Software Design

Anil Kumar Dudyala
Asst.Prof, Dept of CSE
NIT Patna
Source
(Rajib Mall)

Contents being covered

Introduction to software design

Goodness of a design

Functional Independence

Cohesion and Coupling

Function-oriented design vs. Object-oriented design

Summary

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
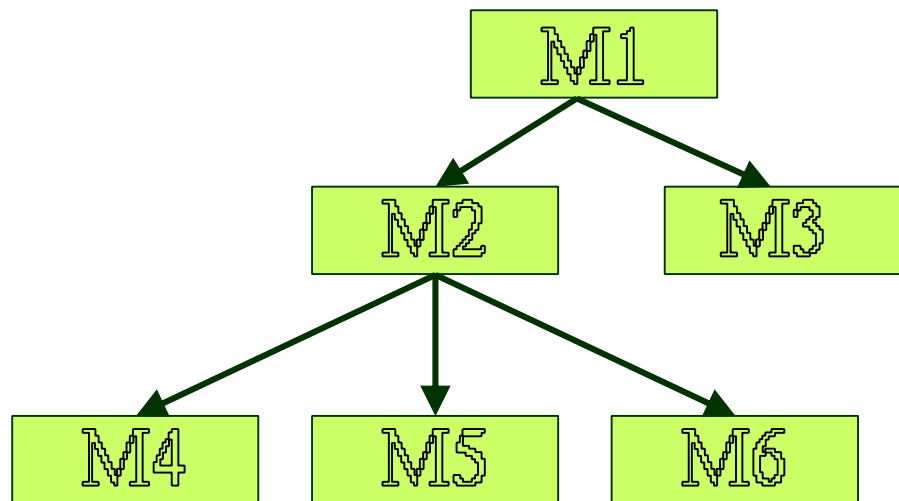- call relationship or invocation relationship

Interface among different modules,
- Data items exchanged among different modules,

Data structures of individual modules,
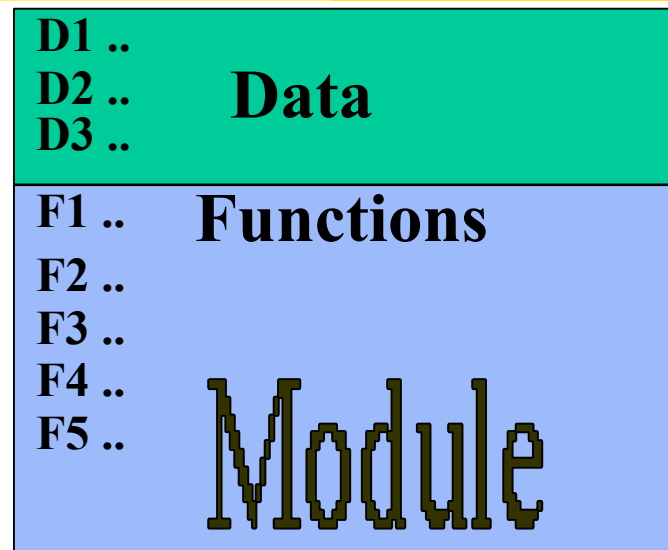
Algorithms for individual modules.

Module Structure

Introduction

A module consists of:
- Several functions
- Associated data structures.

D1 .. D2 .. D3 ..	**Data**
F1 ..	**Functions**
F2 ..	
F3 ..	
F4 ..	
F5 ..	Module

Introduction

Good software designs:

- Seldom arrived through a single step procedure:

- But through a series of steps and iterations.

