Cohesion and Coupling

Introduction

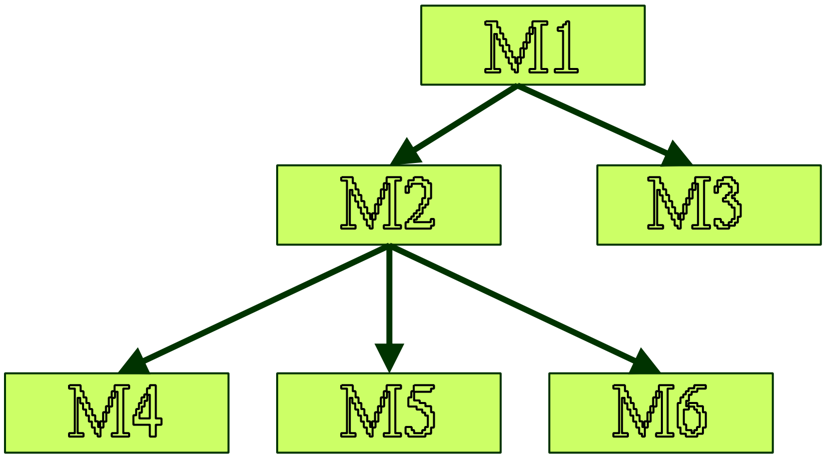
Design phase transforms SRS document:

- To a form easily implementable in some programming language.

Items Designed During Design Phase

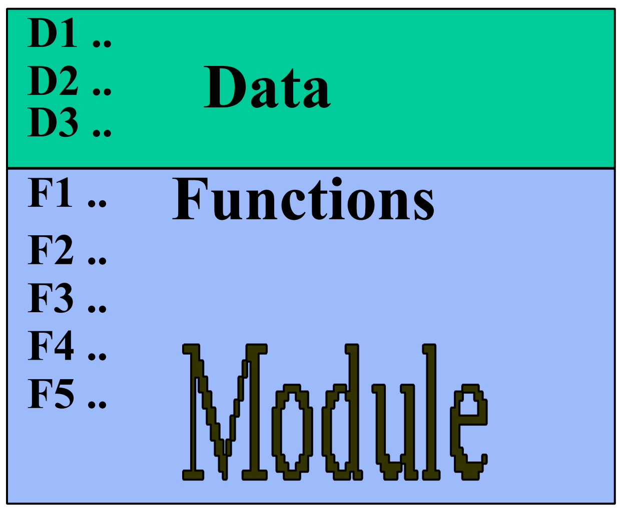
- ✧ Module structure,
- ✧ Control relationship among the modules
  - call relationship or invocation relationship
- ✧ Interface among different modules,
  - Data items exchanged among different modules,
- ✧ Data structures of individual modules,
- ✧ Algorithms for individual modules.

Module Structure



A module consists of:

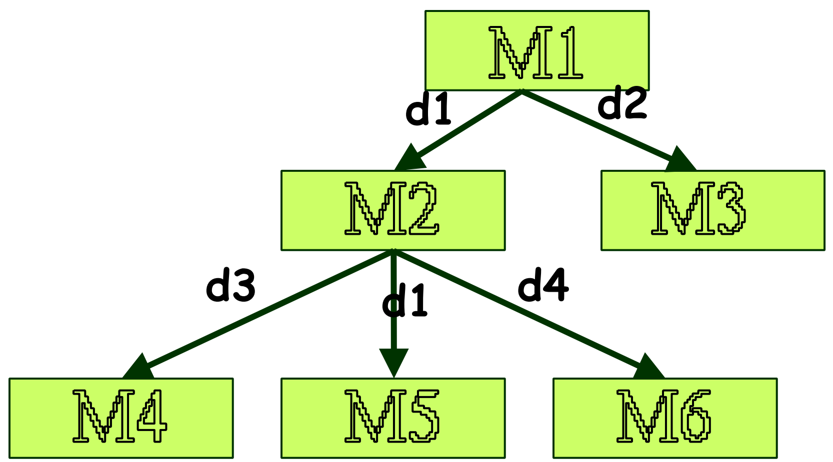
- Several functions
- Associated data structures.



High-Level Design

Identity:

- Modules
- Control relationships among modules
- Interfaces among modules.



Detailed Design

For each module, design:

- Data structure
- Algorithms

Outcome of detailed design:

- Module specification.

How are Abstraction and Decomposition Principles Used in Design?

