1 480 Note, May 1st 201

1.1 Introduction

60 % of a program life time comes from

- feature change
- bugs

how to improve your code. using the Quality Attributes

- Flexibility
- Reusability

- These two attributes hold the highest positions in all the attributes list
How do I know if the design is reusable
- We will use the metric to measure that

- Coupling
- Cohesion
- Corresponding

1.2 Metrics

- Measure for how much a class is intra-related - Example:
- Hotel Reservation class: Fields

- Date stat
- Date rad
- String name
- int billingCard
- Room roomtype

method

- reservation(Date, int, int)
- checkin()
- checkout
- addRoomCharges

Graph that show the relations between fields and methods

What is the benefit of class doing one thing –Answer: you can reuse it easlity, if you have a bug, you know where to find it

Cohesion
Definition:???
- Low cohesion is bad
- High cohesion is good

Coupling
Definition: How much are classes related, or depend on each other
- Example:
if we have classes of A,B,C,D,E and they all related to each other. If we modify one class, we will break everything
-Another example: HTML is a standard, and Microsoft has an extension of HTML, and only IE understands it, you write a program that IE can only understand it. So your website is tied to Microsoft, if Microsoft changes anything, you have to change it too.
- **With Coupling we want:**

- high coupling is very bad
- low coupling is very good

- **Conclusion:** To make the program good(flexible ,a d Reusability): we need to aim for low coupling and high cohesion.

Example for low coupling or more flexible: void foo(ArrayList¡Integer¿): this is not good because it is too narrow, so instead of using ArrayList, we can use List, or Collection which is the interface. So that when you change the passed down object, you don't have to change too much.

Correspondence:
Definition: How much the require match the requirements. How much it match the change of requirement
Example: if you ask web designer for minor change, and it takes too much time and effor, then the design is very bad. The programmer didn't think much about the changes.
Good Correspondence

- Minor requirement changes only needs low cost in design
- Major changes changes need high cost to design

1.3 Design Principles

- Encapsulation
- Information Hiding
- Abstraction
- Inheritance
- Polymorphism
- Substitution

Encapsulation
Definition: is the principle of bundling together the data and code that works on that data.
Example: A class that has public fields, public methods. That's encapsulation but no information hiding. In C, global variables are encapsulation but not information hiding.

Information Hiding: is the principle of hiding the design decisions which are most likely to change **Example:** put every fields in your class to be private. Sort(int[] arr)
Abstraction: the priciple of describing the essiential features of a type without the inessential details. **Example:** List oppose to ArrayList, Parent object as oppose to the subobject **question:** In java, we use classes to do encapsulation and information hiding **Note**: you always tend to lean to information hiding for a better solution

Inheritance and Polymorphsm: They are very close term. Inheritance means sub-class inherit methods and fields from the parent class. Polimorphsm is more like when you can have an array of subtract class(Animal), in that array, there are many many type of animals like dog, cat, turtle. **Substitution**: ???
Pre and post condition is a very important part in Interface.

SO TIRED

Example picture: