Software Design for Embedded Systems

課程：嵌入式系統與軟體工程
開發學校：台大電機系
王勝德 教授

Outline

- Design Concept
- Design process
- Design methods
- Design quality
- Design Documentation

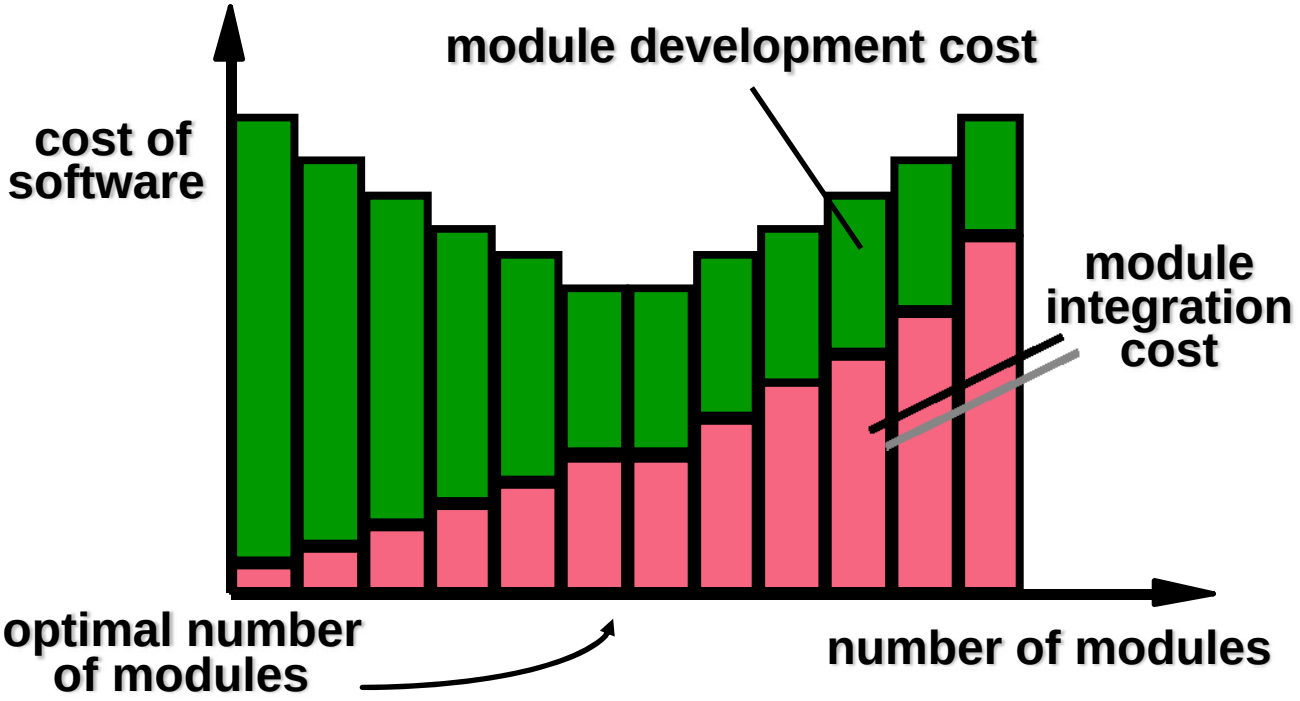
Design Concepts

- **Abstraction**
  - data, procedure
- **Refinement**
  - elaboration of detail for all abstractions
- **Modularity**
  - compartmentalization of data and function
  - Procedure: the algorithms that achieve the function
  - Information hiding: interface design
- **Architecture**
  - Overall structure of the software, Structural properties, Styles and patterns

Modularity

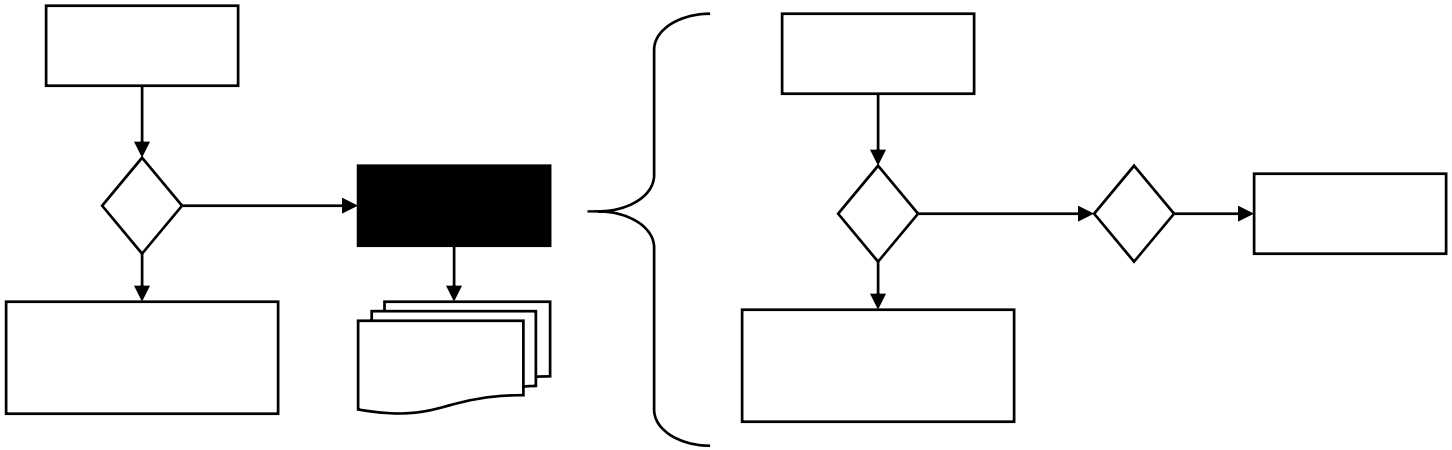
- How can we evaluate an effective modularity
  - Modular decomposability
  - Modular composability
  - Modular understandability
  - Modular continuity
  - Modular protection

