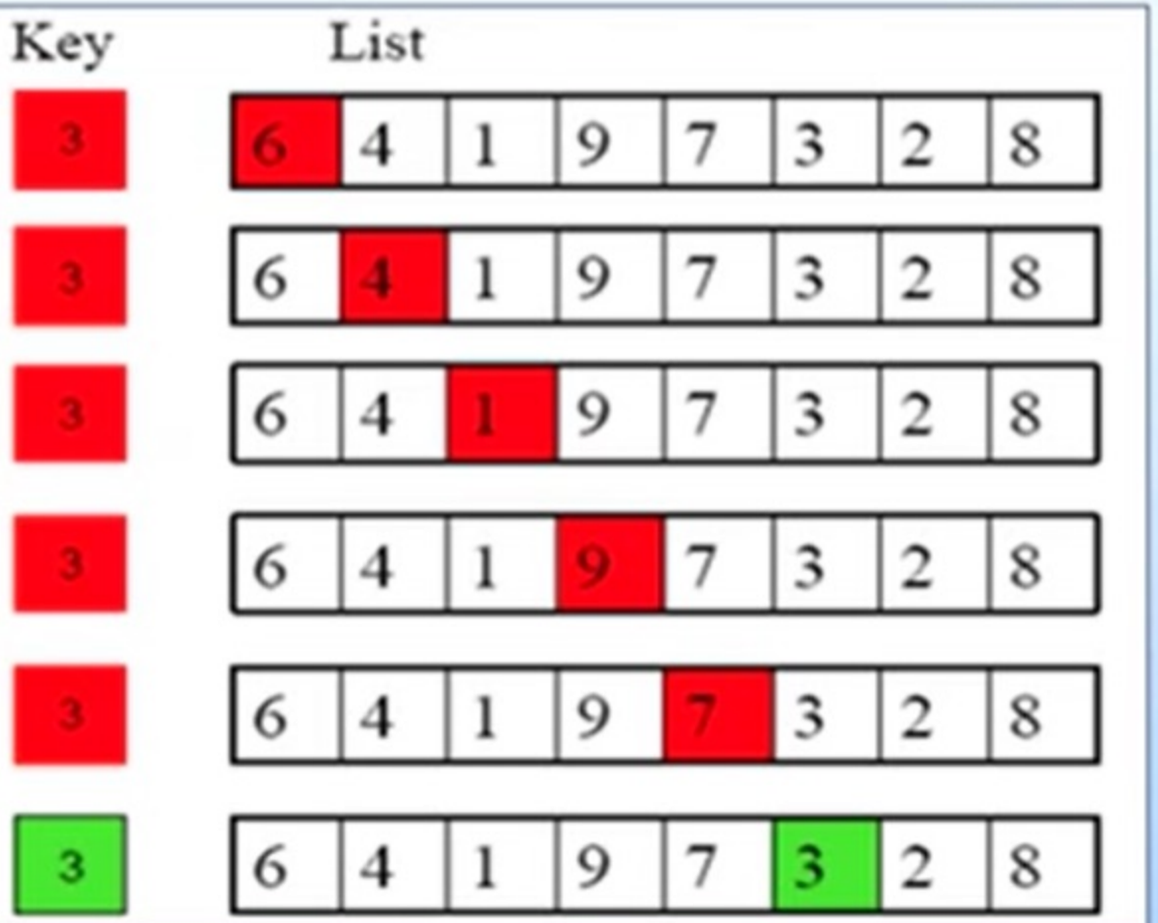
Linear Search Algorithm

If an element is to be retrieved from a specific location in an array, the array has to be traversed from the first position until the element is found.

```
Step 1 : flag = notfound , I=0
Step 2 : If I > n – 1 and flag = found
     Go to 5
Step 3 : If array[I] =  = element
     flag = found
Step 4 : I = I + I
Step 5 : if (flag = = found )
     Print element found at position I+1
          Else
          Print element not found
 Step 6 : Stop
```



Binary Search Algorithm

Searching technique for an element from a sorted array

Divide the array in two parts by finding the middle value in iteration

If the value of the search element is lesser than the item in the middle of this interval, then split the left interval in half. Otherwise split the right interval in half.

Search for the element in the new interval and continue doing this until the search value is found or until the interval is not found to contain the search value.

