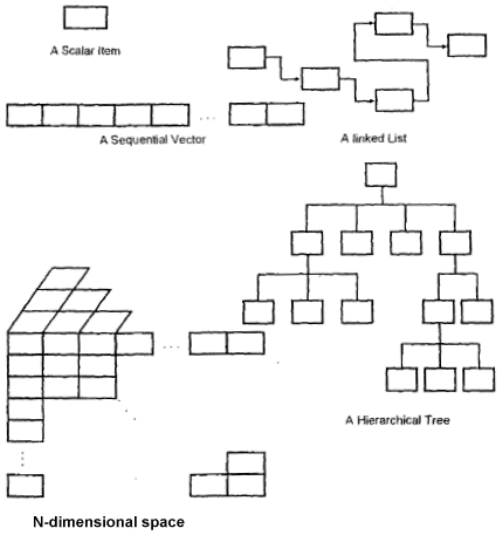
Examples of "Classic" Data Structures

- Scalar Item
  - Represents a single element of information that may be addressed by an identifier; and access may be achieved by specifying a single address in storage.
- Sequential Vector
  - When scalar items are organized as a list or contiguous group, a sequential vector is formed.
- N-dimensional Space
  - This is the result when the sequential vector is extended into an n-dimensional space.

Examples of "Classic" Data Structures

- Linked list
  - A data structure that organizes noncontiguous scalar items, vectors, or spaces in a manner (called nodes) that enables them to be processed as a list.
- Hierarchical Data Structure
  - Implemented using multilinked lists that contain scalar items, vectors, and possibly, n-dimensional spaces.

Data Structure



Information Hiding

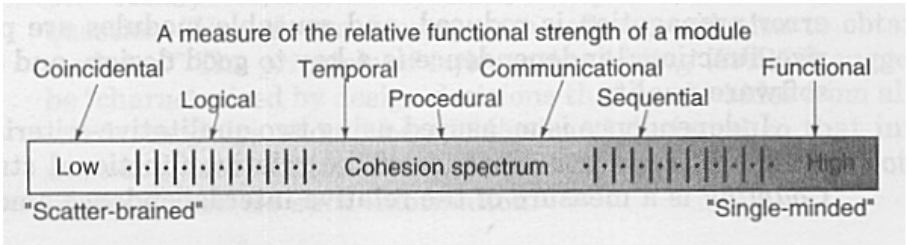
- Suggests that modules be characterized by design decisions that hides from the others
- Modules should be specified that information (procedure and data) contained within a module are inaccessible to other modules that have no need for such information
- Provides the greatest benefits when modifications are required during testing and later during software maintenance
  - Most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to other locations within the software

Functional Independence

- Developing modules with single-minded function and an aversion to excessive interaction with other modules.
  - Each module addresses a specific sub-function of requirements and has a simple interface when viewed from other parts of the program structure.
- Independence is measured using two qualitative criteria: cohesion and coupling.
  - **Cohesion**: Measures the relative degree of integration amongst elements in a module
  - **Coupling**: Measures the relative interdependence among modules.

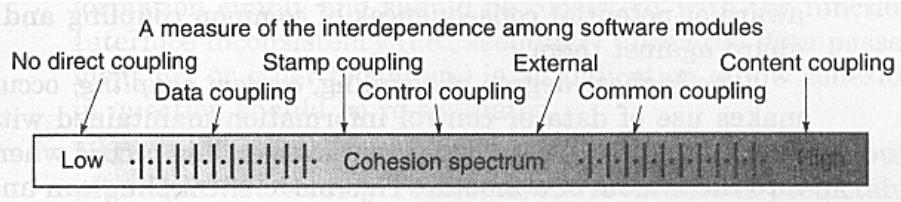
Cohesion

- A module with high cohesion is able to perform a single task within a software procedure, requiring little interaction with procedures being performed in other parts of the program.



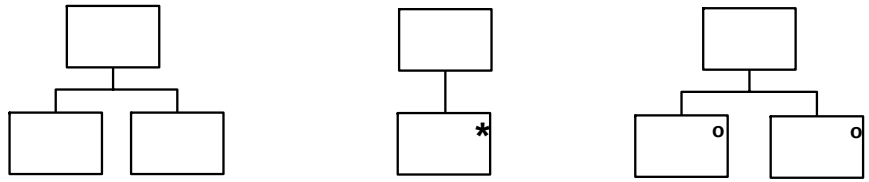
Coupling

- Coupling is a measure of interconnection among modules in a software structure.



