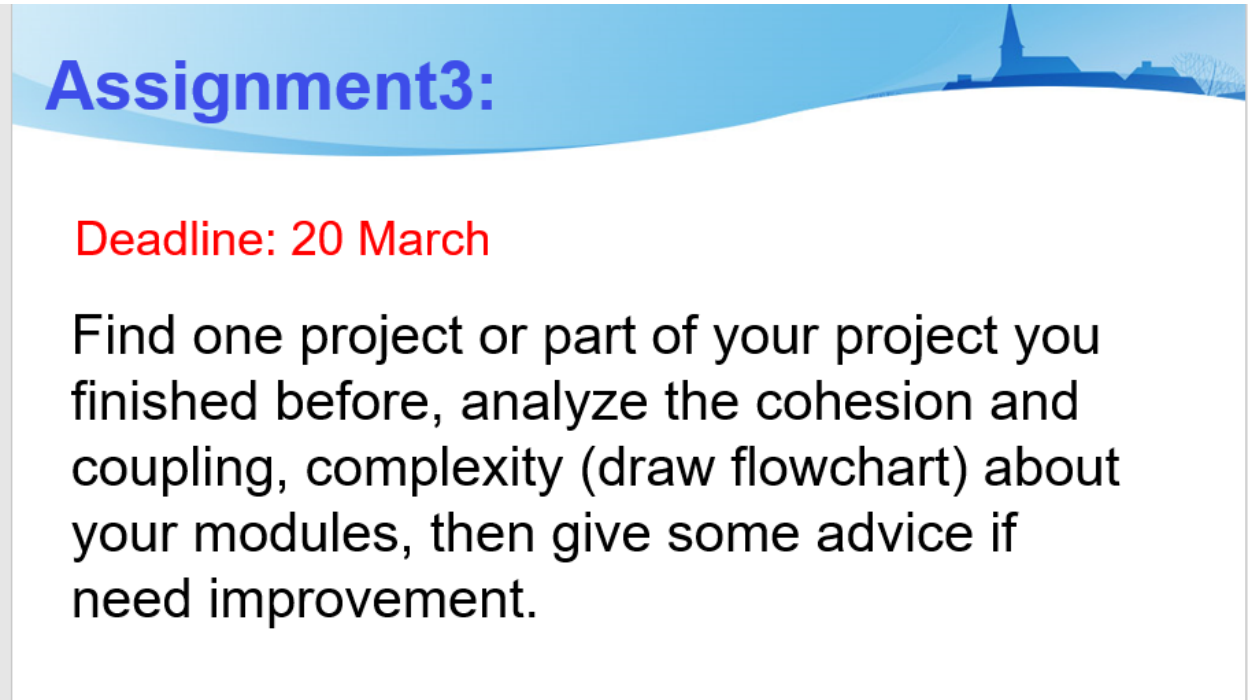
Software Engineering

Assignment 3

Name : ABID ALI

Roll : 2019380141

Send to : lining@nwpu.edu.cn , 480755534@qq.com



Solution:

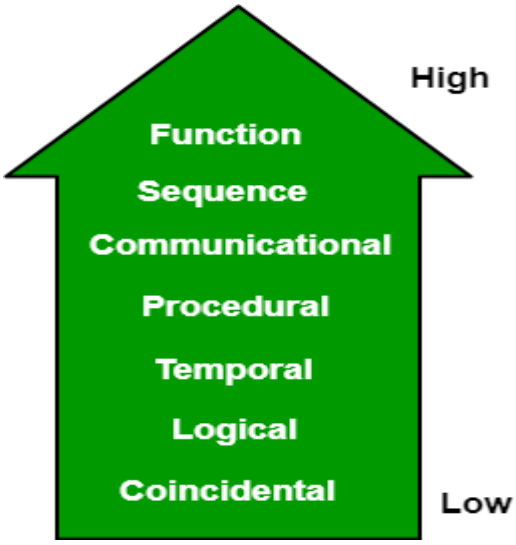
Modularization: Modularization is the process of dividing a software system into multiple independent modules where each module works independently. There are many advantages of Modularization in software engineering. Some of these are given below:

- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.

Cohesion:

Cohesion is a measure of functional strength of a module.