ALGORITHM

1. Initialize low =0, high = n-1
2. While low <= high
3. mid = $\lfloor (low+high)/2 \rfloor$
4. If a[mid] == *Item*
5. Set Pos = mid
6. Break and Jump to step 10
7. Else if Item < a[mid]
8. high = mid -1
9. Else low =mid +1
10. If Pos < 0
11. Print "Element is not found"
12. Else Print Pos

Bubble Sort - Algorithm

TEKNOTURF

Let A be a linear array of n numbers. Swap is a temporary variable for swapping (or interchange) the position of the numbers

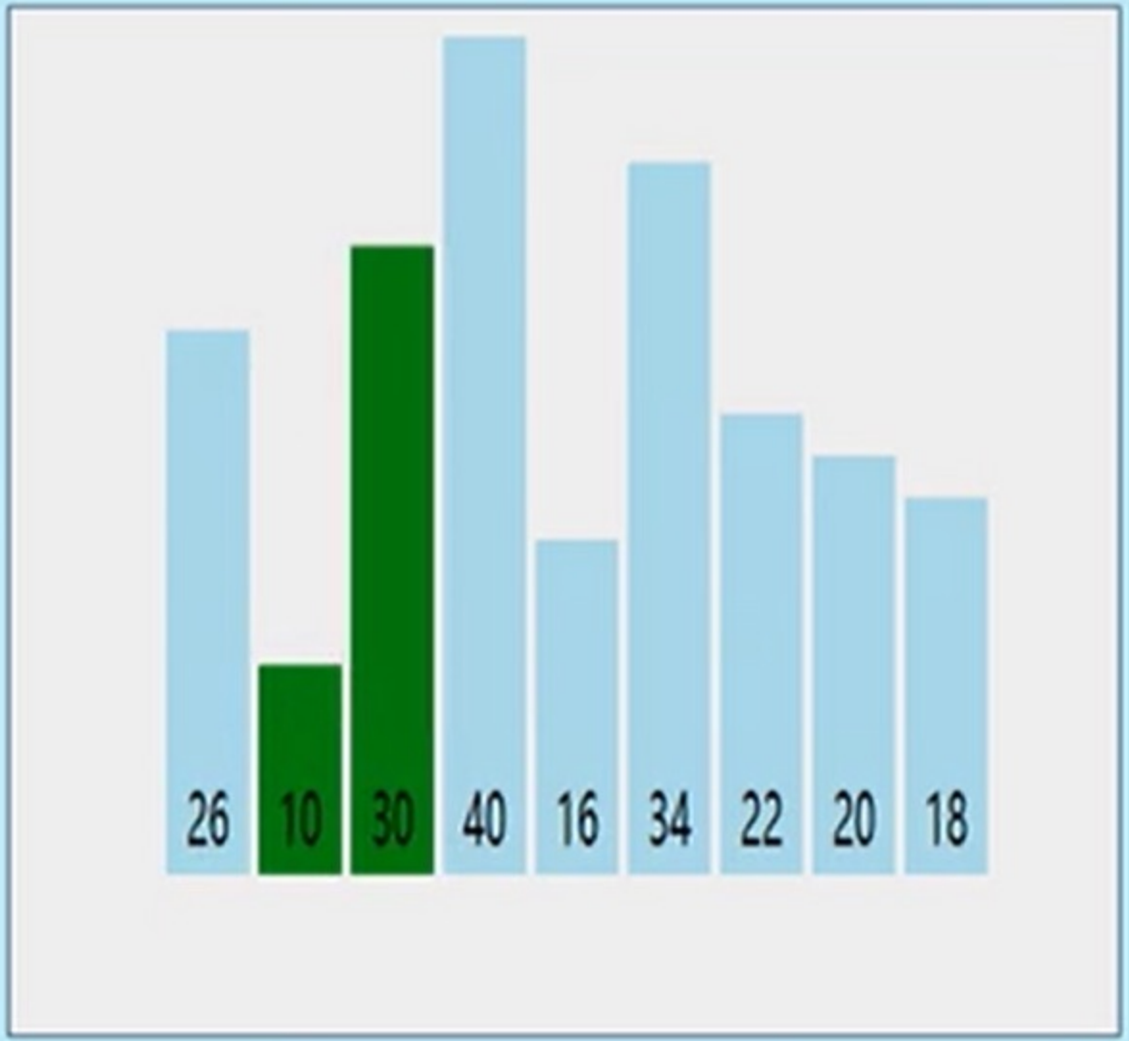
1. Input n numbers of an array A
2. Initialize i = 0 and repeat through step 4 if (i < n)
3. Initialize j = 0 and repeat through step 4 if (j < n – i – 1)
4. If (A[j] > A[j + 1])

(a) Swap = A[j]

(b) A[j] = A[j + 1]