Jackson Structured Programming

- Jackson Structured Programming essentially makes use of three basic notation for the three structured constructs: Sequence, Selection, and Repetition/Iteration



Sequence **Repetition** **Selection**

Demonstration

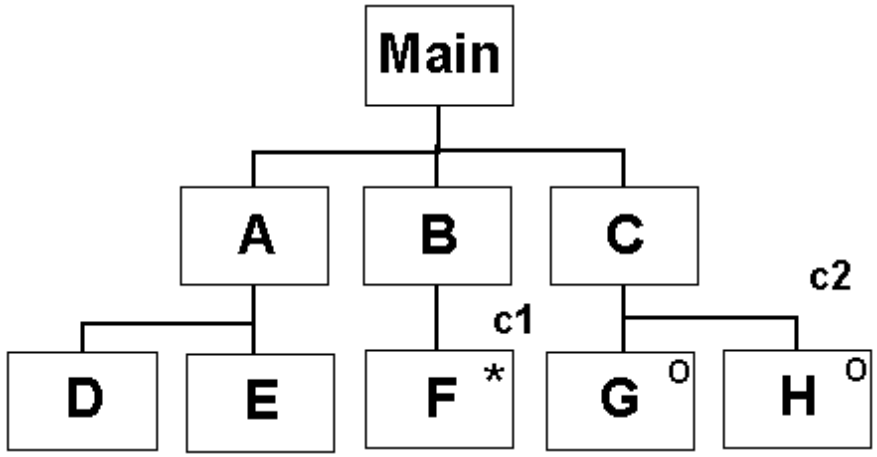
```
main()
{
A(); /* Run subroutine A first */
B(); /* Run subroutine B next */
C(); /* Run subroutine C last */
}

A()
{
D(); /* Run subroutines D and E in this order */
E();
}

B()
{
While condition c1 = true, Do F() /* Based on condition c1 */
}

C()
{
If condition c2 = true, do G(); /* Based on condition c2 */
Else do H();
}
```


Demonstration



Jackson Structured Programming (The Eight Steps)

1) Draw structure diagrams for each set of data such that the structure reflects the way in which the data is to be processed.
2) Identify points of correspondence of a one-for-one nature between components of individual data structures.
3) Produce a program structure diagram using the same notation as that used in data structures, and based on the data structures, combine them at the points of correspondence.
4) For each iteration and selection appearing in the program structure diagram, construct appropriate conditions.

Jackson Structured Programming (The Eight Steps)

5) Examine the specification and the conditions and from these draw up a list of basic program operations in plain language.
6) Allocate the conditions and operations to the appropriate components of the program structure.
7) Produce “schematic logic”, also known as pseudo code, from the program structure.
8) Implement the pseudo-code in a target high-level programming language.

Jackson Structured Programming (A Walk Through)

- A report is to be produced from a simple database file comprised of multiple records. This report will print-serially- every record from this file, and will also have the standard formatting features (e.g. logo and page numbering). Each page will contain up to 38 detail lines, and from there carry forward to the next page; and on each page there will be a "total" number in the page footer, reflecting either a culminate so far or the grand total number of detail lines.
- Making use of JSP, draw a reasonably detailed Program Structure, also populating your representation with sufficient operational and conditional details.

Walk Through – Step 0

- Draw a labeled sketch representing the output form, a report in this instance

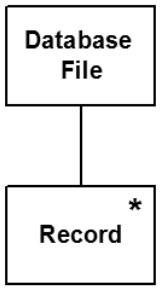
Logo	Date	
Detail line 1…		Page Header
Detail line 2…		
Detail line 3…		
---		Page Body

Detail line 38		
Page Number	Total Lines	Page Footer

Walk Through – Step 1

- Draw data structures using Jackson's notation to represent the input and output forms