*A cohesive module perform a single task or function.



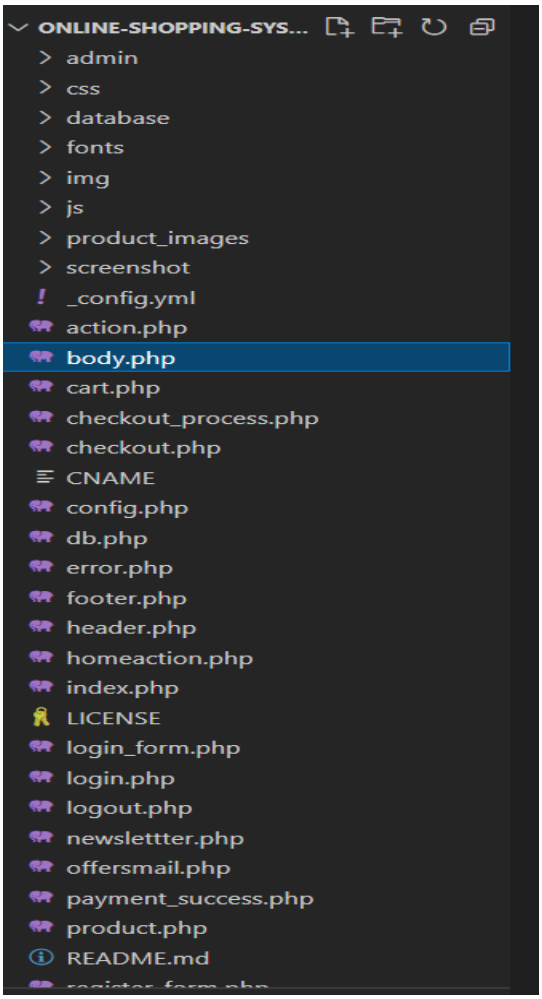
Types of Cohesion:

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.

- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

For the cohesion analysis I took different program I've been working now .

My program is using PHP to create a little web application using back(Mysql) and front (PHP). It has different classes that are performing to completing different tasks.



For the php it uses 2 classes which tasks are to describe and object “login” and “logout”, the “db” class is for connecting to database .

Different servlets classes are for implementing the different tasks according to their class name. And connect to the web application.

As you might see the files themselves are separated too and each for a specific task implementation. Unlike the program where I have a single class performing many different tasks, here I have created a several different classes, each class performing a specific specialized tasks, leading to an easy creation and modification of these classes. And it is a “High cohesion “classes.

admin_info

Field Name	Data type	Size	Constants	Description	Extra
admin_id	int	11	PRI	Admin ID	auto_increment
admin_name	varchar	100	Not Null	Admin Name	
admin_email	varchar	100	Not Null	Admin’s Email	
admin_password	Varchar	100	Not Null	Admin’s Password	

brands

Field Name	Data type	Size	Constants	Description	Extra
brand_id	int	11	PRI	Brand’s ID	auto_increment
brand_title	text		Not Null	Brand Title	

cart

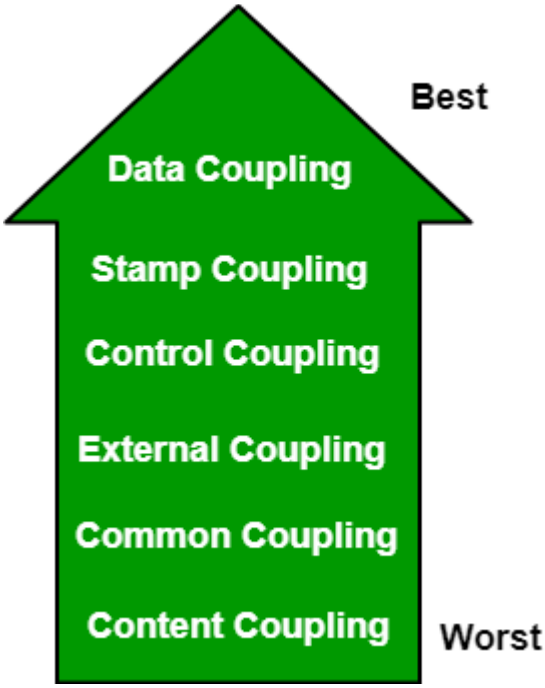
Field Name	Data type	Size	Constants	Description	Extra
id	int	11	PRI	Cart ID	auto_increment
p_id	int	11	Not Null	Package ID	
ip_add	varchar	250	Not Null	IP Address	
user_id	int	11	Null	User ID	
qty	int	11	Not Null	Quatity	

We can see that,each related attribute are in a same table or we can say in same method/function. High cohesion can be seen between a module.