Introduction

Design activities are usually classified into two stages:

- ◦ <mark>Preliminary (or high-level) design</mark>.
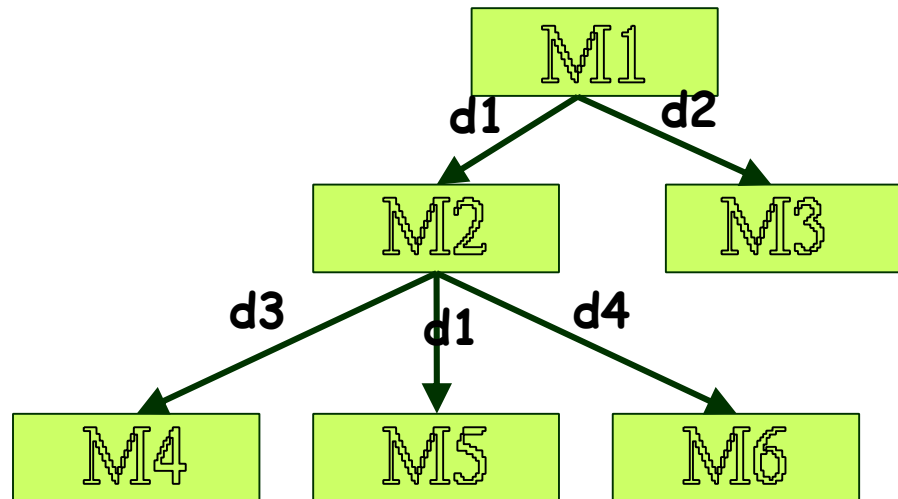- ◦ <mark>Detailed design.</mark>

Meaning and scope of the two stages:

- ◦ Vary considerably from one methodology to another.

High-Level Design

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

High-Level Design

The outcome of high-level design:

- Program structure (or software architecture).

High-Level Design

Several notations are available to represent high-level design:

- Usually a tree-like diagram called [structure chart](#) is used.

- Other notations:

  - Jackson diagram or Warnier-Orr diagram can also be used.

Detailed Design

For each module, design:
- Data structure
- Algorithms

Outcome of detailed design:
- Module specification.

A Classification of Design Methodologies

Procedural (aka Function-oriented)

Object-oriented

More recent:
◦ Aspect-oriented
◦ Component-based (Client-Server)

Does a Design Technique Lead to a Unique Solution?

No:

- Several subjective decisions need to be made to trade off among different parameters.
- Even the same designer can come up with several alternate design solutions.

Analysis versus Design

An analysis technique helps elaborate the customer requirements through careful thinking:

◦ And at the same time consciously avoids making any decisions regarding implementation.

The design model is obtained from the analysis model through transformations over a series of steps:

◦ Decisions regarding implementation are consciously made.

A Fundamental Question

How to distinguish between the superior of two alternate design solutions?

◦ Unless we know what a good software design is:

◦ We can not possibly design one.

Good and Bad Designs

There is no unique way to design a system.

Even using the same design methodology:

- Different designers can arrive at very different design solutions.

We need to distinguish between good and bad designs.

Which of Two is a Better Design?

Should implement all functionalities of the system correctly.

Should be easily understandable.

Should be efficient.

Should be easily amenable to change,
- i.e. easily maintainable.

Which of Two is a Better Design?

Understandability of a design is a major issue:

- Determines goodness of design:
- A design that is easy to understand:
  - Also easy to maintain and change.

Which of Two is a Better Design?

Unless a design is easy to understand,
- ◦ Tremendous effort needed to maintain it
- ◦ We already know that about 60% effort is spent in maintenance.

If the software is not easy to understand:
- ◦ Maintenance effort would increase many times.

Understandability

Use consistent and meaningful names:
- For various design components.

Should make use of abstraction and decomposition principles in ample measure.

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.

Modularity

If modules are **independent**:
- Modules can be **understood separately**,
  - **Reduces the complexity** greatly.
- To understand why this is so,
  - Remember that it is very difficult to break a bunch of sticks but very easy to break the sticks individually.

Layered Design
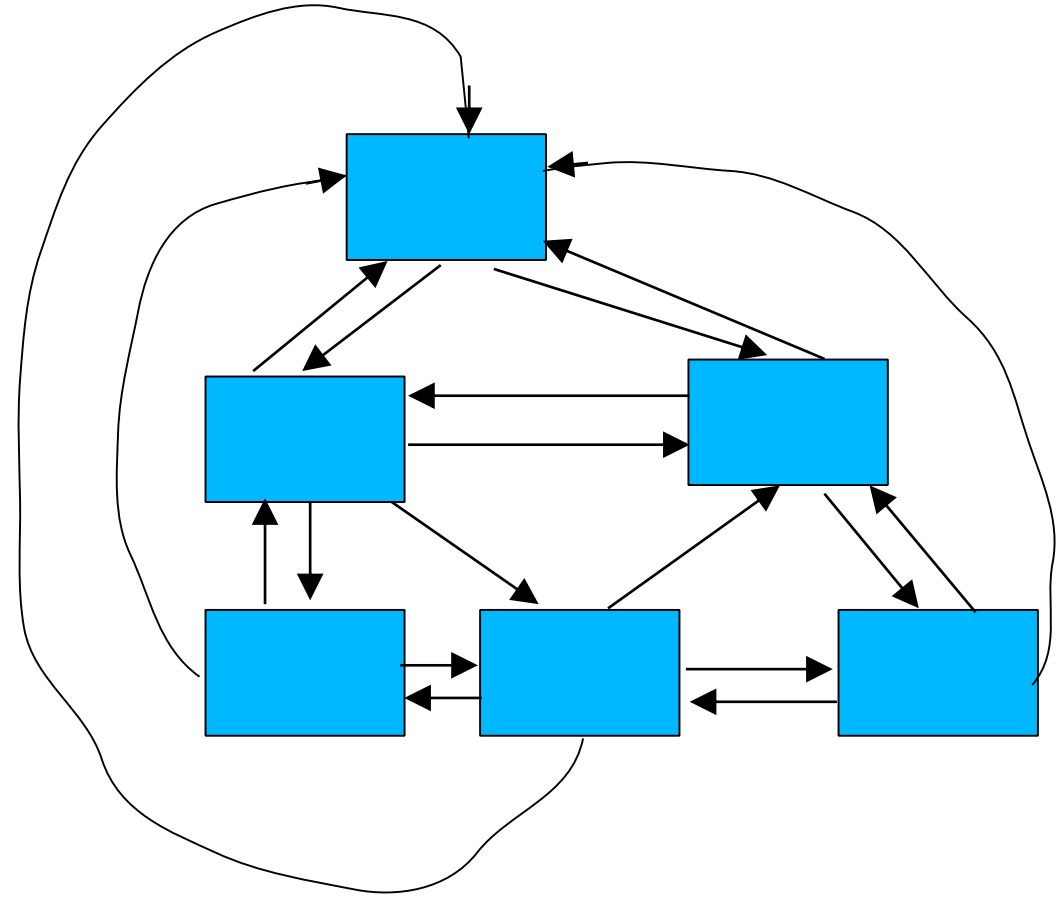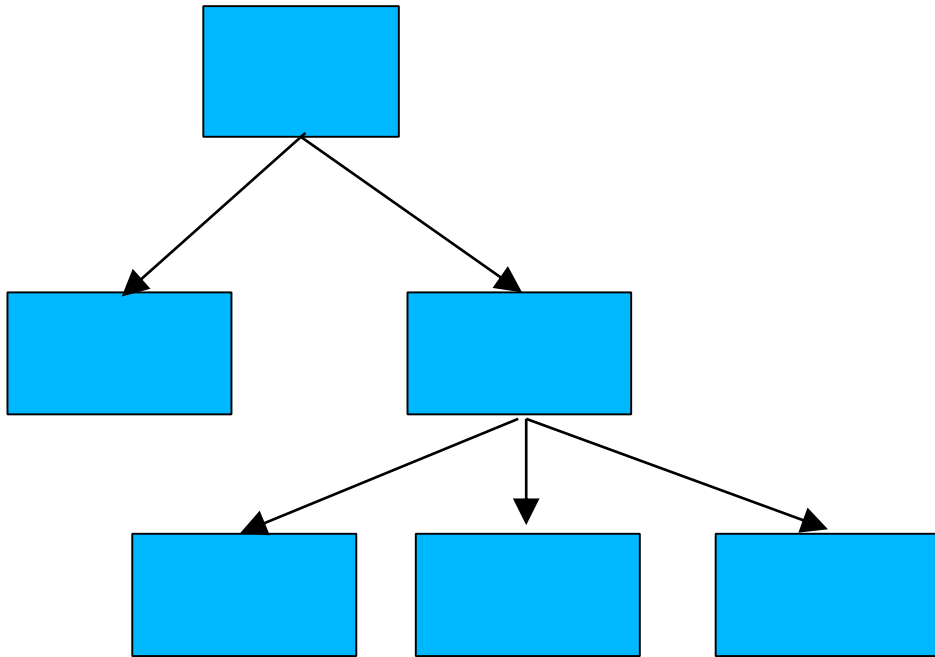
Layered Design

