Software Engineering and Information System

Lecture 09:
MODULAR SOFTWARE DEVELOPMENT

Md. Al-Hasan

Department of Computer Science & Engineering (CSE)

Bangladesh Army University of Science & Technology (BAUST)

Software Engineering- A PRACTITIONER ' S APPROACH *By* Roger S. Pressman

Modularity

- A concept closely tied to abstraction
- Modularity supports independence of models
- Modules support abstraction in software
- Supports hierarchical structuring of programs
- Modularity enhances design clarity, eases implementation
- Reduces cost of testing, debugging and maintenance
- Cannot simply chop a program into modules to get modularly
- Need some criteria for decomposition

https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/

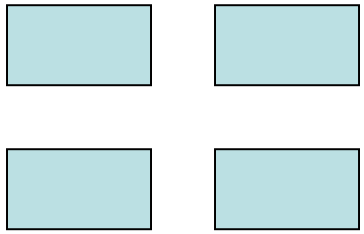
Desired Class/Object Interaction

- Maximize internal interaction (cohesion)
  - easier to understand
  - easier to test
- Minimize external interaction (coupling)
  - can be used independently
  - easier to test
  - easier to replace
  - easier to understand

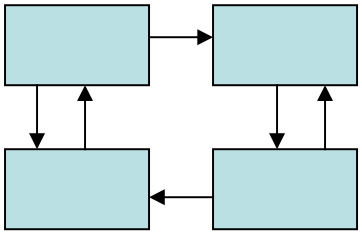
Characteristics of Good Design

- Component independence
  - High cohesion(structure, unity, consistency)
  - Low coupling
- Exception identification and handling
- Fault prevention and fault tolerance

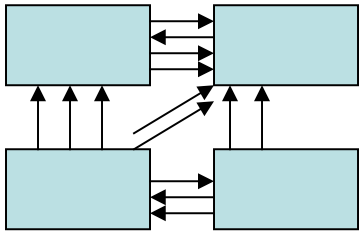
Coupling: Degree of dependence among components



No dependencies



Loosely coupled-some dependencies



Highly coupled-many dependencies

High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

Coupling

- **Coupling addresses the attribute of "degree of *interdependence*" between software units, modules or components.**

Content Coupling

Accessing the internal data or procedural information

Common Coupling

Control Coupling

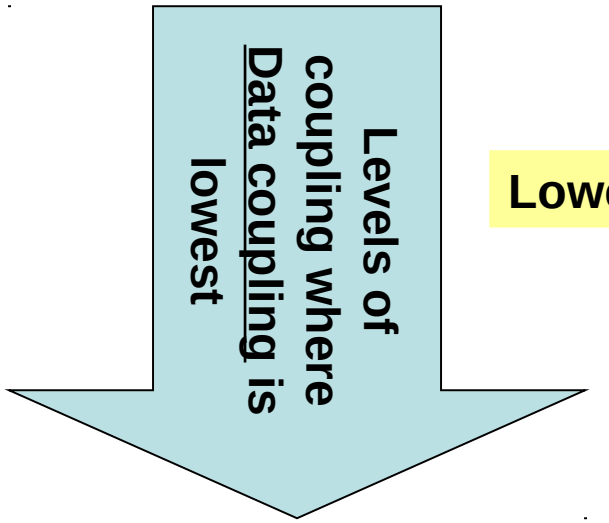
Stamp Coupling

Data Coupling

Passing only the necessary information

No Coupling

Ideal, but not practical



Lower the better

Content Coupling

- Definition: A module directly references the content of another module
  - module ***p*** modifies a statement of module ***q***
  - *Module **p** refers to local data of module **q** (in terms of a numerical displacement)*
  - *Module **p** branches to a local label of module **q***

Content Coupling (cont)

- Content coupled modules are inextricably interlinked
  - Change to module **p** requires a change to module **q** (including recompilation)
  - Reusing module **p** requires using module **q** also
- Typically only possible in assembly languages

Common Coupling

- Using global variables (i.e., *global coupling*)
- All modules have read/write access to a global data block
- Modules exchange data using the global data block (instead of arguments)
- *Single module with write access where all other modules have read access is not common coupling*

Common Coupling (cont)

- Have to look at many modules to determine the current state of a variable
- Side effects require looking at all the code in a function to see if there are any global effects
- Changes in one module to the declaration requires changes in all other modules
- Identical list of global variables must be declared for module to be reused
- Module is exposed to more data than is needed