Coupling:

Coupling can be defined as measuring of the degree of interdependence or interaction between the two modules.

*A good software will have low coupling.

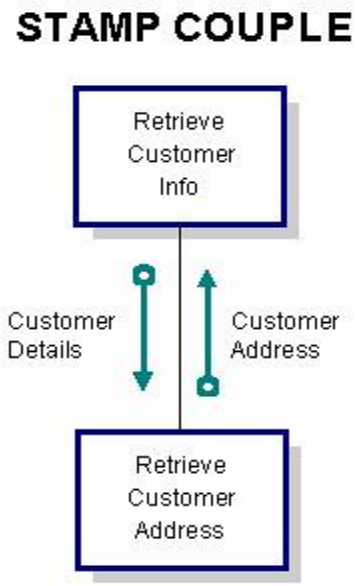


Types of Coupling:

- **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don't contain tramp data. Example-customer billing system.
- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.

- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
- **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Stamp coupling occurs between modules when data are passed by parameters using a data structure containing fields which may or may not be used.



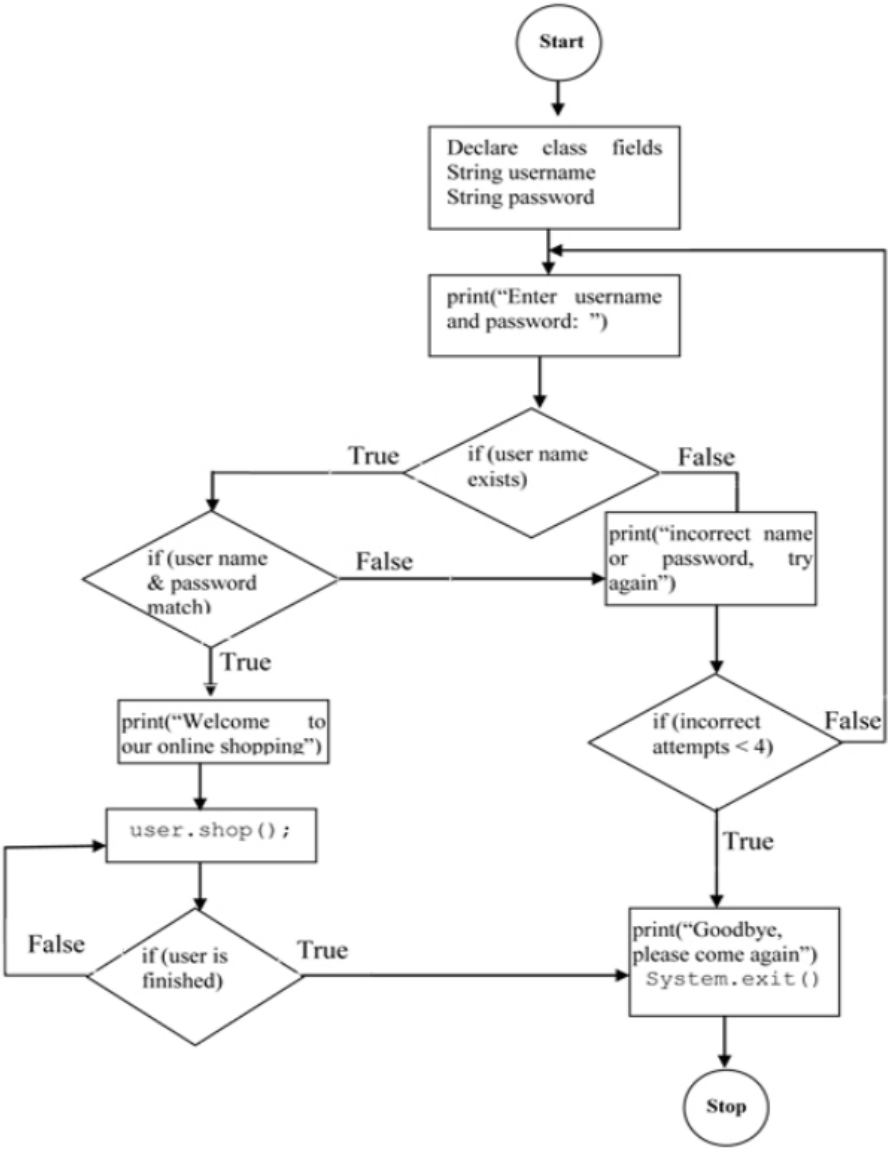
An example of stamp coupling illustrates a module that retrieves customer address using only customer id which is extracted from a parameter named customer details.