Modularity: Trade Off

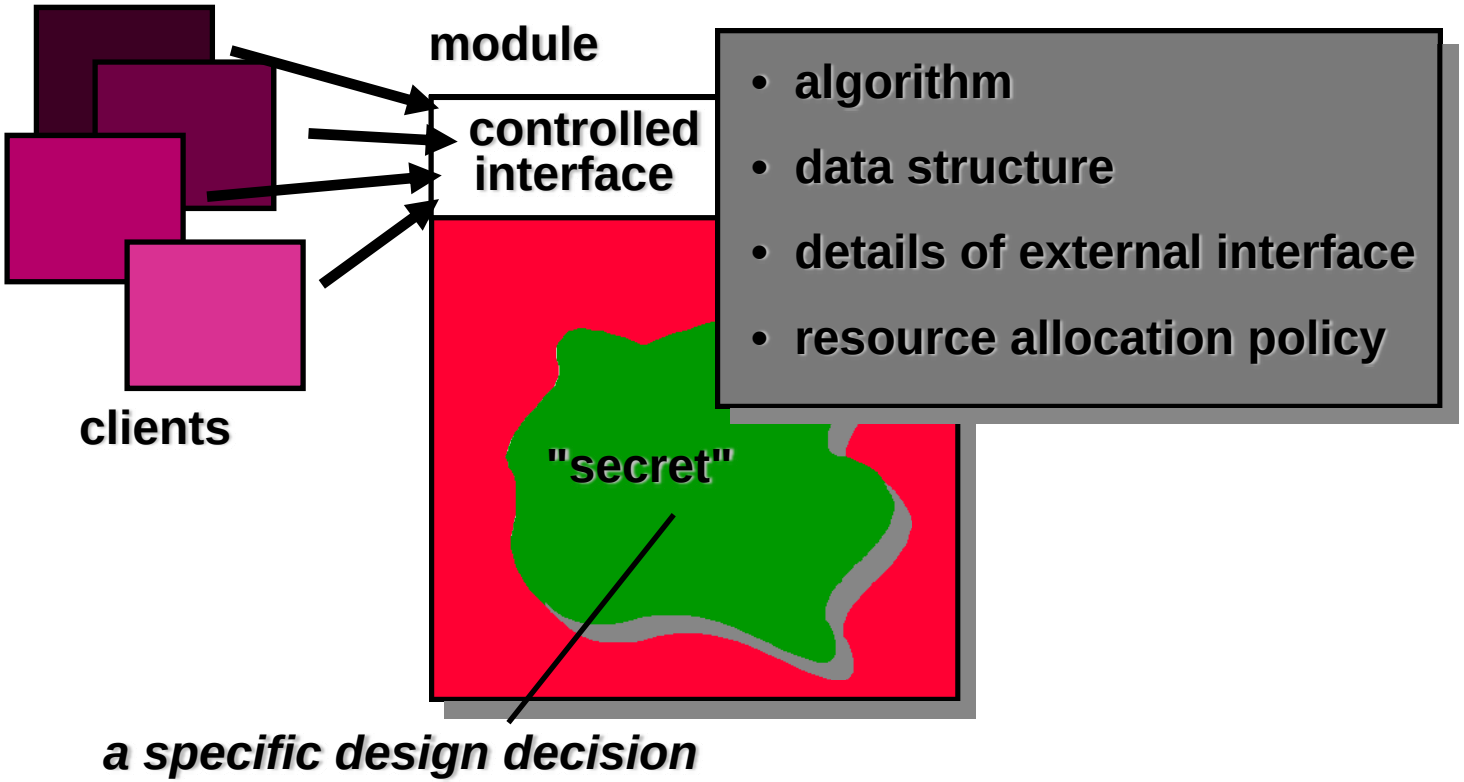


Procedure

- Software procedure focuses on the processing details of each module individually
- Procedure must provide a precise specification processing, including sequence of events, exact decision points, repetitive operations, and even data organization and structure
- Procedure should be layered



Information Hiding



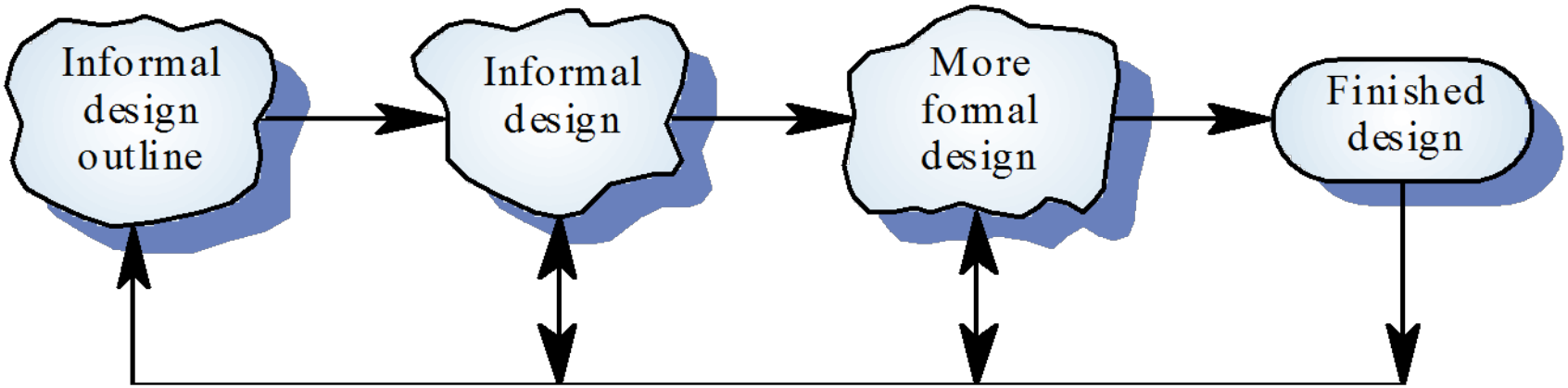
Why Information Hiding?