Control Coupling

- Definition: Component passes control parameters to coupled components.
- May be either good or bad, depending on situation.
  - Bad when component must be aware of internal structure and logic of another module
  - Good if parameters allow factoring and reuse of functionality

Example

- Acceptable: Module p calls module q and q returns a flag that indicates an error (if any)
- Not Acceptable: Module p calls module q and q returns a flag back to p that says it must output the error “I goofed up”

Stamp Coupling

- Definition: Component passes a data structure to another component that does not have access to the entire structure.
- Requires second component to know how to manipulate the data structure (e.g., needs to know about implementation)
- May be necessary due to efficiency factors: this is a choice made by insightful designer, not lazy programmer.

Example

Customer billing system

The print routine of the customer billing accepts a customer data structure as an argument, parses it, and prints the name, address, and billing information.

double printEmployee(Employee& e);

Better

```
double printEmployee(
                String Name,
                String Address,
                float illAmount);
```

Data Coupling

- Definition: Two components are data coupled if there are homogeneous data items.
- Every argument is simple argument or data structure in which all elements are used
- Good, if it can be achieved.
- Easy to write contracts for this and modify component independently.

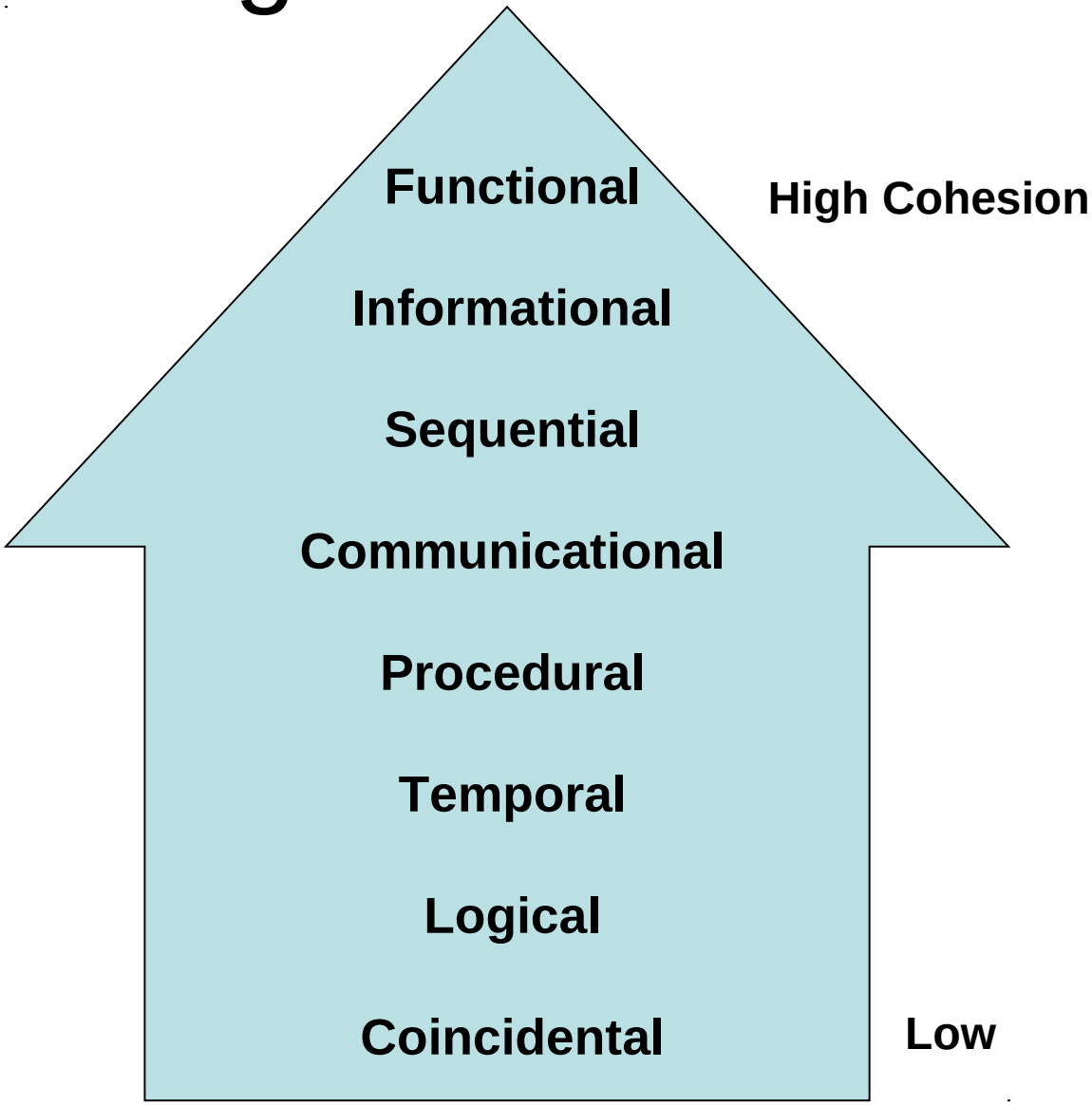
Key Idea in Object-Oriented Programming

- Object-oriented designs tend to have low coupling.

Cohesion

- Definition: The degree to which all elements of a component are directed towards a single task and all elements directed towards that task are contained in a single component.
- Cohesion of a unit, of a module, of an object, or a component addresses the attribute of " degree of *relatedness*" within that unit, module, object, or component.
- Internal glue with which component is constructed
- All elements of a component are directed toward and essential for performing the same task
- High is good

Range of Cohesion



Coincidental Cohesion

- Definition: Parts of the component performs multiple, completely unrelated actions
- May be based on factors outside of the design:
  - skillset or interest of developers
  - avoidance of small modules
- No reusability
- Difficult corrective maintenance or enhancement
- Elements needed to achieve some functionality are scattered throughout the system.
- Accidental Worst form
- Example : *an "Utilities" class*

Coincidental Cohesion - example

```
/*
  Joe's Stuff
*/
class Joe {
public:
    // converts a path in windows to one in linux
    string win2lin(string);

   // number of days since the beginning of time
   int days(string);

   // outputs a financial report
   void outputreport(financedata, std::ostream&);
};
```

Logical Cohesion

