


## Design

---

- Design
  1. System design
  2. Program / Module design
- Properties of modules
  1. Module Coupling
  2. Module Strength / Cohesion
- Module Coupling : Describe the nature, direction and quantity of parameter(s) passed between modules
- Module Strength / Cohesion :How system function coded into modules

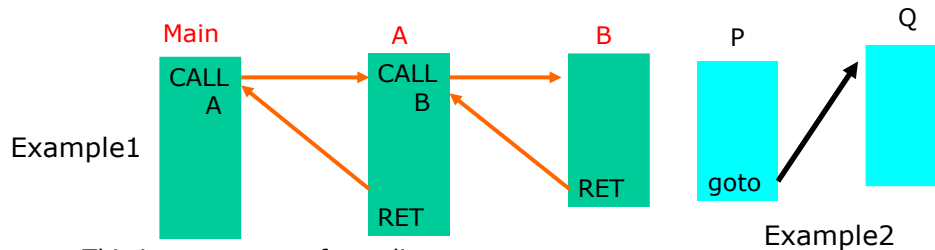
## Module Coupling

---

1. Content Coupling (Worst)
  2. Common Coupling
  3. External Coupling
  4. Control Coupling
  5. Stamp Coupling
  6. Data Coupling
  7. Zero Coupling (Best)
- 

## Content Coupling

- If one module makes a direct reference to the contents of another module
- One module to branch into another module



- This is worst type of coupling
- Potentially dangerous situation, always avoid it.
- There is no information hiding (i.e. called module become open to calling module)

## Common Coupling

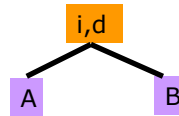
- If two or more modules refers to the same data structure or data element in a common environment
- Global variables
- Include common areas in user program or shared files
- Heterogeneous global data

Example1: Struct { int i,j,k;

double d;

char \*s1,\*s2;

} var1, var2;



## Common Coupling (Cont.)

---

- Example 2

- COMMON A, B, C  
X, Y, Z

- Example 3

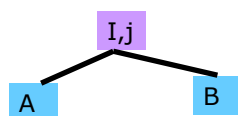
- PERFORM P1 THRU P8  
PERFORM P4 THRU P10

## External Coupling

---

- Special case of common coupling
- Homogeneous global data ( variables are same types)
- Example

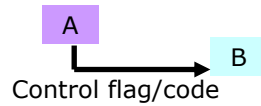
```
struct { int i, j, k, l;  
        char s1, s2; }
```



## Control Coupling

---

- Two module one of which would pass a **control flag** to another

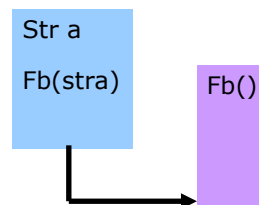


- Violates the principle of information hiding
- Calling module must know the method of operation of the called module

## Stamp Coupling

---

- It involves passing of heterogeneous local data which passes through the parameters
- Needed few elements of the entire structure or can stamp the entire structure
- Problem of overhead



## **Data Coupling**


---

- Pass values of data and return the result
- Practically best coupling



## **Zero Coupling**

---

- Means no coupling between two module
  - Practically this is impossible
  - All physical module must have some coupling however weak is
- 


## Module Strength / Cohesion

---

- Internal activity of a single module
- In general one module should perform one single task
- Module which try to perform many task – validate input, process data, output results – are difficult to define and maintain
- By their very nature, their complexity will leads to coupling problems
- Hence, good coupling should help enforce for good cohesion

## Module Strength / Cohesion

---

- |   |          |
|---|----------|
| 1. Coincidental Strength / Cohesion         | (Worst ) |
| 2. Logical Strength / Cohesion              |          |
| 3. Classical / Temporal Strength / Cohesion |          |
| 4. Procedural Strength / Cohesion           |          |
| 5. Communicational Strength / Cohesion      |          |
| 6. Functional Strength / Cohesion           |          |
| 7. Informational Strength / Cohesion        | (Best )  |
- 

## Coincidental Strength / Cohesion

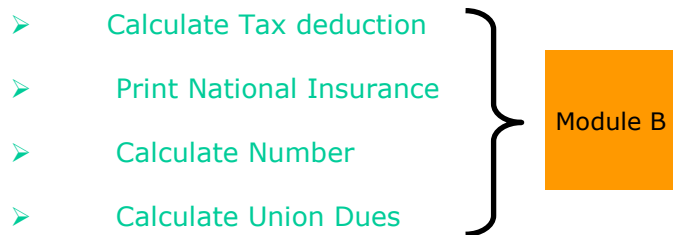
---

- No strength at all
- Multiple task, completely unrelated
- Only one way to describe module task is by describing its logic step-by-step
- Sequence of commands, replacing these sequence by modules
- This is very difficult to create in practice ( only arbitrary random grouping of the lines of the program code could perhaps result in such strength )