Neat arrangement of modules in a hierarchy means:

- Low fan-out
- Control abstraction

Modularity

In technical terms, modules should display:
- High cohesion
- Low coupling.

What is cohesion and 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

Advantages of Functional Independence

Functional independence reduces error propagation.

- Degree of interaction between modules is low.
- An error existing in one module does not directly affect other modules.

Reuse of modules is possible.

Advantages of Functional Independence

A functionally independent module:

- Can be easily taken out and reused in a different program.
  - Each module does some well-defined and precise function
  - The interfaces of a module with other modules is simple and minimal.

Functional Independence

Unfortunately, there are no ways:

- To quantitatively measure the degree of cohesion and coupling.

- Classification of different kinds of cohesion and coupling:

  - Can give us some idea regarding the degree of cohesiveness of a module.

Classification of Cohesiveness

Classification is often subjective:
- Yet gives us some idea about cohesiveness of a module.

By examining the type of cohesion exhibited by a module:
- We can roughly tell whether it displays high cohesion or low cohesion.

Classification of Cohesiveness

IMPORTANT

functional
sequential
communicational
procedural
temporal
logical
coincidental

Degree of cohesion
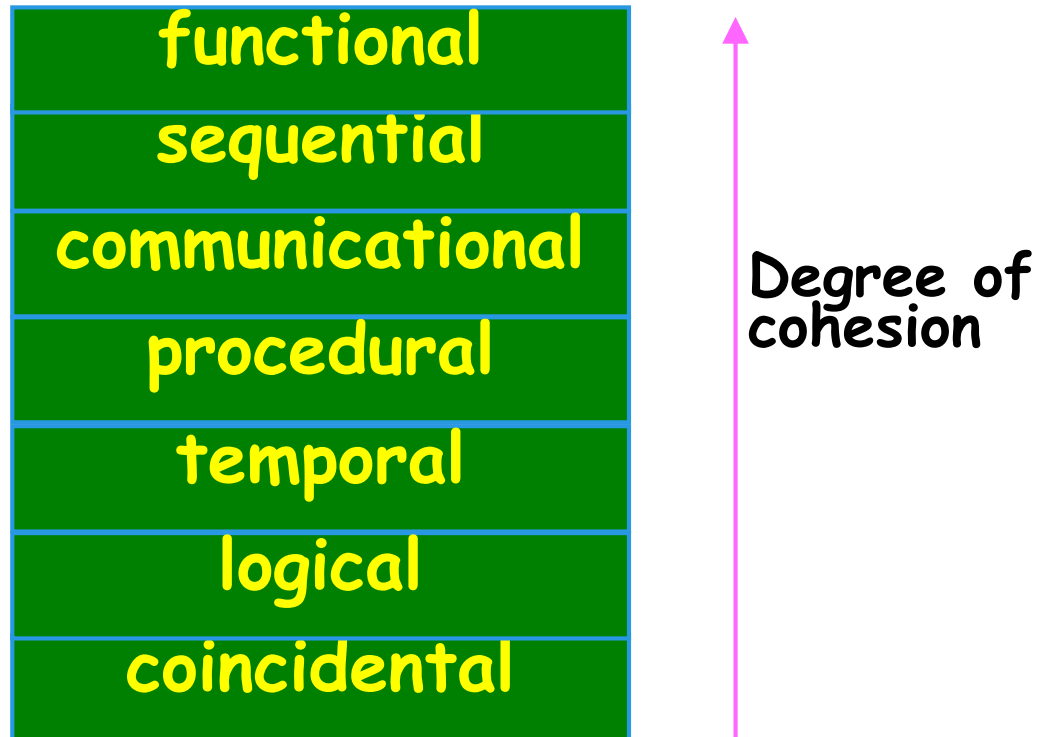
Coincidental Cohesion

