Coupling and Cohesion

Dr. Michelle L. Lee

Coupling
Definition of Coupling

• The degree of interdependence between two or more components.
  – Reduced coupling → Reduced Complexity

• Make components as independent as possible.
  – Eliminate unnecessary relationships
  – Ease the tightness of necessary relationships.

• The fewer the connection, the less chance there is for the ripple effect.
  – Ideally, changing one component should not require a change in another.
  – Should not have to understand details of one component in order to change another (c.f. principle of locality).
Coupling Principles

• Create narrow connections
• Create direct connections
• Create local connections
• Create obvious connections
• Create flexible connections

Narrow (vs. Broad) Connections

• The breadth of connections is number of connections that link two components.
  – A connection that passes 15 pieces of information from one component to another is broader than a connection in which only two are passed.
  – The greater the number of parameters in a method call the broader the connection.
Direct vs. Indirect Connections

• A direct connection is one in which the interface between two components can be understood without having to refer to other prices of information.
  – A method definition including pre- and postconditions has a more direct interface than one that does not.

Direct vs. Indirect Connections

– Pass “simple” pieces of information,
– Do not pass aggregate structures unless the aggregate is a logical entity.
– Do not group elements into arbitrary aggregates just to reduce coupling - it does not work!
Local vs. Remote Connections

• Local Connection
  – All information required to understand the connection is present in the connection itself.

• Remote Connection
  – Coupled through global data.
  – May be hundreds of lines removed from a call point.
  – May be remote in time.

Obvious Connections

• Easier to understand the obscure connections
  – Avoid strange or unnatural connections.
    • E.g storing area code at the end of phone number.

• Direct access to state variables is an obscure connection.
Flexible Connections

• Systems evolve over time.
  – information being passed changes.
  – May want to switch to a different logical or physical devices.

Types of Coupling

• NORMAL
  – Data
  – Stamp
  – Control

• COMMON

• CONTENT

Acceptable

Unacceptable

(C) Michelle Lee, 1999
Normal Coupling

- Two components, $A$ and $B$, are normally coupled if
  - $A$ calls $B$
  - $B$ returns to $A$
  - All information passed between them is by parameters in the call (or a return).

- Types of Normal Coupling
  - Data Coupling
  - Stamp Coupling
  - Control Coupling

Data Coupling

- Two components are $data$ coupled if they communicate through parameters.
- Each parameter is an elementary piece of data.
- Most common form of coupling.
- Harmless if number of parameters is kept to a minimum.
Data Coupling

• Keep interfaces as narrow as possible.
• Avoid *tramp data*:
  – Data passed from component to component that is unwanted and meaningless to most.

Stamp Coupling

• Two normally coupled components are *stamp coupled* if one passes to the other a complete piece of data - one with meaningful internal structure.
  – E.g customer records
Stamp Coupling Warnings

• Do not pass aggregate structures containing many fields when only one or two are needed by callee.
  – You do not want to couple caller to the structure of the aggregate.
  – Creates dependencies between otherwise unrelated components.
• Do not bundle unrelated data.

Control Coupling

• Two components are *control coupled* if one passes to the other a piece of information intended to control the internal logic of another.
  – E.g. flags
• Downside: Caller must understand some of the details about how the logic of the callee is organized.
Control Coupling Warnings

- Avoid hybrid coupling, which results from the assignment of different meaning to different parts of the range of a piece of data.

Common (Global) Coupling

- Two components are *common coupled* if they refer to the same global data area.
- This is an undesirable form of coupling.
- Defect in one component using common global data can affect others.
- Introduces remoteness (with respect to time).
Common (Global) Coupling

- Maintenance effort is increased due to difficulty in knowing what data is used by a particular component.
- Difficult to determine what data must be changed when a component is modified.
- Also difficult to find what components will be affected by a change.