Two principal ways:

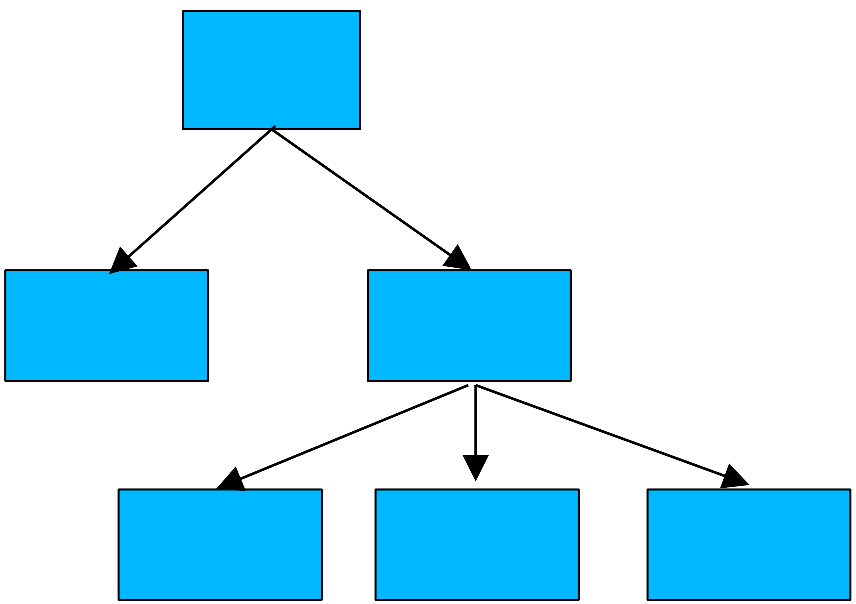
- Modular Design
- Layered Design

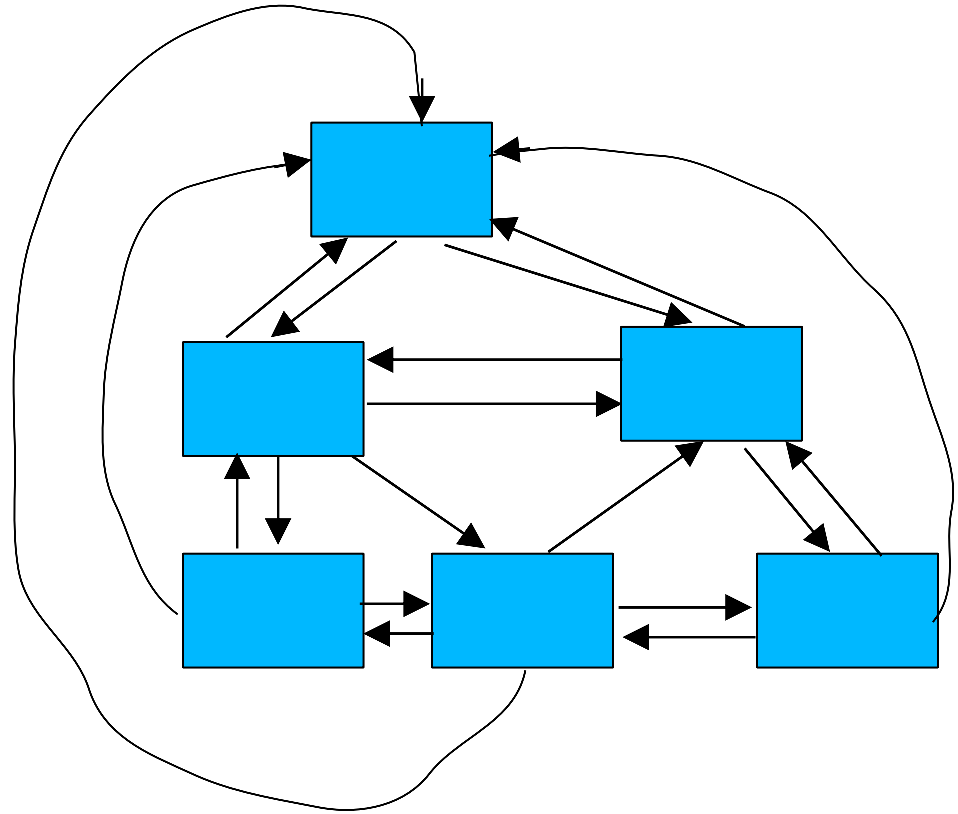
Modularity

Modularity is a fundamental attributes of any good design.

- Decomposition of a problem cleanly into modules:
- Modules are almost independent of each other
- Divide and conquer principle.

Layered Design





Layered Design

Neat arrangement of modules in a hierarchy means:

- Low fan-out
- Control abstraction

Modularity

In technical terms, modules should display:

- High cohesion
- Low coupling.

Cohesion and Coupling

Cohesion is a measure of:

- functional strength of a module.
- A cohesive module performs a single task or function.

Coupling between two modules:

- A measure of the degree of the interdependence or interaction between the two modules.

Cohesion and Coupling

A module having high cohesion and low coupling:

- functionally independent of other modules:
  - A functionally independent module has minimal interaction with other modules.