(c) A[j + 1] = Swap

5. Display the sorted numbers of array A
6. Exit.

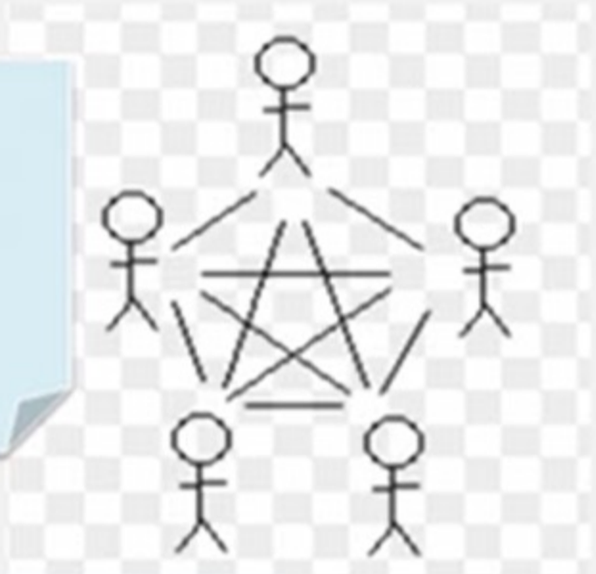


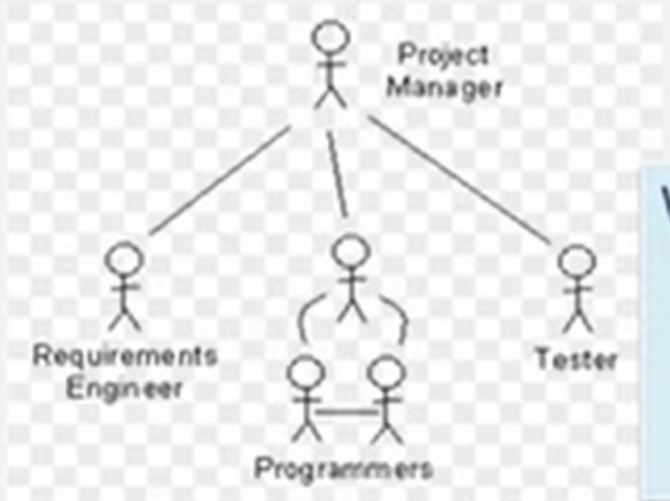
Coupling and Cohesion

TEKNOTURF

- Modules in themselves are not "good" – We must design them to have good properties
- Properties of good Design
  - Component independence
  - Fault prevention and fault tolerance
  - Design for change

With poor module design:
- Hard to understand
- Hard to locate faults
- Difficult to extend or enhance



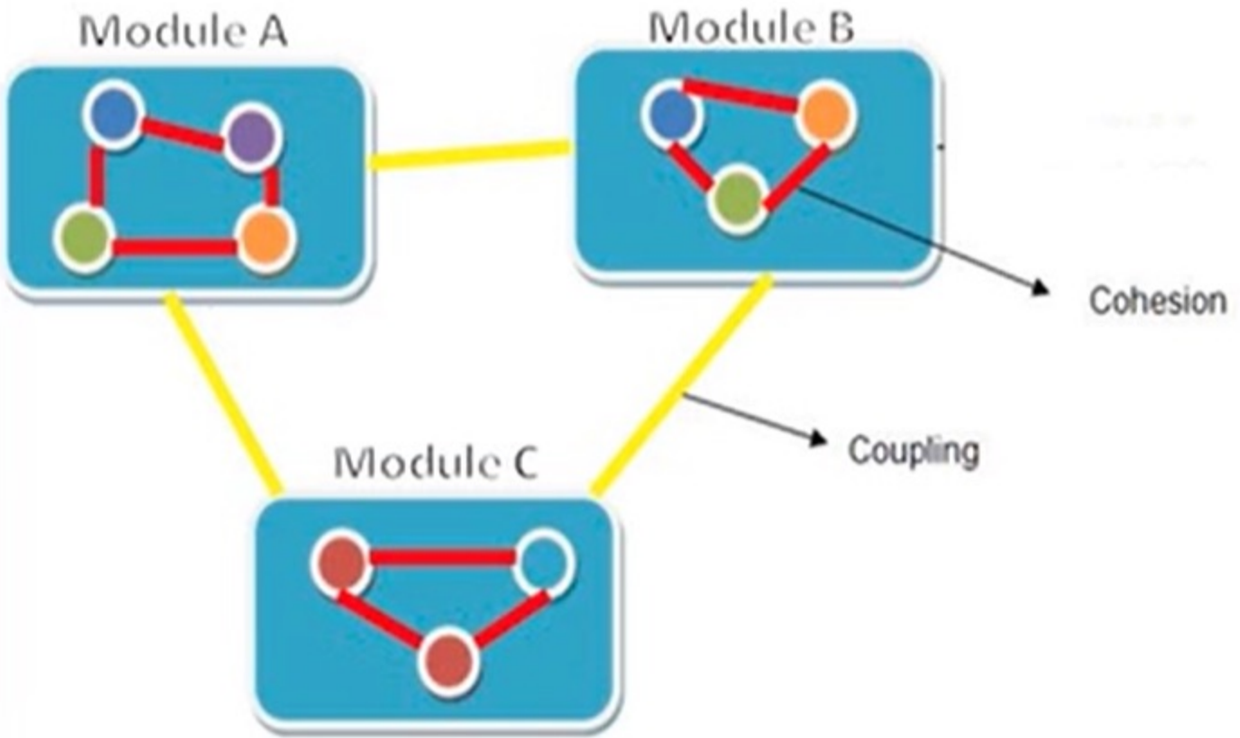


With good module design:
- Maximal relationships within modules (cohesion)
- Minimal relationships between modules (coupling)
- This is the main contribution of structured design