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:
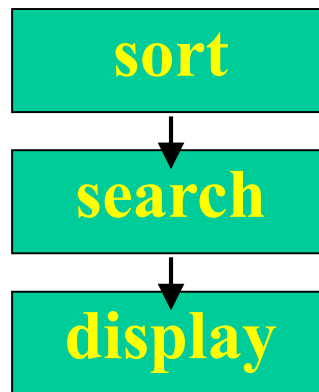- Reference or update the same data structure,

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:

```
┌──────────────┐
│     sort     │
└──────────────┘
        │
        ▼
┌──────────────┐
│    search    │
└──────────────┘
        │
        ▼
┌──────────────┐
│   display    │
└──────────────┘
```

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.

Determining Cohesiveness

Write down a sentence to describe the function of the module

- If the sentence is compound,
  - It has a sequential or communicational cohesion.
- If it has words like "first", "next", "after", "then", etc.
  - It has sequential or temporal cohesion.
- If it has words like initialize,
  - It probably has temporal cohesion.

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.

Coupling

There are no ways to precisely determine coupling between two modules:

- ◦ Classification of different types of coupling will help us to approximately estimate the degree of coupling between two modules.

Five types of coupling can exist between any two modules.

Classes of coupling

data
stamp
control
common
content

Degree 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 common coupled,
- 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.

Neat Hierarchy

Control hierarchy represents:

◦ Organization of modules. software architecture

◦ Control hierarchy is also called program structure.

Most common notation:

◦ A tree-like diagram called structure chart.

Layered Design

Essentially means:
- Low fan-out
- Control abstraction

Characteristics of Module Hierarchy

Depth:
- Number of levels of control

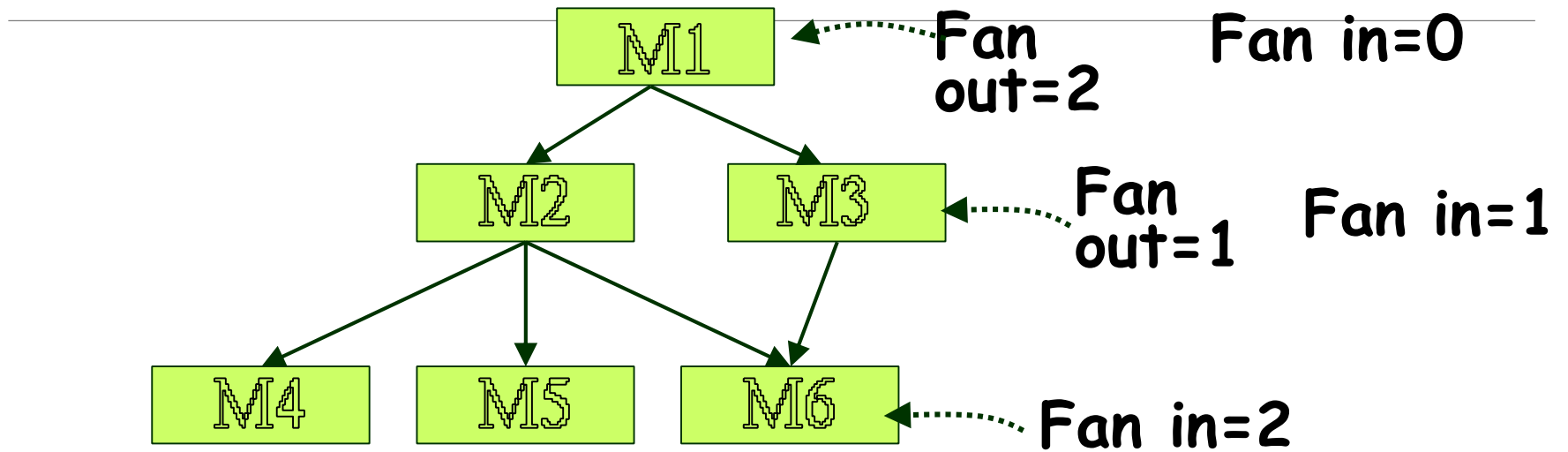**Width:**
- Overall span of control.