Advantages of Functional Independence

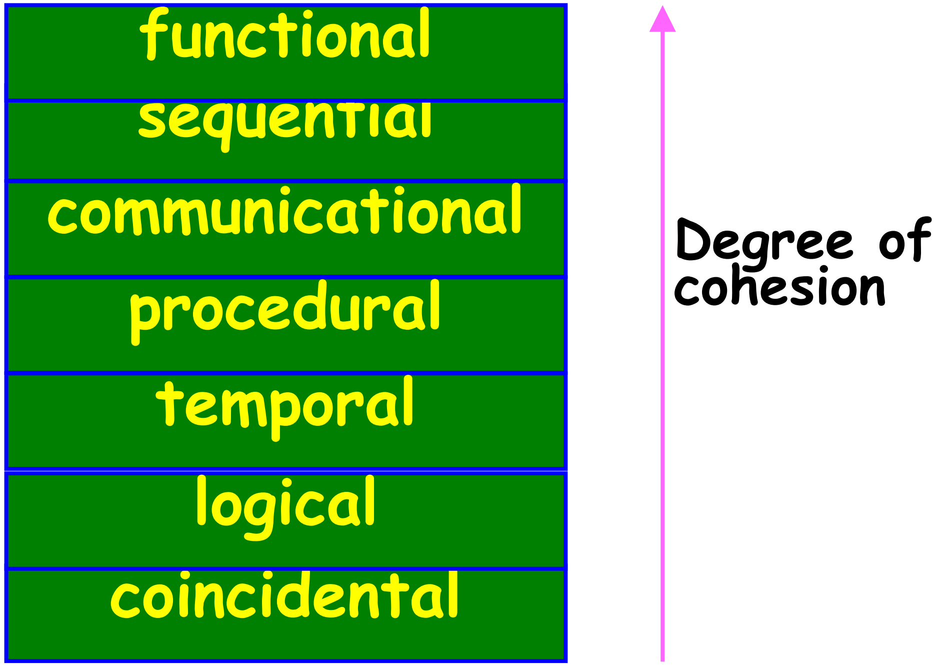
Better understandability and good design:

Complexity of design is reduced,

Different modules easily understood in isolation:

- Modules are independent

Classification of Cohesiveness



Coincidental Cohesion

The module performs a set of tasks:

- Which relate to each other very loosely, if at all.
  - The module contains a random collection of functions.
  - Functions have been put in the module out of pure coincidence without any thought or design.

Logical Cohesion

All elements of the module perform similar operations:

- e.g. error handling, data input, data output, etc.

An example of logical cohesion:

- A set of print functions to generate an output report arranged into a single module.

Temporal Cohesion

The module contains tasks that are related by the fact:

- All the tasks must be executed in the same time span.

Example:

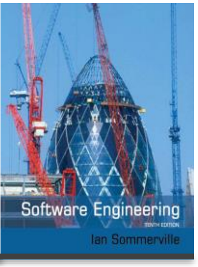
- The set of functions responsible for
  - initialization,
  - start-up, shut-down of some process, etc.

Procedural Cohesion

The set of functions of the module:

- All part of a procedure (algorithm)
- Certain sequence of steps have to be carried out in a certain order for achieving an objective,
  - e.g. the algorithm for decoding a message.

Communicational Cohesion



All functions of the module:

- Reference or update the same data structure,

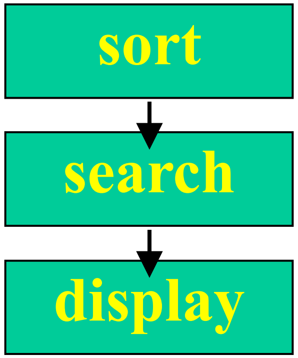
Example:

- The set of functions defined on an array or a stack.

Sequential Cohesion

Elements of a module form different parts of a sequence,

- Output from one element of the sequence is input to the next.
- Example:



Functional Cohesion

Different elements of a module cooperate:

- To achieve a single function,
- e.g. managing an employee's pay-roll.

When a module displays functional cohesion,

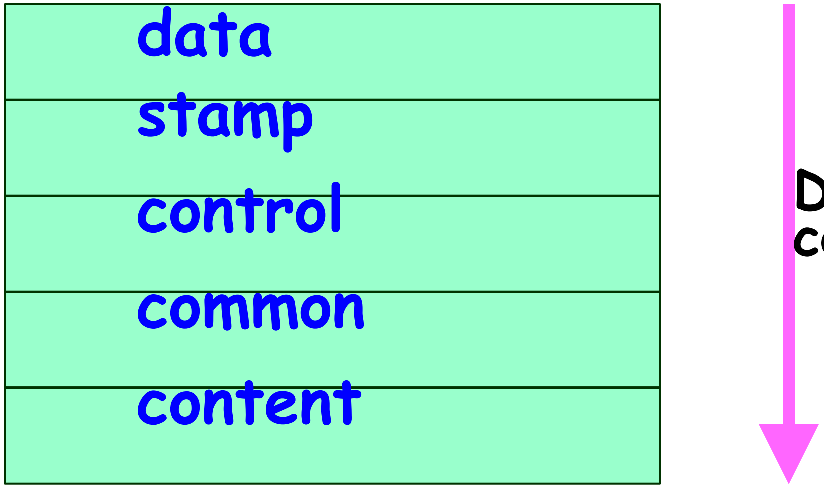
- We can describe the function using a single sentence.