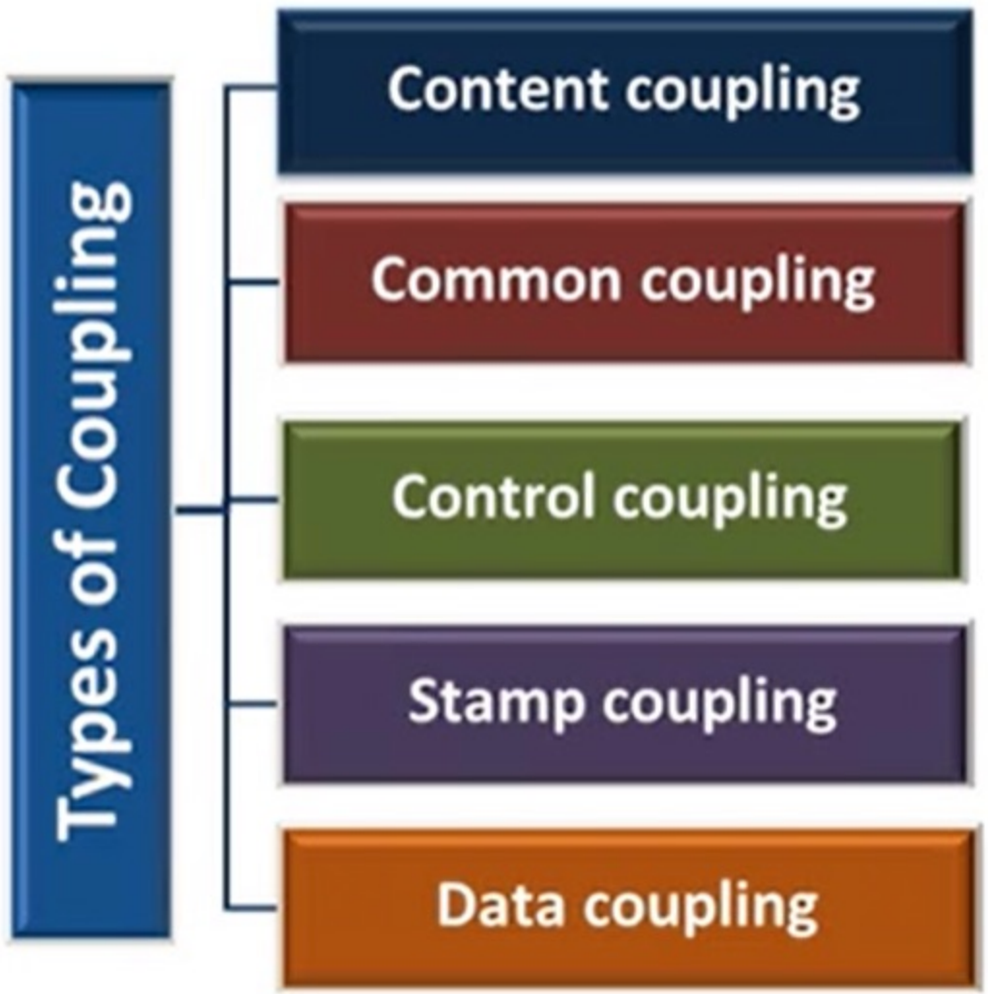
Coupling and Cohesion

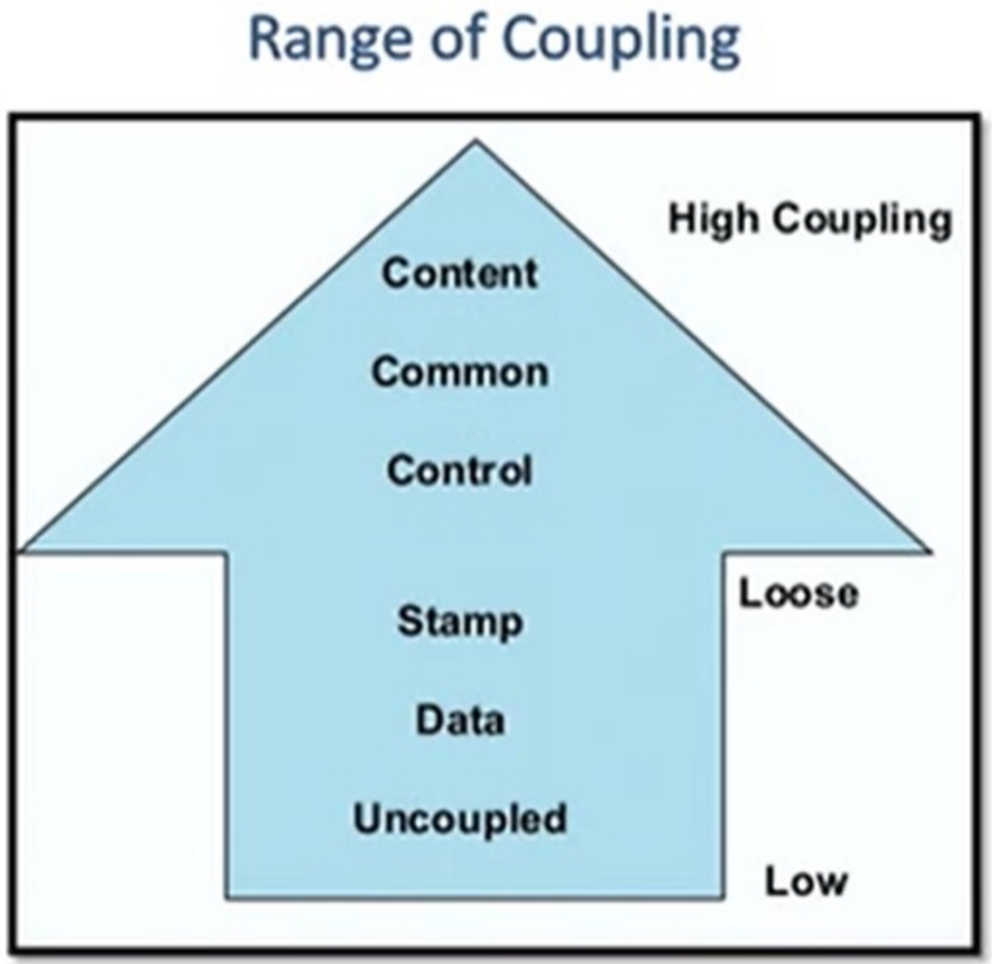
TEKNOTURF

- The goal of design is to divide the system into modules and assign responsibilities among the components so that they have
  - High cohesion within the modules
  - Loose coupling between modules
- The principle of coupling and cohesion are the most important design principles



