# Lesson 7 Notes and Examples

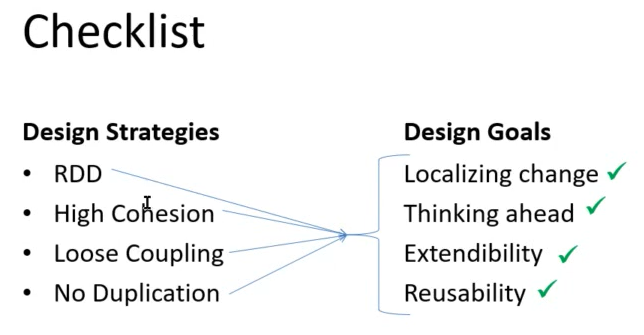
## Goals for good design

1. Design
   1. Example you need a storyboard to know how to get there
      1. Example for storyboarding that Mr. Wilder teachers is Google Design
      2. Always design before typing
2. Test
3. Code

\*\* It will never be “ALL DONE”

## Responsibility Drivers Design (RDD)

1. Abstraction
   1. Looking at the big picture
   2. What language should I use? Then which classes should I have?
2. Modularization
   1. What are all the classes, methods and variables do they use – it is broken down
3. Cohesion
   1. Aim for high cohesion
   2. Means every instruction/every unit should do one job
   3. Example
      1. If you are making an online chess games then we should have a package ca.bcit.chess
         1. This package should only do the chess game
         2. We will have a chess board
            1. Color
            2. 8 by 8 squares
         3. Work class and move method
            1. That should only know how to move rooks
            2. Should also know how to move queens
      2. Every piece should have one job
      3. One method should do one thing
      4. Instructions should do one thing
      5. One class should do one thing
   4. Example
      1. One class for Car
         1. Will only do car stuff
      2. One class for temperature
         1. What is the temperature in the car (separate from car class)
      3. Every unit has one job
   5. We want **high cohesion**
   6. Low cohesion is when a method or a class is doing more than one job
      1. Example
         1. If you are trying to figure out if a number is prime.
         2. If a method isPrime it should just determine if the argument is primary then return true or false but instead what people do for starting is your isPrime method asks for a number, validates the number if it is a positive integer greater than 1 then it determines if it is prime or not then it prints some stuff to the screen. This is not what it should do. It should only check if the number is prime and say True or False if it is not.
4. Coupling
   1. Want low coupling
   2. How related are two classes, methods or variables or two packages
   3. You do not want them to be too related
      1. Example
         1. Jason wilder is a teacher he was also a program hear last year so if he quit BCIT they would lose an instructor and program head – they are coupled
      2. Example
         1. If you have two classes and you change one then other class needs to change – this is bad
      3. Example
         1. When they made the tesla electric car. You didn’t have to re-learn how to drive. It has a steering wheel when you turn it left the car turns left, when you turn it right the car turns right, when you push the peddle on the right the car speeds up. Even though a lot of things have changed they have decoupled the user interface from the motor end. You don’t have to drive it like an army tank. Just because some parts of it changed you don’t have to change all of it.
      4. Example
         1. If another instructor had to teach this class like Paul then everything would be like normal.
      5. Example of tight coupling
         1. If I left my house and never came home my family will be greatly effected compared to BCIT.
   4. If you have to change one class then you do not want to have other classes/methods to change. You need to reseparate the code from the interface itself
      1. Example
         1. If we go to McDonalds and we get a iceream then we get it
         2. Even if they get a new supply from a new ice cream machine
         3. If for example they change it to a self serve icecream then that is totally different and we do not want this.
5. Avoid code duplication
   1. This also include avoiding magic numbers like when you have repeated numbers throughout the class
   2. We do not want any type of publication anywhere
   3. If you have to change the same value in multiple areas you might miss some changes

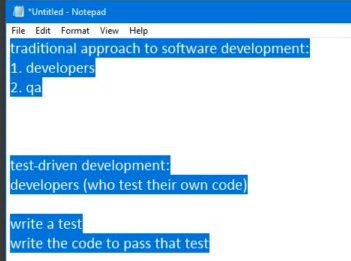


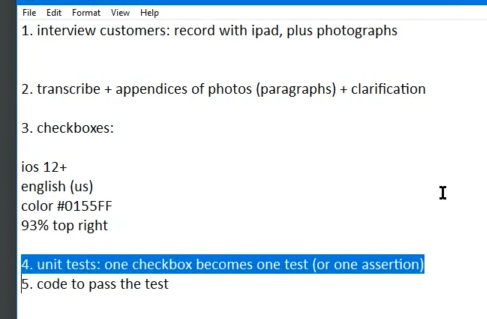
## Testing

* We have to get good at testing
* Logical errors can get brutal
* Its hard to get is 100% error free but its easier to get it 95% error free
* The last 5 to 10% takes 10 to 20 years to get good at

## JUnit

* J stands for Java
* Junit testing should be done during development
* Example
* bad
  + - Typical student will code them comment later
    - Code then test later
  + Good:
    - Comment as we go
    - Test as we go
  + We build a bridge first then we test it – this doesn’t make sense then you will have to start again – put a bit of foundation down then test the build a bit then test etc.
* Example
  + We create a empty array for all provinces
  + We test the code see if it has all the provinces it will initially have error because nothing has been added yet





* Number 4 and 5 keep repeating until done

## Design by Contract

* This means every single method completely distrusts all the arguments and validates them
  + We check for if is null or is blank
  + It will check all the inputs it will crash on purpose and show you where it has crashed
* Example
  + If you say you need a day in the year from 1 to 365 and somebody put sin 500 then before it returns anything it will check it and validate
  + Needs to make sure it follows all the rules

## Print statements

* Could check by doing print statements by when you are done you need to make sure you get rid of them once done