- Definition: Elements of component are related logically and not functionally.
- Several logically related elements are in the same component and one of the elements is selected by the caller.
- May include both high and low-level actions in the same class
- May include unused parameters for certain uses
- Interface is difficult to understand
  - in order to do something you have to wade through a lot of unrelated possible actions
- Example: *grouping all mouse and keyboard input handling routines*

Logical Cohesion

```
class Output {
public:

    // outputs a financial report
    void outputreport(financedata);

   // outputs the current weather
   void outputweather(weatherdata);

   // output a number in a nice formatted way
   void outputint(int);
};
```

Temporal Cohesion

- Definition: Elements of a component are related by timing.
- Difficult to change because you may have to look at numerous components when a change in a data structure is made.
- Increases chances of regression fault
- Component unlikely to be reusable.
- Often happens in initialization or shutdown
- *Example: a function which is called after catching an exception which closes open files, creates an error log, and notifies the user*

Temporal Cohesion – Example

```
class Init {
public:

        // initializes financial report
        void initreport(financedata);

        // initializes current weather
        void initweather(weatherdata);

        // initializes master count
        void initcount();
}
```

Procedural Cohesion

- Definition: Elements of a component are related only to ensure a particular order of execution.
- Actions are still weakly connected and unlikely to be reusable
- Changes to the ordering of steps or purpose of steps requires changing the module abstraction
- *Example: a function which checks file permissions and then opens the file*

Procedural Cohesion – Example

```
class Data {
public:
        // read part number from an input file and update
       //  the directory count
        void readandupdate (data&);
};
```

Communicational Cohesion

- Definition: Module performs a series of actions related by a sequence of steps to be followed by the product and all actions are performed on the same data
- Action based on the ordering of steps on all the same data
- Actions are related but still not completely separated
- Module cannot be reused

Communicational Cohesion

```
class Data {
public:

        // update record in database and write it to
       // the audit trail
        void updateandaudit (data);
};

class Trajectory {

    // calculate new trajectory and send it to the printer
    void newtrajectoryandprint();
};
```

Sequential Cohesion

- Methods are together in a class because the output from one part is the input to another part like an assembly line
- The output of one component is the input to another.
- Occurs naturally in functional programming languages
- Good situation
- *Example: a function which reads data from a file and processes the data*