- Reduces the likelihood of “side effects”
- Limits the global impact of **local design decisions**
- Emphasizes communication through **controlled** interfaces
- Discourages the use of global data
- Leads to encapsulation—an attribute of high quality design
- Results in higher quality software

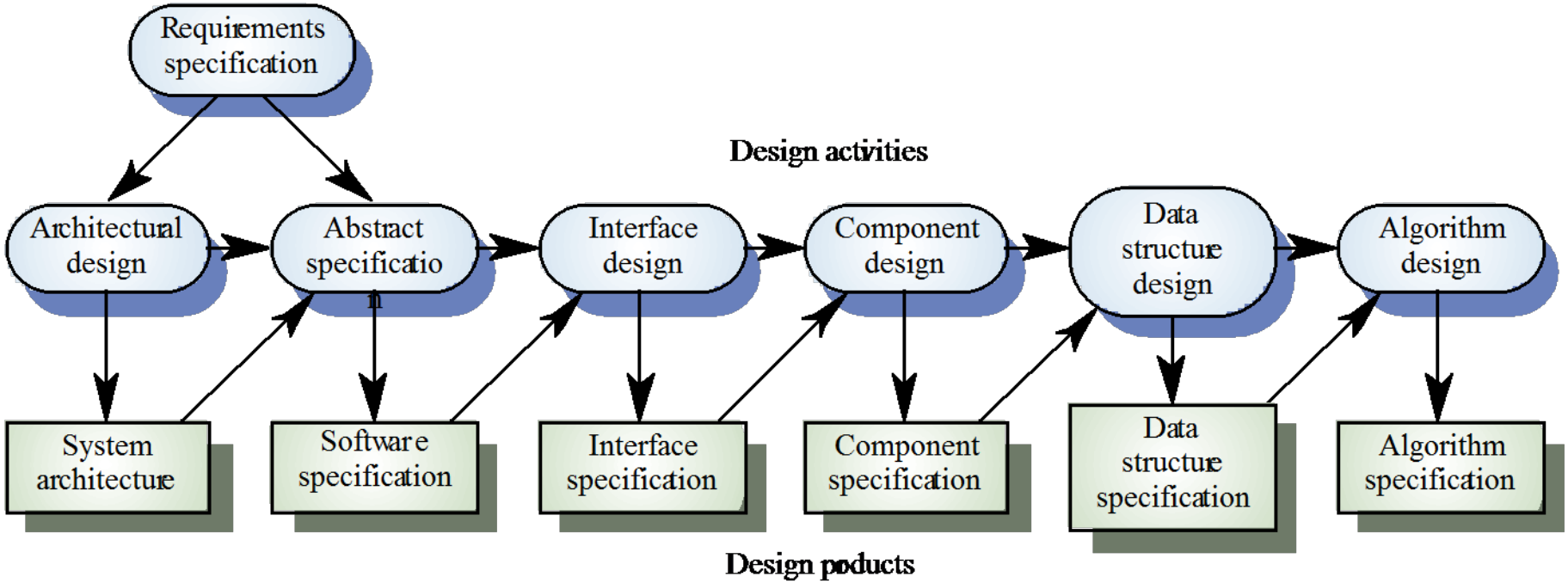
Design Process

Design process

- The system should be described at several different levels of abstraction
- Design takes place in overlapping stages. It is artificial to separate it into distinct phases but some separation is usually necessary



Phases in the design process



Design phases

- *Architectural design* Identify sub-systems
- *Abstract specification* Specify sub-systems
- *Interface design* Describe sub-system interfaces
- *Component design* Decompose sub-systems into components
- *Data structure design* Design data structures to hold problem data
- *Algorithm design* Design algorithms for problem functions

Design Methods

The notion that good [design] techniques restrict creativity is like saying that an artist can paint without learning the details of form or a musician does not need knowledge of music theory

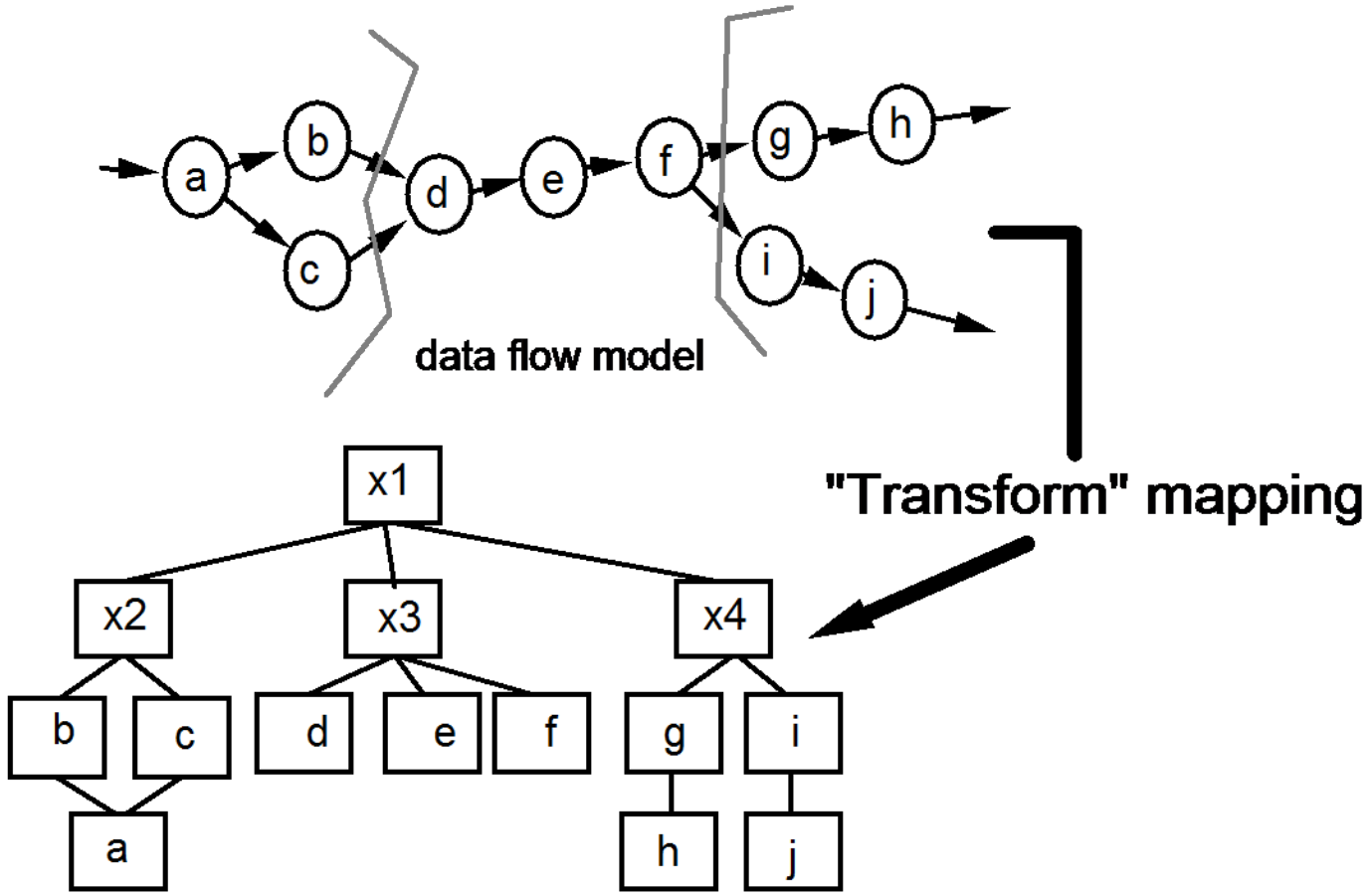
Marvin Zelkowitz et al.

Design Methods

- Decision tables
- ER
- Flowcharts
- JSD, Jackson structured programming
- OBJ
- OOD
- PDL, program design language
- Petri nets
- SA/SD
- State chart
- …

Structure chart

- Originated in the early 1970 with Yourdon and Constantine
- In the design phase, the data flow diagrams are transformed into a collection of modules
  - Result: structure chart
- No strict rules to transform a DFD into a structure chart
- How to choose the top level module in the structure chart?
  - Many data-processing systems are essentially transform-centered
  - Picking the central transform as the top level module


a
b
c
d
e
f
g
h
i
j
data flow model
"Transform" mapping
x1
x2
x3
x4
b
c
d
e
f
g
i
a
h
j

Design Quality

Design Quality

- Design quality is an elusive concept. Quality depends on specific organizational priorities
- A 'good' design may be the most efficient, the cheapest, the most maintainable, the most reliable, etc.
- The attributes discussed here are concerned with the **maintainability** of the design
- Quality characteristics are equally applicable to function-oriented and object-oriented designs