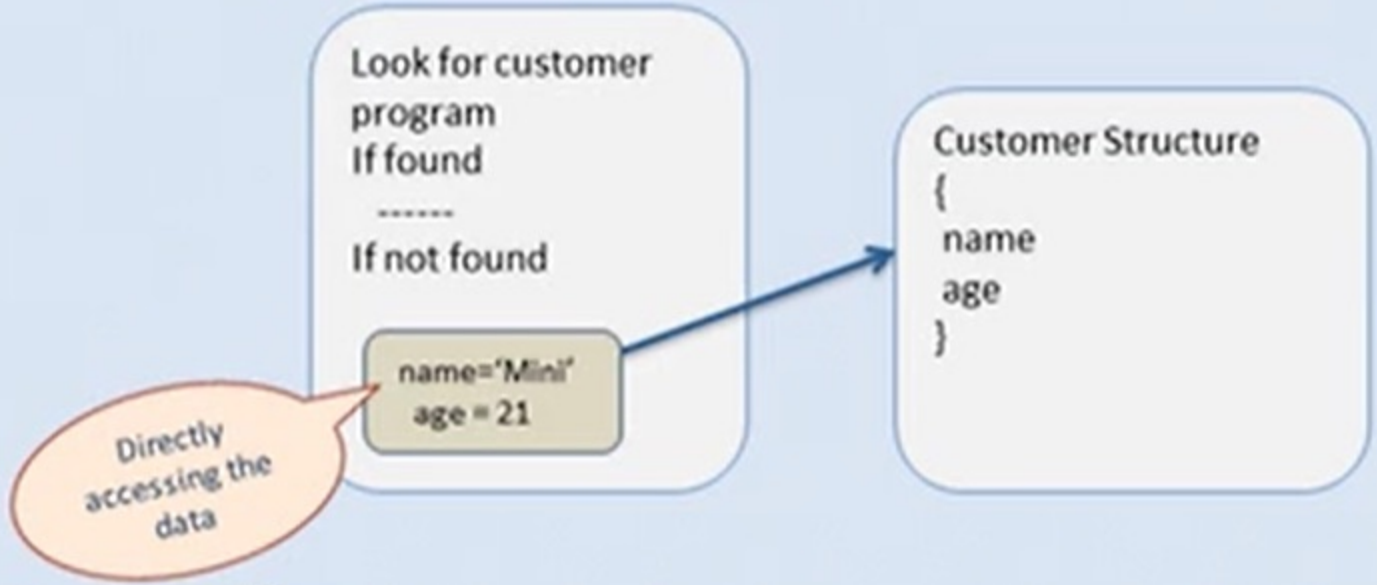
Types of coupling





Content coupling

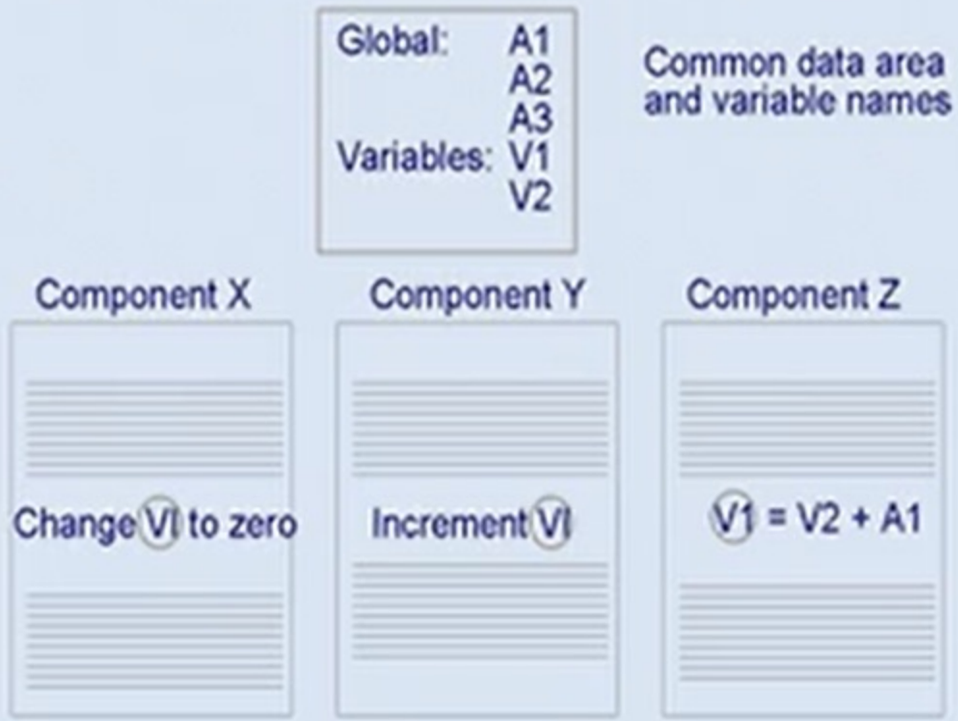
- Occurs when one component modifies an internal data item in another component, or when one component branches into the middle of another component



- To reduce content coupling
  - Hide the data so that it can be accessed only by calling the method that can access or modify the data

