Textbook Assignment 8

1. Give a counter example for each of the following two basic characteristics of good software design.

Consistency across design

a) The design can be a little inconsistent where consistency is not required. Such places include places the user may not visit.
b) The consistency of the design is one of the major things that a good software requires for proper functionality but the design itself is not significant for it to be efficient
c) If there is a website where there are daily users visiting, after looking at the statistics, it is determined that only 10% of the website is being used by those daily users. In such cases, the developer will yield better results if they focus on the consistency of that 10% of the website that is being used by the users and focus less on the remaining 90%.

Completeness of the design

A) Design completing is vital and must be focused on with great attention to detail while finishing the software. After all, the main purpose of the software is to help the users and be easy enough to use and get a grasp of. Having a design that is complete defeats this purpose.

B) With what we said in the above example of the website that users use only 10% of, completeness of the design is not something the user will consider. Therefore, it won't make or break the software.

2. What is the cyclomatic complexity of the design flow shown in the following figure where the diamond shapes represent decision branches, and the rectangles are statements?

Closed figures + 1, number of nodes having conditions + 1, or number of binary branches + 1, they all yield the same answer: 3+1 = 4.

3. For each of the following 7 levels of cohesion, using an example Java code skeleton to explain the level, in terms of designing or allocating methods in a class. Why the cohesion in a higher level is stronger than the cohesion in the level below?

Functional

The cohesion is strongest in this level. All the items focus on performing a single task, because no element is doing unrelated activity. Below is a code of a program that finds the factorial of a number, as an example.

```
    static int fac(int n)
    {
    if (n == 0)
    return 1;

    return n*fac(n-1);
    }
```

Sequential

Occurs when the output of a module is an input for another module.

```
public class Whatever{


    public Whatever() {
    //whatever happens
    }

    public Whatever (Whatever w){
    //do some functions
    }
}
```

Communicational

When models use the same input and output. Communicational cohesion means they work on the same data. Push and pop of a stack is one of those that comes to mind.

STACK.PUSH(X)

STACK.POP(X)

Procedural

Occurs when all related items are performed in a certain way.

```
File file = new File("wtv");
boolean exists = file.exists();
if(exists)
{
    System.out.println("E")
}
```

Temporal

Occurs when elements of module are time-related

Logical

Occurs when modules are grouped because they user the same logic and/or perform the same type of task.

Coincidental

Occurs when there is no relationship between the modules. No need, of course, to mention that this is the weakest cohesion.

Higher system maintainability and simpler modules are some of the advantages of stronger cohesion.

4. For each of the following 6 levels of coupling, using an example Java code skeleton to explain the level, in terms of relationships between classes. Why the class coupling in a lower level is weaker than the coupling in the level above?

Content Coupling

Occurs when a module refers to the internals of another module. Considered the worst coupling

Common Coupling

Occurs when modules communicate using global data areas

Control Coupling

Occurs between modules when data are passed that influence the internal logic of a module

If and switch statements.

```
if(name = User.name)
{
    // do whatever
}
```

Stamp Coupling

Occurs when multiple modules share a common data structure.

Data Coupling

Occurs between two modules when data are passed by parameters using a simple argument list and every item in the list is used.

No Coupling

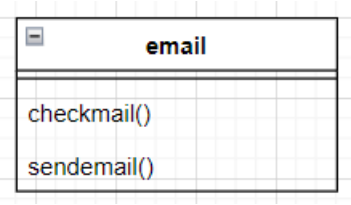
Occurs when a system's components do not have knowledge of definitions of other separate components.

There is no data security complexity in lower-level couplings. As they get more complex and complicated, the data security also increases. In no coupling, data modification can't be done.

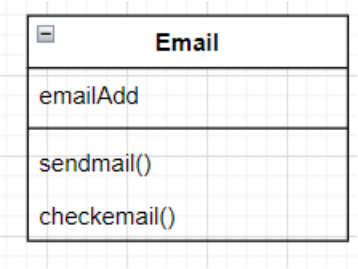
5. Explain why in class design the more methods in a class depend on common instance variables of the class, the more cohesive of the class.

Cohesion refers to what the class (or module) can do. Low cohesion would mean that the class does a broad, unfocused variety of actions. High cohesion means that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class.

Example of low cohesion



Example of high cohesion:



High cohesion is when you have a class that does a specific job. Low cohesion is when a class does a lot of jobs that don't have much in common.

So, if class design the more methods in a class depend on common instance variables of the class, the more cohesive the class.

6. In OOP, if an object a of class A wants to send a message to another object b of class B, object a has to know object b in some way. For each of the following ways for message passing defined in Law of Demeter, using a Java code example to explain how object a knows object b. Is there some other way to make object a know object b other than these which violates Law of Demeter? Give an example

- the object itself
- the object's attributes (instance variables)
- the parameters of the methods in the object

- any object created by a method in the object
- any object returned from a call to one of the methods of the object
- any object in any collection that is one of the above categories

```
public class A {

    public void method1() {
        method2();
    }

    public void method2() {
        //dosomething
    }
}
```

```
public class A {

    public void method1(B b) {
        b.method2();
    }
}

    class B {

    public void method2() {
        //do something
    }
}
```

```
public class A {

    public void method1() {
        B b = new B();
        b.method2();
    }
}

    class B {

    public void method2() {
```

```
        //do something
    }
}
```

```
public class A {
    private B b;
    public void method1() {
        b = new B();
        b.method2();
    }
}

    class B {
    public void method2() {
        //do something
    }
}
```

```
public class A {
    public void method1() {
        b.STATIC_INSTANCE.method2();
    }
}

    class B {
    public static final B STATIC_INSTANCE = ...;
    public void method2() {
        //wtv
    }
}
```