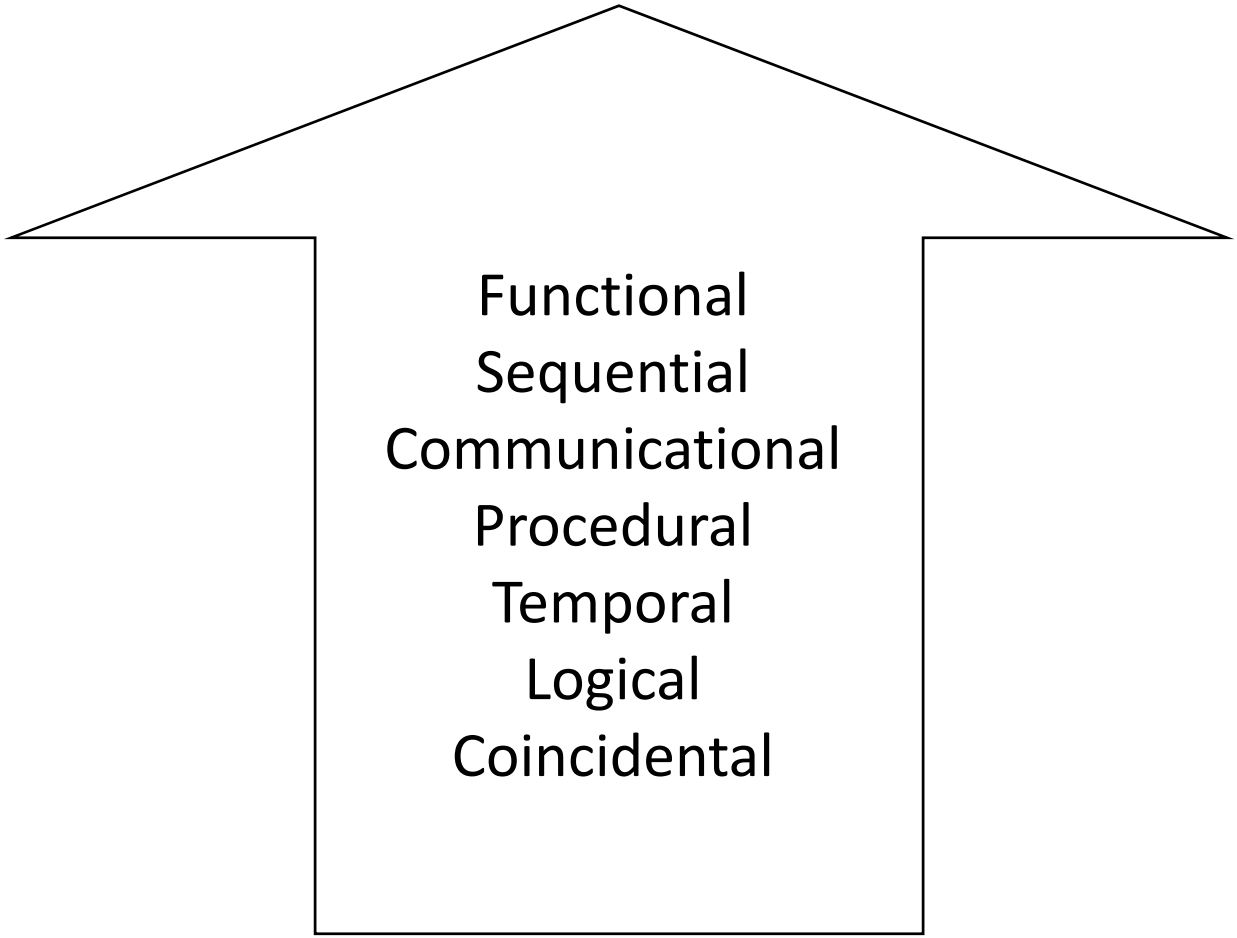
Component Independence

- To make the components independent of one another
  - Easier to understand how a component works if it is not intricately tied to others
  - Easier to modify an independent component
  - Easier to trace and fix bugs

- To recognize and measure the degree of component independence in a design (Yourdon 1978)
  - Coupling
  - Cohesion

Cohesion

- Cohesion refers to the internal "**glue**" with which a module is constructed
- The more cohesive a module, the more **related** are the internal parts of the module to each other and to its overall purpose
- A component is cohesive if all elements of the module are directed toward an essential for performing the same task
- Cohesion is a desirable design component attribute as when a change has to be made, it is **localized** in a single cohesive component


Functional
Sequential
Communicational
Procedural
Temporal
Logical
Coincidental

Cohesion levels

- Coincidental cohesion (weak)
  - Parts of a component are simply bundled together
- Logical association (weak)
  - Components which perform similar functions are grouped
- Temporal cohesion (weak)
  - Components which are activated at the same time are grouped
- Procedural cohesion (weak)
  - The elements in a component make up a single control sequence

Cohesion levels

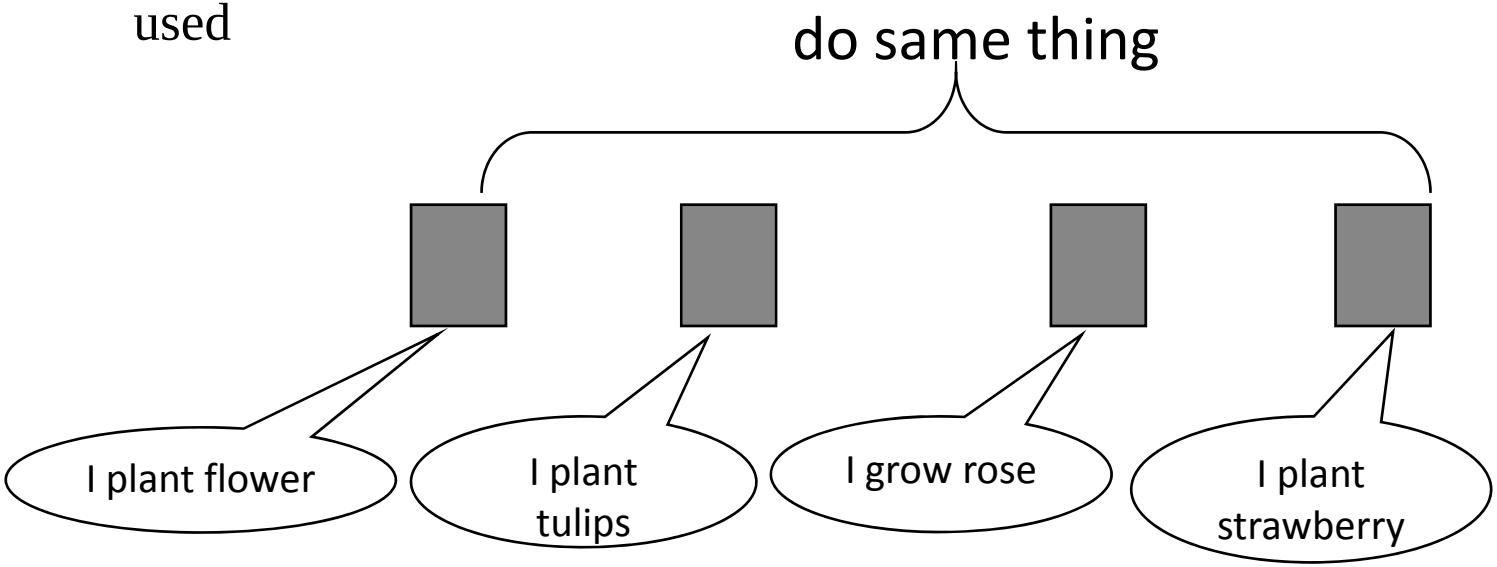
- Communicational cohesion (medium)
  - All the elements of a component operate on the same input or produce the same output
- Sequential cohesion (medium)
  - The output for one part of a component is the input to another part
- Functional cohesion (strong)
  - Each part of a component is necessary for the execution of a single function
- Object cohesion (strong)
  - Each operation provides functionality which allows object attributes to be modified or inspected