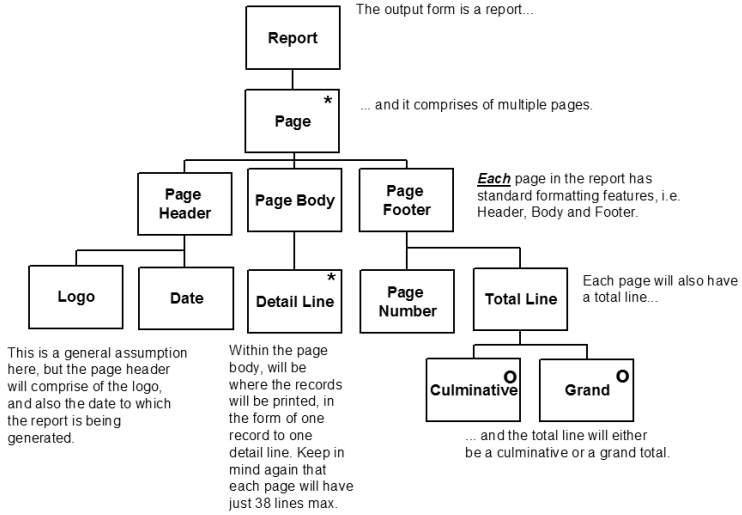
Input Data Structure



The input to this scenario is a database file...

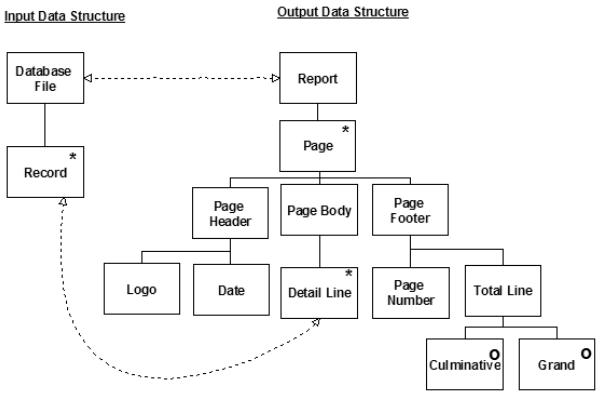
... and it comprises of multiple instances of "record"

Walk Through – Step 1



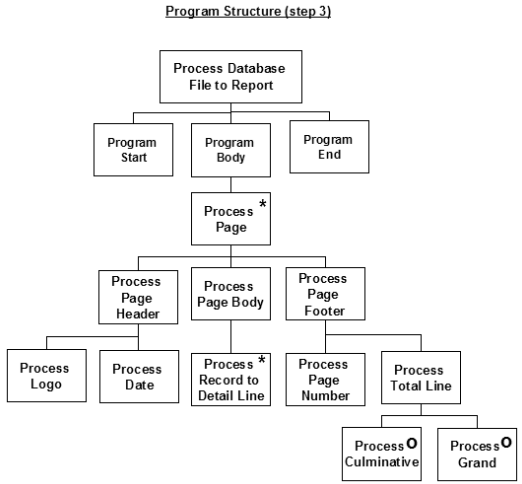
Walk Through – Step 2

- Where appropriate, draw lines of direct correspondence between the input and output data structure diagrams.



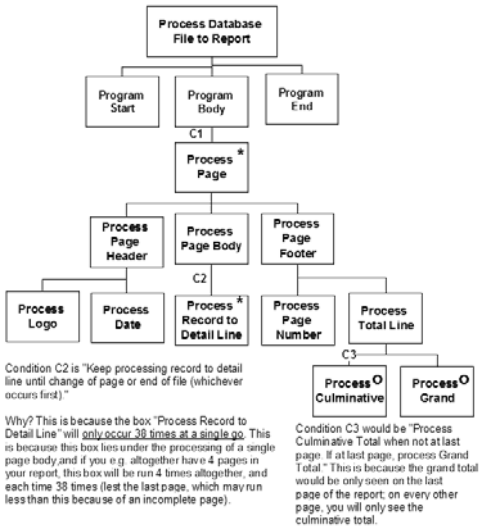
Walk Through – Step 3

- Based on the lines of correspondence, combine both data structure diagrams to produce a program structure.



Walk Through – Step 4

- Identify the conditional components, and design conditions for these components.



Walk Through – Step 5

- Based on program specifications, identify and design operations to reflect functional processing.

Walk Through – Step 6

- Allocate completed conditions and operations to the program structure.

Walk Through – Step 6