## Coincidental Strength / Cohesion (Cont.)

---

### Example



## Logical Strength / Cohesion

---

- A module that perform similar tasks, which are related logically is called Logical Cohesiveness
  - i.e. All editing, produces all output regardless of type
  - Similar edit check may be made on more than one data item i.e. **date of transaction**
- A better way is to construct a DATE\_CHECK module and call this module whenever a date check is necessary\
- Characterized
  - Single function
  - uniform function interface
  - Information hiding

## Logical Strength / Cohesion (Cont.)

---

- **Example** Table\_Operate ( **FNCODE**, ARG1, ARG2,ARG3)

0:Clear

1:Add

2>Delete

3:Search

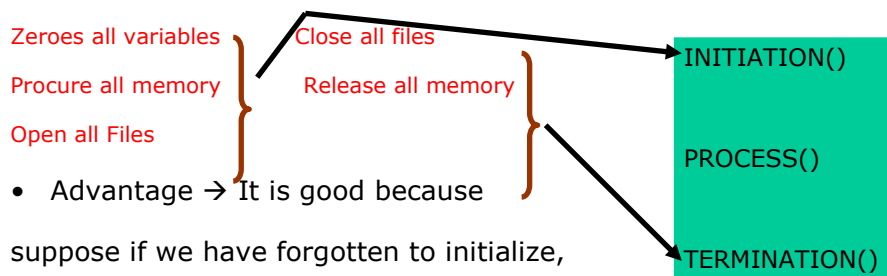


- Disadvantage
  - To add some extra feature, we have to add some extra argument
  - A single change, would require a number of changes inside the module which is very costly



## Classical/Temporal Strength / Cohesion

- Very similar to logical strength
- All functions related to time are grouped into one module
- A better way is to construct a DATE\_CHECK module and call this module whenever a date check is necessary



- Advantage → It is good because suppose if we have forgotten to initialize, we have only to scan process part

## Classical/Temporal Strength / Cohesion (Cont.)

- Disadvantage → Addition of a new file we have to be change all the functions

## Procedural Strength / Cohesion

- When a flowchart ( we define system functions) structure is divided into a number of section and each section is represent one module
- It is some time problem specific and domain specific
- Example

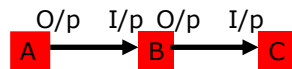
A branching application problem :

```
if (tx_code ==5)
{
    read_next_tx();
    print_E15();
    update_bal();
    Clear_buffer();
}
```

4 functions are unrelated they are application specific

## Communicational Strength / Cohesion (Cont.)

- When the processes are communicate with each other, are included in the same module
- Concerned with a data structure / file
  - Module read the file, process it and write output back to the file



## **Functional Strength / Cohesion**


---

- Practically best strength/cohesion
- When a module perform a single unambiguous task
- If we are able to conclude at a single statement describing about the activity of the module, then we have achieved functional strength



## **Informational Strength / Cohesion**

---

- Practically not possible to implement
  - Multiple functions related by the same data structure
  - Multiple entry points corresponding to different functions
  - No jump between different entry points
  - Advantage → Information Hiding
- 

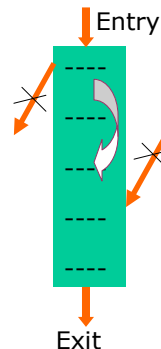
## Program Design

### Modular / Structured program design

- No goto
- Structure program uses structured control structures to improve program clarity and maintenance
- Top\_down development
- There are three basic control structure
  - Sequence
  - Decision/Selection
  - Iteration/Looping

## Program Design (Cont.)

- Properties of Structured programming
  - Single entry point and single exit point
  - Program have no. jumps from outside to inside except at the beginning
  - Program have no. jumps from inside to outside except at the end
  - Jump inside should be avoided
- Ways of removing goto's from program  
(Use Loops i.e. While, do\_while, Repeat Until)
- Convert Unstructured programming to equivalent structured programming



## Program Design (Cont.)

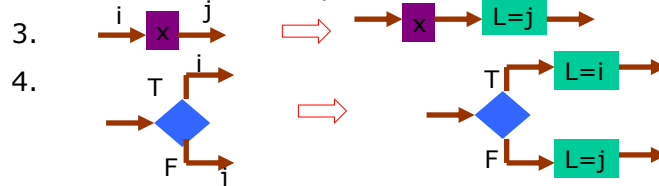
- Methods
  1. Formal
  2. Informal
- Informal method consist of two techniques
  1. Use of code duplication
  2. Use of Boolean flag(s)
- Formal Method
  - Use Mill's theorem

## Program Design (Cont.)

- Mill's theorem
  - There exists a corresponding equivalent structured programming given by number of nodes and flow for a proper unstructured programming with n nodes (process and decision )

- Rules

1. Entry node/flow marked by '1'
2. Exit flow marked by '0'



## Program Design (Cont.)

- Rules

5. General structure