Coincidental cohesion

- This is when elements are formed into a module simply because they happen to fall together,
  - Unrelated functions, processes or data are related to one another for reasons of convenience
  - A component contains
    - Checking a user's security classification
    - Printing this week's payroll

Logical cohesion

- Elements of a module are grouped together according to a certain class of activity - **they do the same kind of thing**.
  - **Read-All-Files** is a module which does all the file reads.
  - Regardless of where the input is coming from or how it will be used



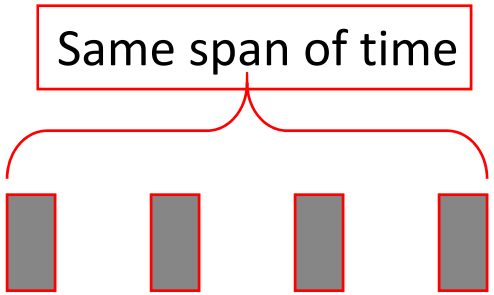
An Example of logically Cohesion

```
CASE file-code OF

        1:      Read customer transaction record
                If not EOF
                        increment customer-transaction-count
                end-if

        2:      Read customer master record
                If not EOF
                        increment customer-master-count
                end-if

        3:      Read product master record
                If not EOF
                        increment product-master-count
                end-if
        ENDCASE

END
```

Temporal cohesion

- This is a time relation. The elements are **related by the time** at which they are executed

- Usually used in initialization
  - Initialisation
    Open files
    Enter date
    Initialise counts / sums to zero
    Print report headings, etc
    END

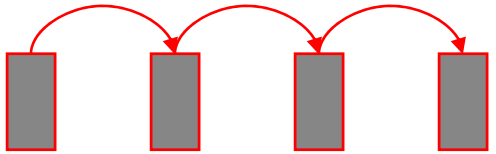


- Must be executed with the same span of time
- The order is not important

Procedural cohesion

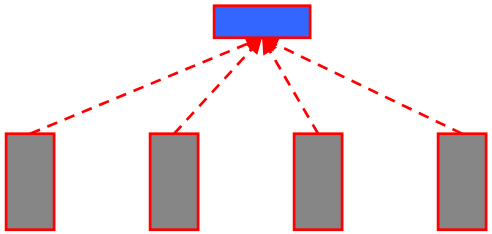
- When elements are related because they operate according to a procedure
  - the sequence is preset by a **standard** procedure

```
Read-student-records-and-total-student-ages
        No-of-records = 0
        Total-age = 0
        Read student-record
        While not EOF
                Tot-age = Tot-age + student-age
                No-of-records = No-of-records + 1
        End do
END
```



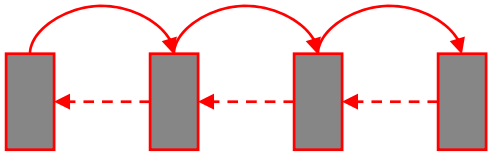
Communicational cohesion

- The elements are grouped together because they all operate on the **same central piece of data**
  - e.g. a module may contain the validation procedures of ***all*** fields in a record.



Sequential cohesion

- Module contains elements which are dependent on the processing of previous elements
- The output from one element is the input to the next



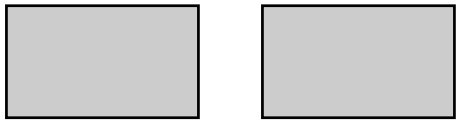
Functional Cohesion

- All elements of a module contribute to the performance of a **single functional**
  - A functional cohension component performs only the function and nothing else

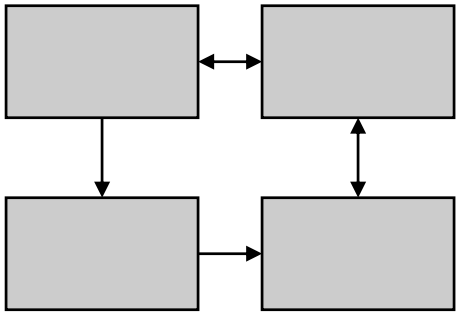
Coupling

- Coupling is a measure of the extent of information **interchange between modules**

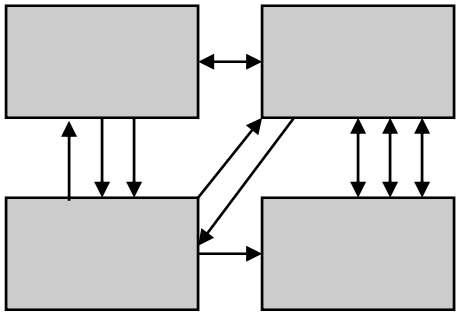
- Tight coupling
  - Large dependence on the structure of one module by another.
  - This is bad!
- Loose coupling
  - Large dependence on the structure of one module by another.
  - Modules with loose coupling are more independent and easier to maintain.
  - This is good!