Fan-out:
- A measure of the number of modules directly controlled by given module.

Characteristics of Module Structure

Fan-in:

- Indicates how many modules directly invoke a given module.

- High fan-in represents code reuse and is in general encouraged.

Module Structure

Layered Design

A design having modules:

- With high fan-out numbers is not a good design:
- A module having high fan-out lacks cohesion.

Goodness of Design

A module that invokes a large number of other modules:

◦ Likely to implement several different functions:

◦ Not likely to perform a single cohesive function.

Control Relationships

A module that controls another module:
- Said to be **superordinate** to it.

Conversely, a module controlled by another module:
- Said to be **subordinate** to it.
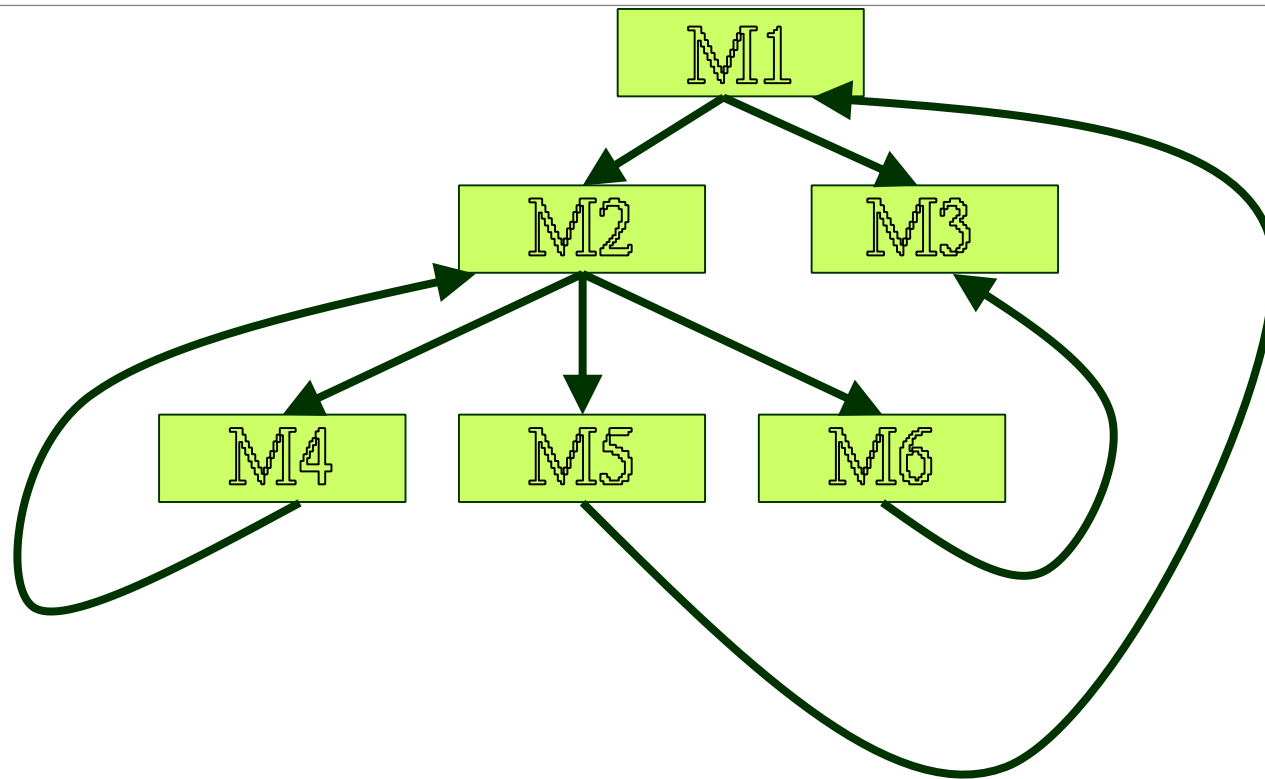
Visibility and Layering

A module A is said to be visible by another module B,
- If A directly or indirectly calls B.