Coupling

Coupling indicates:

- How closely two modules interact or how interdependent they are.
- The degree of coupling between two modules depends on their interface complexity.

Classes of coupling



Data coupling

Two modules are data coupled,

- If they communicate via a parameter:
  - an elementary data item,
  - e.g an integer, a float, a character, etc.
- The data item should be problem related:
  - Not used for control purpose.

Stamp Coupling

Two modules are stamp coupled,

- If they communicate via a composite data item
  - such as a record in PASCAL
  - or a structure in C.

Control Coupling

Data from one module is used to direct:

- Order of instruction execution in another.

Example of control coupling:

- A flag set in one module and tested in another module.

Common Coupling

Two modules are <u>common coupled</u>,

- If they share some global data.

Content Coupling

Content coupling exists between two modules:

- If they share code,
- e.g, branching from one module into another module.

The degree of coupling increases

- from data coupling to content coupling.

Function-Oriented Design

A system is looked upon as something

- That performs a set of functions.

Starting at this high-level view of the system:

- Each function is successively refined into more detailed functions.
- Functions are mapped to a module structure.

Example

The function create-new-library- member:

- Creates the record for a new member,
- Assigns a unique membership number
- Prints a bill towards the membership

Object-Oriented Design

System is viewed as a collection of objects (i.e. entities).

System state is decentralized among the objects:

- Each object manages its own state information.

Object-Oriented Design Example

Library Automation Software:

- Each library member is a separate object
  - With its own data and functions.
- Functions defined for one object:
  - Cannot directly refer to or change data of other objects.

Object-Oriented Design

Objects have their own internal data:

- Defines their state.

Similar objects constitute a class.

- Each object is a member of some class.

Classes may inherit features

- From a super class.

Conceptually, objects communicate by message passing.

Object-Oriented versus Function-Oriented Design

Unlike function-oriented design,

- In OOD the basic abstraction is not functions such as “sort”, “display”, “track”, etc.,
- But real-world entities such as “employee”, “picture”, “machine”, “radar system”, etc.

Object-Oriented versus Function-Oriented Design

In OOD:

- Software is not developed by designing functions such as:
  - update-employee-record,
  - get-employee-address, etc.
- But by designing objects such as:
  - employees,
  - departments, etc.

Object-Oriented versus Function-Oriented Design

Grady Booch sums up this fundamental difference saying:

- “Identify verbs if you are after procedural design and nouns if you are after object-oriented design.”

Object-Oriented versus Function-Oriented Design

Function-oriented techniques group functions together if:

- As a group, they constitute a higher level function.

On the other hand, object-oriented techniques group functions together:

- On the basis of the data they operate on.

Fire-Alarm System

We need to develop a computerized fire alarm system for a large multi-storied building:

- There are 80 floors and 1000 rooms in the building.

Different rooms of the building:

- Fitted with smoke detectors and fire alarms.

The fire alarm system would monitor:

- Status of the smoke detectors.

Whenever a fire condition is reported by any smoke detector:

- the fire alarm system should:
  - Determine the location from which the fire condition was reported
  - Sound the alarms in the neighboring locations.

The fire alarm system should:

- Flash an alarm message on the computer console:
  - Fire fighting personnel man the console round the clock.

Function-Oriented Approach:

✧ /* Global data (system state) accessible by various functions */

```
BOOL  detector_status[1000];
int   detector_locs[1000];
BOOL  alarm-status[1000]; /* alarm activated when status set */
int   alarm_locs[1000]; /* room number where alarm is located */
int   neighbor-alarms[1000][10];/*each detector has at most*/
                        /* 10 neighboring alarm locations */
```

The functions which operate on the system state:

```
interrogate_detectors();
get_detector_location();
determine_neighbor();
ring_alarm();
reset_alarm();
report_fire_location();
```

Object-Oriented Approach:

- class detector
- attributes: status, location, neighbors
- operations: create, sense-status, get-location,
- find-neighbors
- class alarm
- attributes: location, status
- operations: create, ring-alarm, get_location,
- reset-alarm
- In the object oriented program,
  - appropriate number of instances of the class detector and alarm should be created.

Concluding Remark

Though outwardly a system may appear to have been developed in an object oriented fashion,

- But inside each class there is a small hierarchy of functions designed in a top-down manner.