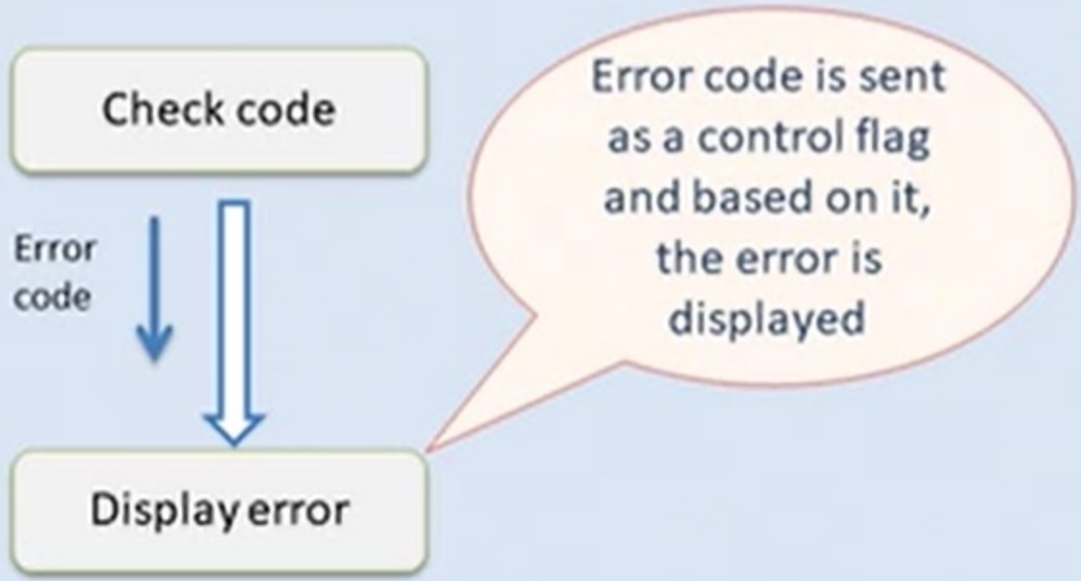
Common Coupling

- Two modules have write access to the same global data



Control coupling

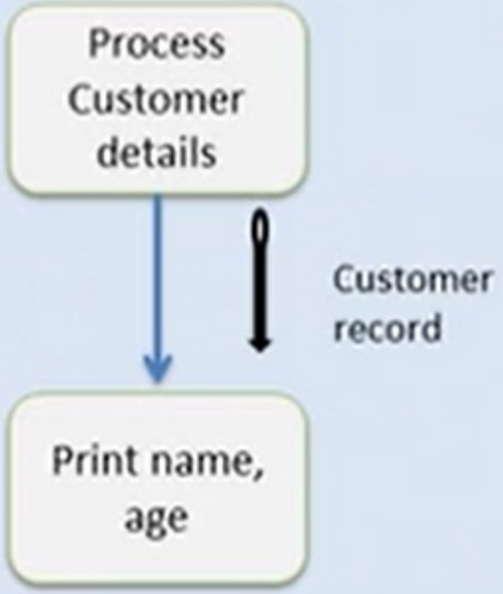
- One module passes an element of control to the other
- It is impossible for the controlled module to function without some direction from the controlling module



Types of Coupling

Stamp Coupling

- Data structure is passed as parameter, but the called module operates on only some of individual components



Data Coupling

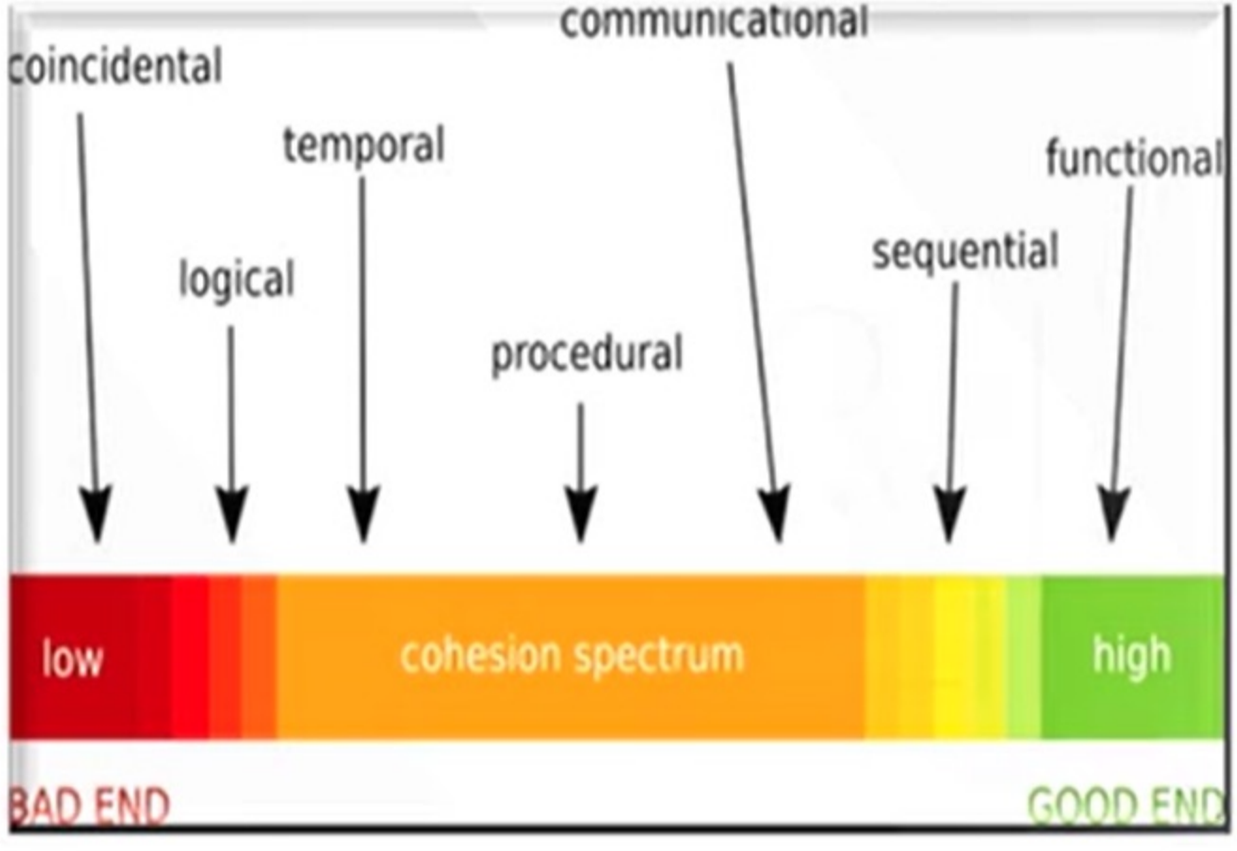
- Every argument is either a simple argument or a data structure in which all elements are used by the called module

Types of Cohesion

TEKNOTURF

Types of Cohesion

- Co-incidental Cohesion — Worst Cohesion
- Logical Cohesion
- Temporal Cohesion
- Procedural Cohesion
- Communicational Cohesion
- Sequential Cohesion
- Functional Cohesion — Best Cohesion



Types of Cohesion

TEKNOTURF

Logical Cohesion

Here, elements perform similar tasks and the activities to be executed are chosen from outside the module.
Operations are related, but the functions are significantly different.