The layering principle requires
- Modules at a layer can call only the modules immediately below it.

Bad Design

Abstraction

A module is unaware (how to invoke etc.) of the higher level modules.

Lower-level modules:
- Do input/output and other low-level functions.

Upper-level modules:
- Do more managerial functions.

# Abstraction

The principle of abstraction requires:

- Lower-level modules do not invoke functions of higher level modules.
- Also known as layered design.

High-level Design

High-level design maps functions into modules {fi} {mj} such that:

- ◦ Each module has high cohesion
- ◦ Coupling among modules is as low as possible
- ◦ Modules are organized in a neat hierarchy

High-level Design
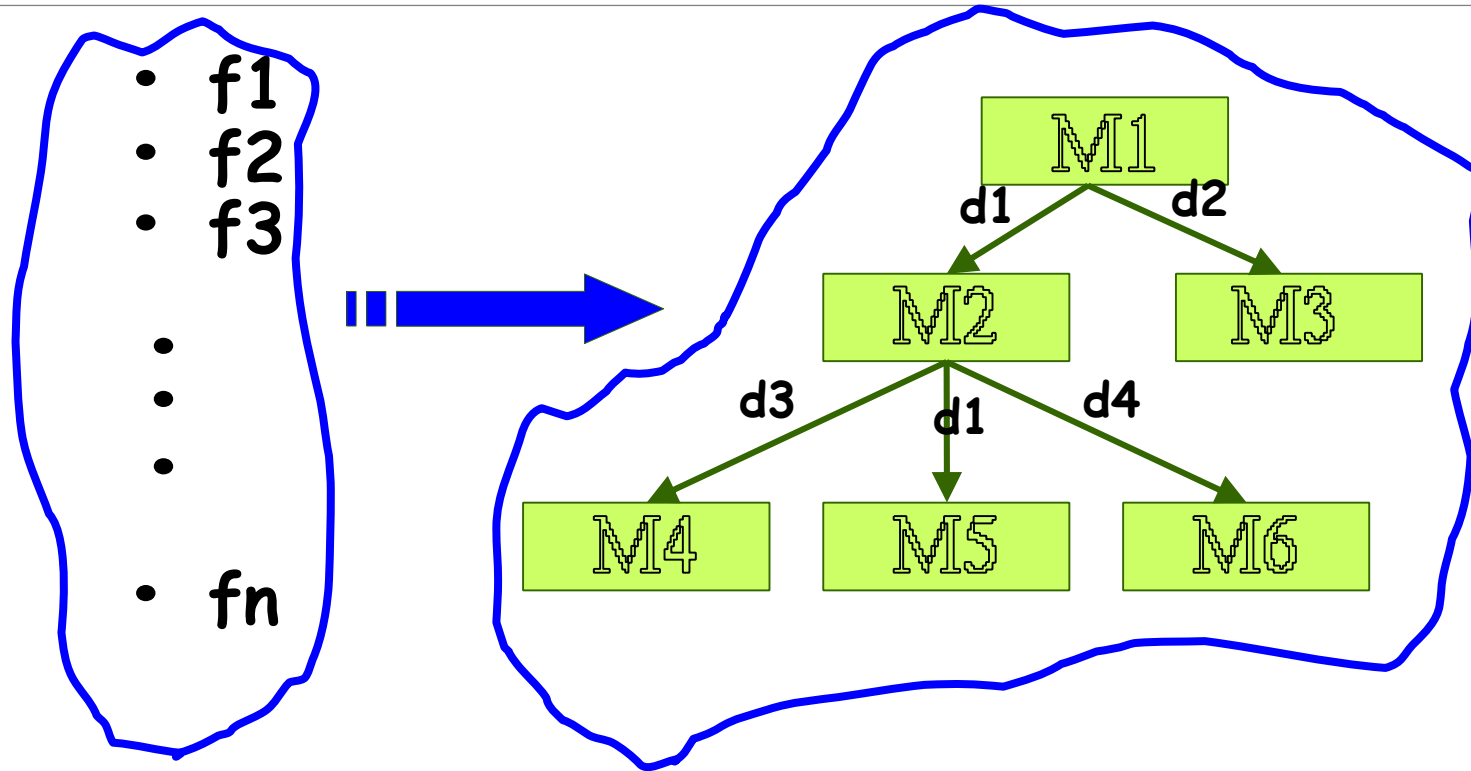
Design Approaches

Two fundamentally different software design approaches:

◦ Function-oriented design

◦ Object-oriented design

Design Approaches

These two design approaches are radically different.

- However, are complementary
  - Rather than competing techniques.
- Each technique is applicable at
  - Different stages of the design process.

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

Example

Create-library-member function consists of the following sub-functions:

- ◦ Assign-membership-number
- ◦ Create-member-record
- ◦ Print-bill

Function-Oriented Design

Each subfunction:

◦ Split into more detailed subfunctions and so on.

Function-Oriented Design

The system state is centralized:
- Accessible to different functions,
- Member-records:
    - Available for reference and updation to several functions:
        - Create-new-member
        - Delete-member
        - Update-member-record

Function-Oriented Design

Several function-oriented design approaches have been developed:

- ◦ Structured design (Constantine and Yourdon, 1979)
- ◦ Jackson's structured design (Jackson, 1975)
- ◦ Warnier-Orr methodology
- ◦ Wirth's step-wise refinement
- ◦ Hatley and Pirbhai's Methodology

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 <mark>may inherit features</mark>
◦ 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

In OOD:

- State information is not shared in a centralized data.

- But is distributed among the objects of the system.

Example:

In an employee pay-roll system, the following can be global data:

- ◦ employee names,
- ◦ code numbers,
- ◦ basic salaries, etc.

Whereas, in object oriented design:

- ◦ Data is distributed among different employee objects of the system.

Object-Oriented versus Function-Oriented Design

Objects communicate by message passing.

- One object may discover the state information of another object by interrogating it.

Object-Oriented versus Function-Oriented Design

Of course, somewhere or other the functions must be implemented:

◦ The functions are usually associated with specific real-world entities (objects)

◦ Directly access only part of the system state information.

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.

Object-Oriented versus Function-Oriented Design

To illustrate the differences between object-oriented and function-oriented design approaches,

- let us consider an example ---
- An automated fire-alarm system for a large building.

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.

Fire-Alarm System

Different rooms of the building:
- Fitted with smoke detectors and fire alarms.

The fire alarm system would monitor:
- Status of the smoke detectors.

Fire-Alarm System

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.

Fire-Alarm System

The fire alarm system should:

◦ Flash an alarm message on the computer console:

  ◦ Fire fighting personnel man the console round the clock.

Fire-Alarm System

After a fire condition has been successfully handled,

◦ The fire alarm system should let fire fighting personnel reset the alarms.

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.

Object-Oriented versus Function-Oriented Design

In the function-oriented program :

◦ The system state is centralized

◦ Several functions accessing these data are defined.

In the object oriented program,

◦ The state information is distributed among various sensor and alarm objects.

Object-Oriented versus Function-Oriented Design

Use OOD to design the classes:

- Then applies top-down function oriented techniques
  - To design the internal methods of classes.

Object-Oriented versus Function-Oriented Design

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.

Summary

We started with an overview of:

◦ Activities undertaken during the   software design phase.

We identified:

◦ The information need to be produced at the end of the design phase:

  ◦ So that the design can be easily implemented using a programming language.

Summary

We characterized the features of a good software design by introducing the concepts of:

- ◦ fan-in, fan-out,
- ◦ cohesion, coupling,
- ◦ abstraction, etc.

Summary

We classified different types of cohesion and coupling:

- ◦ Enables us to approximately determine the cohesion and coupling existing in a design.

Summary

Two fundamentally different approaches to software design:

- Function-oriented approach
- Object-oriented approach

Summary

We looked at the essential philosophy behind these two approaches

◦ These two approaches are not competing but complementary approaches.