Uncoupled -
no dependencies



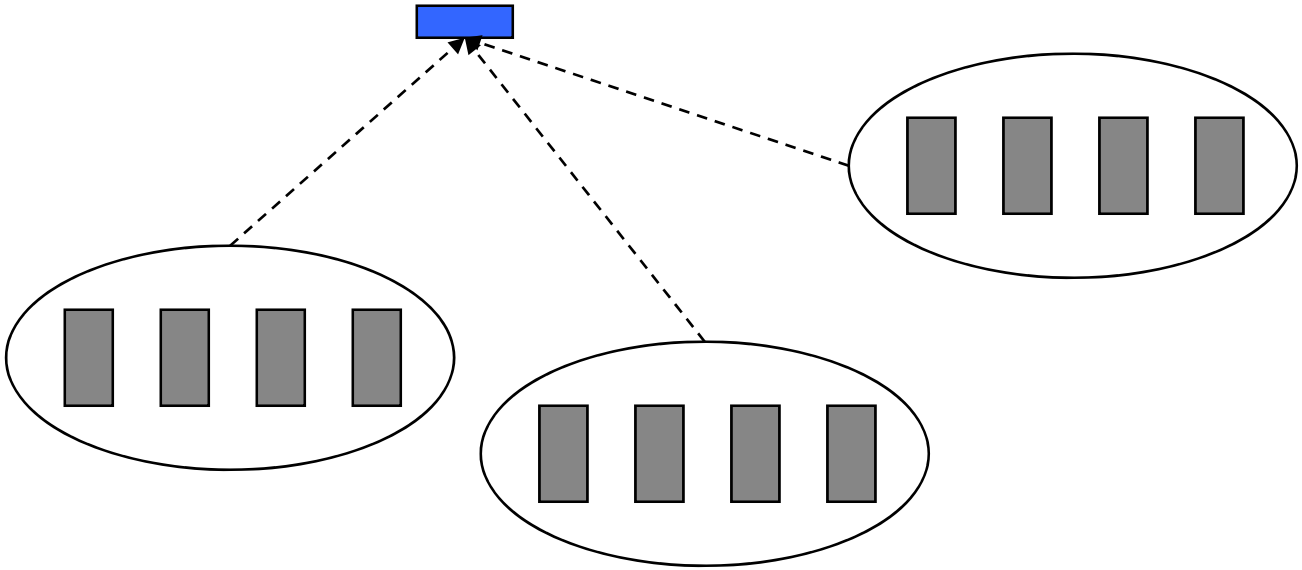
Loosely coupled -
some dependencies



Highly coupled -
many dependencies

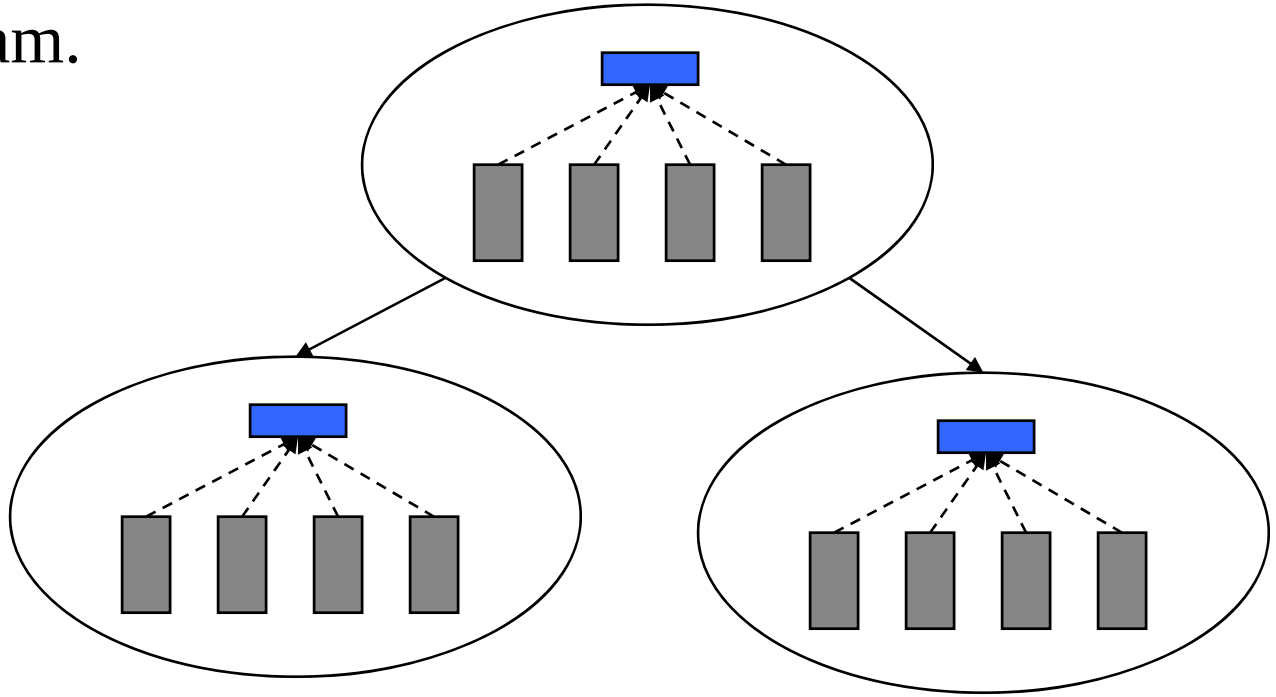
Module Communication - global data

- Data defined in the calling environment and used, from the calling environment, by the called module.
- accessible from all modules which are part of the module in which it is declared.
- The scope of the variable is the module and all contained in it.



Module Communication - local data

- Local data is variables which are declared inside a module and are used inside this module only.
- The data in these variables can only be accessed inside the program.

