# *Lab 11 – Design Criteria: Cohesion & Coupling*

Date assigned: Monday, March 27, 2017

Date due: **Monday, March 27, 14:50**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will be able to:

* Understand the basic concepts of Cohesion
* Understand the basic concepts of Coupling
* Understand the MVC model as an example of a highly cohesive, low coupled design

Lab Set Up

1. Create a copy of this document called YourUsername\_E21\_L11.docx and fill in with your responses.

To do:

**Part A – Coupling**

1. **Stamp coupling** occurs when a method of class A has parameters (or local variables or a return value) of class B and depends on the structure of the B object (i.e. uses part of it).

For Example:

Suppose we have a class Person with a method called getBirthDate (). Suppose we now want to create another class DriversLicense with a method called isJuniorOperator() (which returns true if an individual is under 18). One way to structure this would be as follows:

boolean isJuniorOperator(Person p)

{

Date birthDate = p.getBirthDate();

// return true if birthDate is less than 18 years

// before today’s date

}

However, if we changed the getBirthDate() method of Person in some way (e.g. renamed it or changed the way we stored information about the driver), we might also have to change the

isJuniorOperator() method of DriversLicense.

How can you reduce this coupling?

**Inside the Person class you could have a method isMinor that will return true or false if they’re older/younger than 18 and that will be equivalent to isJuniorOperator. Is junior operator could just return isMinor.**

2a. **Control coupling** arises when a method does different things depending on the value of a “flag” parameter.

For Example:

In your video store, you might eventually create a method like this:

updateCustomer(int whatKind, Customer customer), where whatKind takes on the values ADD, EDIT or DELETE, and customer is used for EDIT, but is not used at all for ADD, and only the id is used for DELETE.

How can this coupling be avoided? Does it also affect cohesion? Please explain.

**This method definitely reduces the cohesion of a program. If you have a method doing various different things, it’s probably better to just write multiple methods and chose which method to call instead of always calling the same one. If you need, you could create a method that choses which method to call, so you’re always calling the same method, which would create a level of abstraction too. This coupling can be avoided by creating multiple methods and just choosing which method to call in the first logic instead of having one method perform a variety of tasks.**

2b. How can the control coupling be avoided in the example below?

public routineX(String command)

{

if (command.equals("drawCircle")

{

drawCircle();

}

else

{

drawRectangle();

}

}

**In this method, instead of choosing which method to call based on the flag passed in, you could do this logic inside the class that calls routineX. The issue with this is that maybe you’d want this logic in multiple places throughout the code because drawCircle and drawRectanlge are called in multiple different places throughout the code. You could also replace the string with an Enum to make it a little more cohesive.**

**//If the person is already setting the string, they have the logic to determine what method to call already there, so just call the method there instead.**

4. **Common coupling** arises when a method depends on a global variable.

For Example: In a video store problem, the rental rate can be set by management. One might address this by including a variable rentalRate in the class StoreDatabase - in which case code like the following might occur:

amountDue += StoreDatabase.rentalRate;

This is undesirable, because any change to the variable (e.g. giving it a different name or changing its type can “break” other classes - sometimes in startling ways (like charging someone

$3000 for a rental, because the representation for rentalRate was changed from a float representing a dollar amount to an int representing a number of cents!)

How can this be avoided?

**This can be avoided by getting rid of the global variable and passing it around through parameters to different methods. Global variables can make things really messy, so it’s better to pass them around. You can also prevent issues with globals by having strongly enforced naming standards and uses of globals. But just avoid it by passing the values through parameters.**

**//Hide the datatype and just use accessors and mutator**

**Part B – Cohesion**

1. How would you rate the following code from the point of view of cohesion? Can the cohesion be increased?

class BudgetReport {  
      
    void connectToRDBMS() {  
    }

    void generateBudgetReport() {  
    }

    void saveToFile() {  
    }

    void print() {  
    }  
}

**The connectToRDBMS could be moved to a different class. Having the DB layer code inside the BLL code means that your class is now doing multiple different things, which it shouldn’t be doing. I would move the connectToRDBMS to a BudgetReportDB class.**

2. Communicational cohesion means that all the modules that access or manipulate certain data are kept together (e.g. in the same class) - and everything else is kept out. What are its advantages?

**Less discrepancies between data and less likelihood of messing up the correct data if only one place is actually able to manipulate it.**

3. Utility Cohesion is when related utilities which cannot be logically placed in other cohesive units are kept together.Give a practical example from your everyday programming.

**Class libraries containing methods that are all related. For example, the HVK class library containing all the DB and BLL layer logic for each of the different functionalities that the HVK system needs to have**

4. Explain coincidental cohesion, temporal cohesion, and logical cohesion. Give an example of each one of them.

**Coincidental cohesion: Coincidental cohesion is when parts of a module are grouped arbitrarily; the only relationship between the parts is that they have been grouped together.**

**Temporal cohesion: Temporal cohesion is when parts of a module are grouped by when they are processed – the parts are processed at a particular time in program execution.**

**Logical cohesion: Logical cohesion is when parts of a module are grouped because they are logically categorized to do the same thing, even if they are different by nature.**

**Part C – Design study: MVC**

1. Read [this article](http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/) about MVC. Stop at the point that you learn that “Models are fat”. This is a Ruby on Rails pattern that is language and platform agnostic, so the concepts apply universally.

2. Complete the following table indicating which components are affected by which functional changes. Place an ‘X’ in the appropriate MVC columns, or provide details. The first row is completed for you as an example.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Change request | Model | View | Controller | Other (provide details) |
| Add additional help messages and prompting with mouseover/hover |  | X |  |  |
| Switch Database technologies and data access mechanism | X |  |  |  |
| Change colours used in the web page to match branding/marketing colours |  | X |  |  |
| New rule to validate customer information | X |  |  |  |
| Update an existing business process (add additional steps, invisible to the end user) | X |  |  |  |
| Add support for mobile devices (previously supported only desktop running IE) |  | X | X |  |
| Support for web pages in French |  | X |  |  |
| Change the URL used to reach specific activities |  |  | X |  |
| Create different web sites for different languages (same functionality) |  | X |  |  |

To Submit

Upload the document to the Moodle page for this course.