As a rule of thumb, never pass a data structure containing many fields to a module that only needs a few.

Some of the practitioners of structured design do not make a distinction between the passing of parameters in an unstructured format (as described under data coupling) and the passing of parameters in a data structure (stamp coupling).

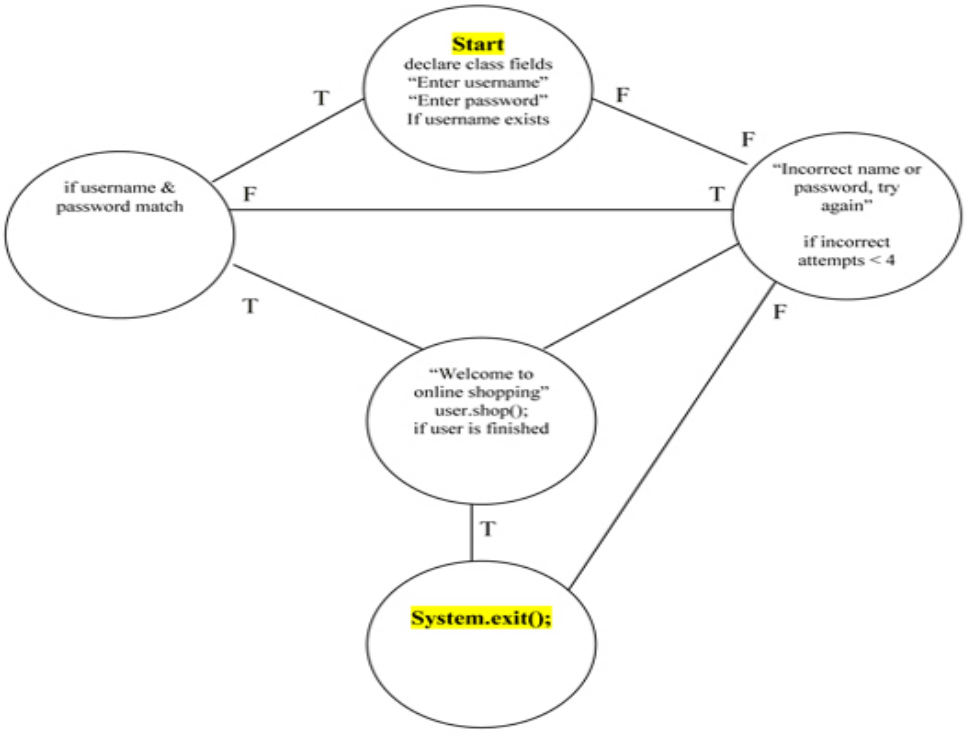
We can use also a create a "view" so that excess information is not given to the user and requested information is received by the user when certain information is passed by user.
For example ,If a person want to see the address of customer then input the name of customer .That person should not get excess information .Like Phone no. ,age and etc.

This part contains , a Person's username and password are correct before granting access to an online E-commerce site .



First of all we will create a flow graph/diagram based on the flow chart above .

So lets have a look what have we done here



Cyclomatic Complexity = number of **edges** - number of **nodes** + 2

= (7 - 5) +2 = 4

The Cyclomatic Complexity of this software application is 4

Some advice can be presented by try to keep the code more loosely coupled situation compared to present situation, then one module can perform without impacting other modules . Trying to make it more easier so that we can easily write DRY code that is easy to work with.