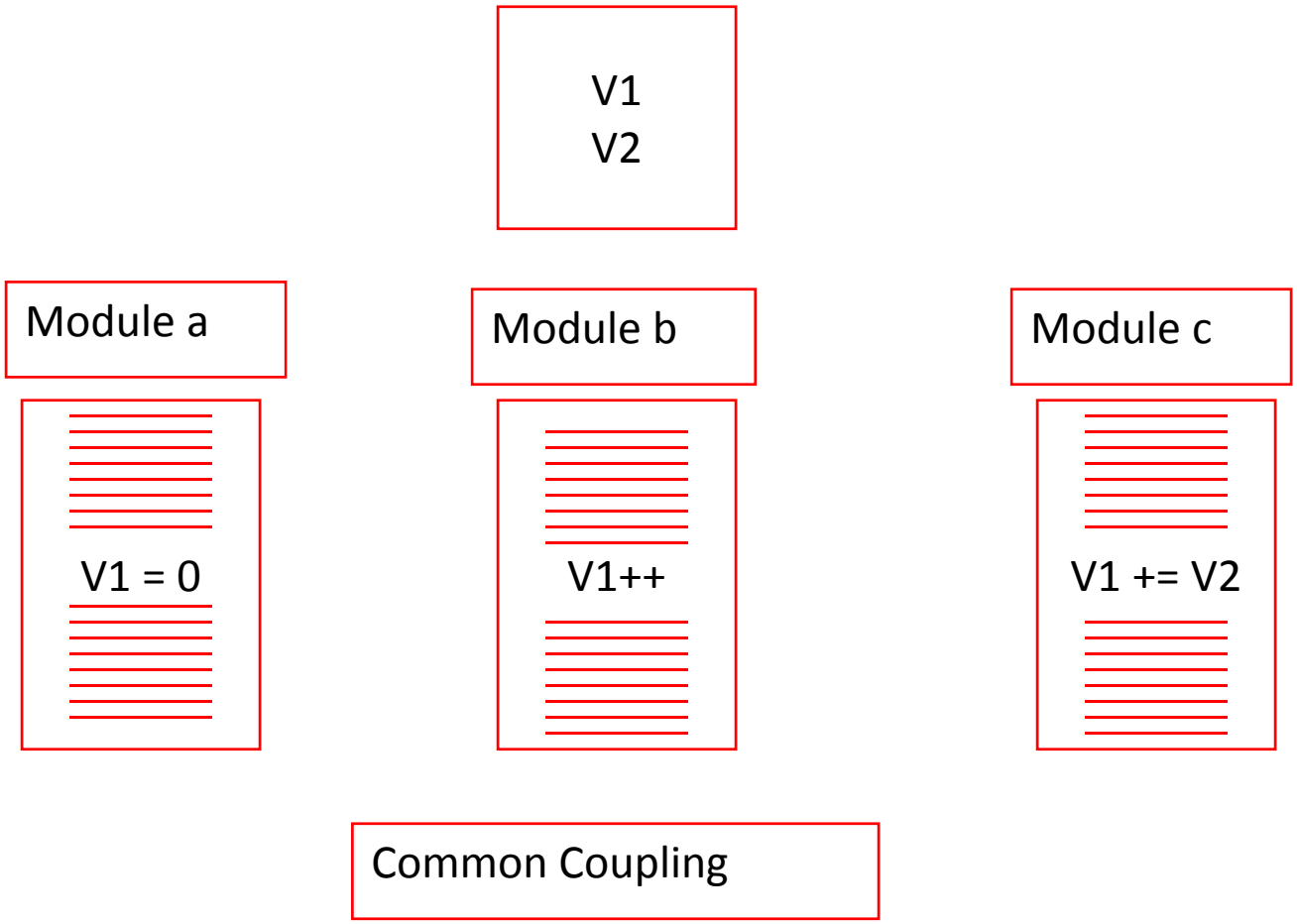


Module Communication

- Coupling depends on
  - The reference made from one module to another
    - M1 invokes M2 => M1 depends on M2 for completion of its function
  - The amount of data passed from one module to another
    - M1 passes a parameter to M2
  - The amount of one module has over the other
    - M1 passes a control flag to M2.
    - The flag tells M2 the state of some resource, which process to invoke, or whether to invoke a process at all

Types of Coupling

- Content coupling
  - One module directly affects the working of another module
    - A module changes another module's data
    - Control is passed from one module to the middle of another (as in a jump)
- Common coupling
  - This occurs when all modules reference the same global data structure
  - All data is shared by modules
- External coupling
  - Modules communicate through an external medium, such as files


V1
V2
Module a
V1 = 0
Module b
V1++
Module c
V1 += V2
Common Coupling

More Types of Coupling

- Control coupling
  - This occurs when a module **passes a control variable** to another module to control the other module's logic
    - A program flag
- Stamp coupling
  - One module passes a non-global **data structure** (rather than a simple data) to another module in the form of a parameter
- Data coupling
  - A module passes a non-global data variable to another module
  - This is more independent than stamp coupling, because it is also less dependent on the data structure

Coupling and inheritance

- Object-oriented systems are loosely coupled because there is **no shared state** and objects communicate using message passing

- However, an object class is **coupled to its super-classes**
  - Changes made to the attributes or operations in a super-class propagate to all sub-classes
  - Such changes must be carefully controlled

Design Quality – Understandability, Adaptability, Tractability

Understandability

- Related to several component characteristics
  - *Cohesion*. Can the component be understood on its own?
  - *Naming*. Are meaningful names used?
  - *Documentation*. Is the design well-documented?
  - *Complexity*. Are complex algorithms used?
- Informally, high complexity means many relationships between different parts of the design. hence it is hard to understand
- Most design quality metrics are oriented towards complexity measurement

Adaptability

- A design is adaptable if:
  - Its components are loosely coupled
  - It is well-documented and the documentation is up to date
  - There is an obvious correspondence between design levels (design visibility)
  - Each component is a self-contained entity (tightly cohesive)
- To adapt a design, it must be possible to trace the links between design components so that change consequences can be analysed

Design tractability