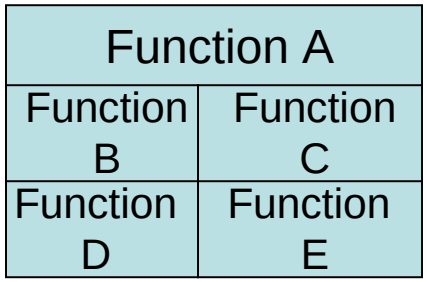
Informational Cohesion

- Definition: Module performs a number of actions, each with its own entry point, with independent code for each action, all performed on the same data.
- Different from logical cohesion
  - Each piece of code has single entry and single exit
  - In logical cohesion, actions of module intertwined
- ADT and object-oriented paradigm promote

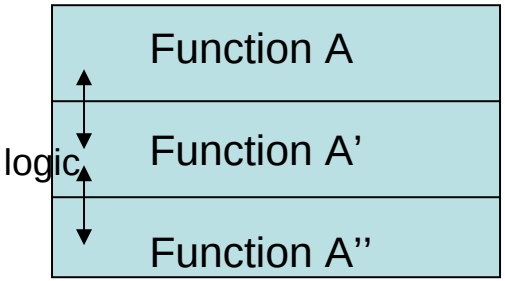
Functional Cohesion

- Definition: Every essential element to a single computation is contained in the component.
- Every element in the component is essential to the computation.
- Ideal situation.
- *Example: tokenizing a string of XML*

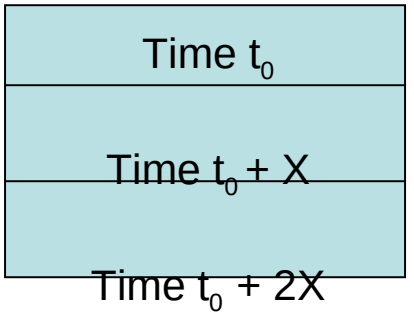
Examples of Cohesion-1



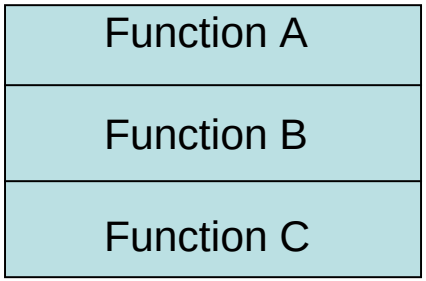
Coincidental
Parts unrelated



Logical
Similar functions

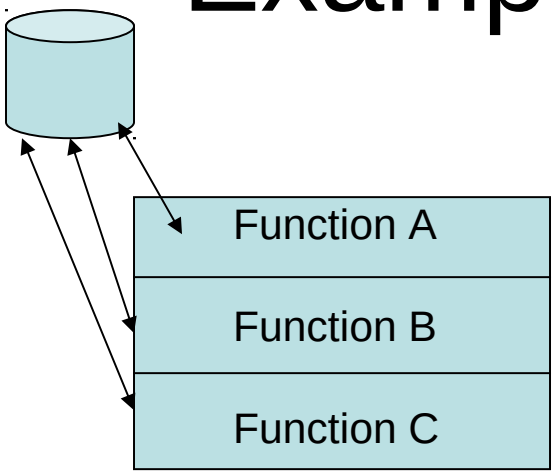


Temporal
Related by time

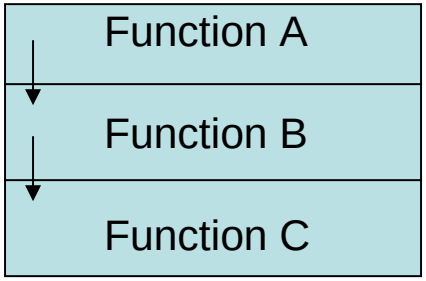


Procedural
Related by order of functions

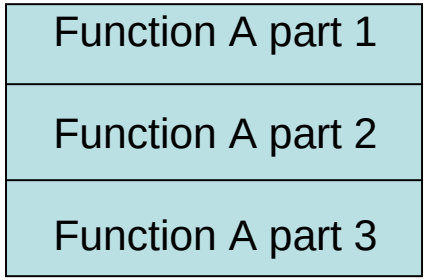
Examples of Cohesion-2



Communicational
Access same data



Sequential
Output of one is input to another



Functional
Sequential with complete, related functions

Sample Questions

- P1: What is the effect of cohesion on maintenance?
- P2: What is the effect of coupling on maintenance?
- P3: Produce an example of each type of cohesion. Justify your answers.
- P4: Produce an example of each type of coupling. Justify your answers.