Temporal Cohesion

Module's data and functions are related because they are used at the same time in an execution.
Elements are grouped by when they are processed.

Module

```
Module(data[] , type)
   if type is bar
        display as bar chart
   else if type is pie-chart
        display as pie-chart
   else if type is graph
        display as graph
    .....
```

Task is the same which is "display", but the 'how to be displayed' is decided by the calling function.

Process Error Module

```
processError()
   Release the database connection
   Open error file
   Write error log
   Send error message to user
```

All the activities of the module are performed when an error occurs in the software.

Types of Cohesion

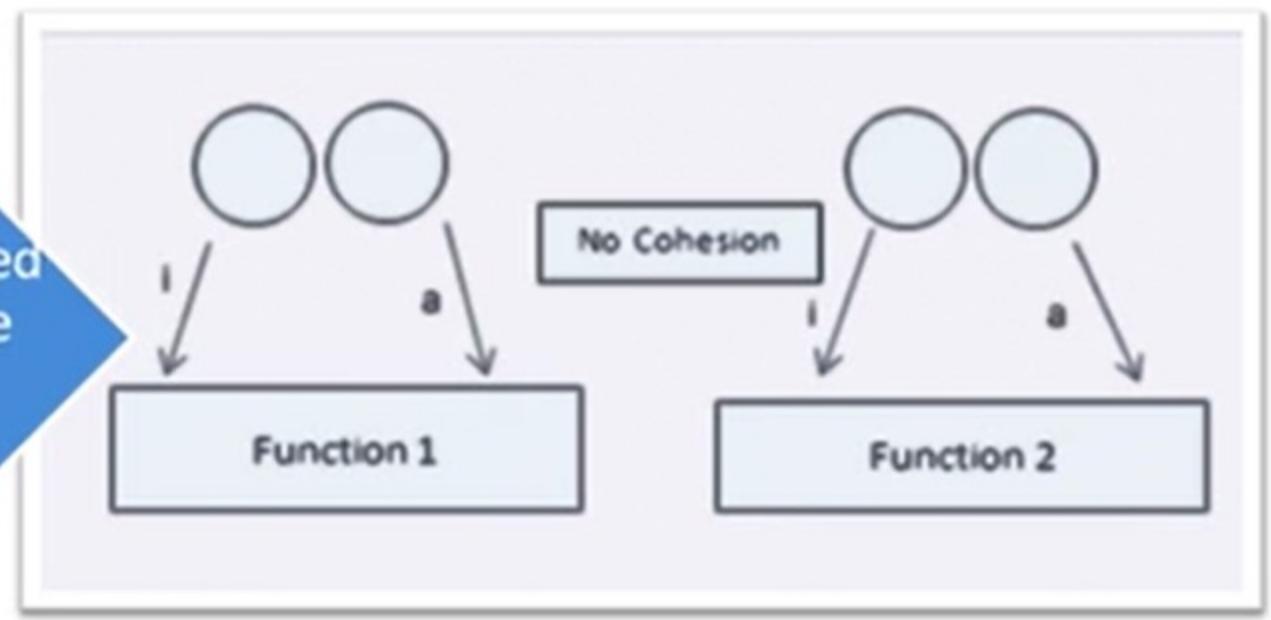
Co-incidental Cohesion

Module performs multiple, completely unrelated actions

Module 1

- Process customer details
- Calculate total sales
- Perform transaction deletion
- Read sales details

Has unrelated tasks inside module



Types of Cohesion

TEKNOTURF

Procedural Cohesion

Similar to temporal, and functions pertain to some related action or purpose

Module

processCustomer()

create customer id

add new record to customer file

display customer id to user

All the activities of the module related to creating a customer record

.

Communicational Cohesion

Module which has activities executed sequentially and work on same data

Module

processCustomer(customer id)
get customer details
get customer purchaser details
display customer id, customer name, purchase items

All the activities of the module act on the customer id

Types of Cohesion

TEKNOTURF

Sequential Cohesion

Elements are involved in activities such that output data from one activity becomes input data to the next activity

CalculateGrade Module

```
CalculateGrade(marks) {
  calculateSum
        ↓   sum
  calculateAverage
        ↓   average
  calculateGrade
}
```

The output of calculateSum is given as input to calculateAverage

Functional Cohesion

Functionally cohesive module performs exactly one action.
Highly recommended Cohesion

Example: `convertCelciusToFarenheit`

Advantages

- More reusable
- Easier corrective maintenance
  - Fault isolation
  - Reduced regression faults
- Easier to extend product

SYMBOLS IN FLOWCHART

START/END

CONNECTOR

INPUT/OUTPUT

FLOW DIRECTION

PROCESS

DECISION