Content (Pathological) Coupling

- Two components exhibit content coupling if one refers to the internals of the other in any way.
- This form of coupling is unacceptable. It forces you to understand the semantics implemented in another component.
Determine Coupling Type

• Ask:
  – Can programmers work independently? To what degree?
  – What may change that may cause changes to multiple components?

• A pair of components may be coupled in more than one way:
  – Their coupling is determined by their worst type that they exhibit.
Effects of Object Technology on Coupling

• Object Technology adds additional relationships that often result in increased coupling.
• Proper application of design principles can reduce the complexity often found in procedural programs.
  – E.g. effective use of access control can eliminate common coupling.

Class Coupling

• The coupling between two classes is a measure of how much they depend on each other (degree of dependency)
• Addresses two related issues
  – How much do I need to understand about related classes in order to understand a particular class?
  – If changes are made in one class, what is the potential impact with respect to other classes in the system?
Class Coupling

- Coupling can range from none (independent classes) to very strong (e.g. friend classes in C++)
- Dependency between classes can be in one direction (Class A depends on B, but not conversely) or both directions
- Direct correlation between low coupling and reusability
- General goal: Minimize the amount of coupling between classes

Coupling due to Aggregation

- Nothing really new: do not access state variables directly (proper access control can prevent this).
- Use abstraction to create well-defined method interfaces that only expose logical properties of object.
Coupling due to Interface

- **Abstract Interface Coupling:**
  - One class references the operations contained in the public interface of an abstract class (weakest form of coupling)

- **Concrete Interference Coupling:**
  - One class reference the operations contained in the public interface of a concrete class (generally acceptable form of coupling)

Coupling due to Inheritance

- **Inheritance Coupling:**
  - One class is a subclass of another
    - String coupling but generally desirable (factor out commonality and promote reuse)
    - Minimize coupling introduced by inheritance by not accessing inherited instance variables directly (where possible, use inherited operations to access to inherited instance variable)
Coupling due to Inheritance

• Adds another dimension to the coupling problem.
• What you see is not necessarily all that you get.
  – Many non-apparent details.
• Inheritance forces you to look in potentially many places in order to understand a component.

Acceptable forms of Coupling in Object-Oriented Programs

• Normal Coupling (all forms)
• Context Coupling
  – The only acceptable form of common coupling.
• Type Coupling
Context Coupling

• Classes are like mini systems:
  – Have their own set of methods
  – Encapsulate a set of state variables.
  – But, classes are more than this, they define types!

• State variables in a class act like global variables in a system:
  – Local methods that depend on them are common coupled together.

Type Coupling

• Coupling that results from the use of common type declarations.
  – Probably the most common form of coupling in an object-oriented program.
Unacceptable Forms of Coupling in Object-Oriented Programs

- Content Coupling
- Common Coupling (Except for Context Coupling)
  - This means no public state variables!
  - What about protected?
    - Best to avoid
    - Encapsulated behind a protected selector method.

Content Coupling

- Internal Representation Coupling: one class has direct access to the private attribute values (instance variables) of another class
  - Very strong coupling and usually undesirable
  - Example: friend classes in C++
  - Possible application: container class and an associated iterator class (two highly interdependent classes which work closely together to achieve a common goal)
Cohesion

- The cohesion of a component is a measure of how well it fits together.
- A component should implement a single logical function or should implement a single logical entity.
Seven levels of cohesion

- Coincidental cohesion
  - The parts of a component are not related but simply bundled into a single component
- Logical association
  - Components which perform similar functions such as input, error handling, etc. are put together in a simple component.
Seven levels of cohesion

- Elements contribute to activities of the same general category.
- Activities are selected from outside the component.
- Logically cohesive components contain a number of activities of the same kind.
- We choose which pieces to use.

Seven levels of cohesion

- Temporal cohesion
  - All of the components which are activated at a single time, such as start up or shut down are put together.
  - Tend to be composed of partial functions whose relationship is that the happen to be carried out at a particular time.
  - Activities are usually more closely related to activities in other components than they are to each other.
  - Harder to reuse.
Seven levels of cohesion

- **Procedure cohesion**
  - The elements in a component make up a single control sequence
  - Control flows from each activity to the next.
  - Elements are involved in different and (potentially) unrelated activities.
- **Drawbacks:**
  - Tend to be composed of pieces of functions that have little relationship to one another (except for order and time that they are carried out).

Seven levels of cohesion

- **Communicational cohesion**
  - All the elements of a component operate on the same input data or produce the same output data
- **Sequential cohesion**
  - The output from one element in the component serves as input for some other element
  - Good coupling, easily maintained.
Functional Cohesion

- Functional cohesion
  - The parts of a functionally cohesive component contribute to a single purpose or problem.
  - Each part of the component is necessary for the execution of a single function
  - Single-purpose
    - Low-level (methods, classes)
    - Medium-level (packages, class categories)
    - High-level (systems, sub-systems)

---

Cohesion Decision Tree

---

Comparison of Cohesion Level

<table>
<thead>
<tr>
<th>Cohesion Level</th>
<th>Coupling</th>
<th>Cleanliness of Purpose</th>
<th>Maintainability</th>
<th>Understandability</th>
<th>Effect on Overall System</th>
<th>Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Sequential</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Fairly good</td>
<td>Medium</td>
</tr>
<tr>
<td>Communicational</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Procedural</td>
<td>Variable</td>
<td>Poor</td>
<td>Variable</td>
<td>Variable</td>
<td>Bad</td>
<td>Bad</td>
</tr>
<tr>
<td>Temporal</td>
<td>Poor</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Bad</td>
<td>Bad</td>
</tr>
<tr>
<td>Logical</td>
<td>Bad</td>
<td>Bad</td>
<td>Bad</td>
<td>Poor</td>
<td>Bad</td>
<td>Bad</td>
</tr>
<tr>
<td>Coincidental</td>
<td>Bad</td>
<td>Poor</td>
<td>Bad</td>
<td>Bad</td>
<td>Bad</td>
<td>Bad</td>
</tr>
</tbody>
</table>


(C) Michelle Lee, 1999

Object-Oriented Cohesion
Class Cohesion

• Class cohesion
  – Each operation provides function which allows the attributes of the class to be modified, inspected or used as a basis for service provision.

• A cohesive object is one where a single entity is represented and all of the operations on that entity are included with the object.

Class Cohesion

• Cohesion is a measure of strength of connectivity of the items within a class (binding strength)

• Whereas coupling describe the relationships between classes, cohesion describes the relationships within the class.

• Address the purpose of the class: Do the items (attributes, operations, etc.) of a class work together toward a single, common purpose?
Class Cohesive

- Highly cohesive classes are usually more easily understood and less susceptible to change (more stable)
- General goal: strong class cohesion

Types of Class Cohesion

- Single Abstraction Cohesion:
  - The class encapsulates a single abstraction (strong cohesion)
    - all items work together to achieve a common purpose
    - usually provides operations in the public interface which manipulate “hidden” representation details
Types of Class Cohesion

• Sequential
  – The items within the class must be accessed or activated in a particular order
    • may be required at some level of abstraction within the system but sequential nature should be hidden as much as possible
    • limit coupling to a class with sequential cohesion
    • “… a system decomposition based on the order in which operations are to be executed stands to suffer considerable from any change of external specifications” - Bertrand Meyer

• Temporal Cohesion:
  – Items are bound together within a class because they must be used at approximately the same time
    • example: system initialization class
    • weak form of cohesion which should generally be avoided.
Types of Class Cohesion

• Logical Cohesion:
  – There is a weak, logical connection among the items in the class but not in terms of data or control
  • example: class of mathematical functions (sine, cosine, etc.)

Class cohesion

• Cohesion means a unit represents a single part of the problem solution.
• The inheritance promotes reusability but reduce cohesion
  – It is no longer consider that object as a separate unit. All super-classes also have to be inspected if the object’s functionality is to be understood completely.