- **TDD Nuggets**
  - Test first or test last
    - Ideally test-first which challenges you to think about the class interface thoroug
    - However in reality it's a mix. Sometimes you begin with design discussion, implemen the basic skeleton first and move towards creating tests
  - Design Principles used most frequently
    - ➜ YAGNI (You ain't gonna need it
    - ➜ DRY (Don't Repeat Yourself
    - ➜ SRP (Single Responsibility Principle)
    - Less number of lines in a method, less methods and more classe
  - Testability and Design Evolution
    - Design
      - ➜ Strategic Design
        - High level design of any system. Defines major architecture components and the interaction between them.
        - Does not get into the details of the implementation
        - In TDD the strategic design can be done on a whiteboard or a piece of paper.
      - Tactical Design
        - * Defines how exactly the component is going to work.
        - * The tactical design contains each and every minute detail of that component.
        - happens while developing the application through tests
      - TDD
        - In reality it's difficult to evovle a design with just tests without considering the strategic design in mind. If not done properly you may get into mess and create classes which may not make much in the bigger picture. You may reach to a situation where refactory also doesnt work anymore.
      - Design Patterns
        - Don't insert Design Pattern Forcefully
        - Create Simple Design At First
          - Look for evolving design patterns in code
    - Testability of the class
      - Testing method also should follow SRP
      - Test method also should affirm the atomocity of a test
      - At most there should be 1-2 assertions in a test method
    - Identify abstractions per class
      - Write as if you are writing in English
        - If something is not readable, refactor it
          - create methods, classes
  - Test Smells
    - > 200-300 lines of code for a tes
      - Problems in design probably
    - test data within a test class
      - pollutes the test class itself
      - Instead use ObjectMother
  - Testing Challenges
    - Testing of private methods
      - If testing is an issue make them default scope
      - If reusable, refactor and put them as public methods
    - Should we test setters or getters just for the heck of increasing the Testabil
      - Groovy doesnt care about private/protected part while testing. So you may not need to have setters or getters
      - You can create a simple class which just tests the setters getters mechanically
      - Why not public fields
        - Pollutes the namespace
          - Increases Class Length
          - Lowers Code Coverage
        - Difficult to track who modified it and when
        - Difficult to provide setters later
          - Possible in Groovy
          - New Generation Languages thinking on lines of developer
    - Testing for void methoc
      - Test for behavior change instead of returned value
  - Rule Book
    - Not Necessary To Follow All The Rule
    - Experience and Decide