Agile & Extreme Programming

Scrum Practises

Principles:

- Transparency: Common understanding between all members
- Inspection: Frequently inspect progress towards sprint goal
- Adaptation: Adapt quickly as required

```
Scrum practises
        Sprint Planning - Product Sprint Backlog
        Sprint - Work on backlog
        Daily Scrum - Report progress in last 24hr & next 24hr goals
        Sprint Review - Review sprint with stakeholders
        Sprint Retrospective - Review sprint with team to find issues

Scrum artifacts
        Product Backlog - Development goals & req
        Sprint Backlog - Product backlog to be completed this sprint
        User Story - what, why, how for a feature from view of user
        Increment - Sum of all product backlog items complete in sprint
        Definition of Done - Common point quality and expectations
```

"***As a*** *<role or persona>*, ***I can*** *<goal/need>* ***so*** **that** <why>"

- GIVEN: A set of initial circumstances (e.g. bank balance)
- WHEN: Some event happens (e.g. customer attempts a withdrawal)
- THEN: The expected result as per the defined behavior of the system

ID	GIVEN	WHEN	THEN
01	User balance = $23	User asks to withdraw $20	Withdrawal is authorized AND User balance is now $3

Extreme Programming

12 practices:

- 40 hour week - programmer welfare
- metaphor - guide development w/ shared story of sys functions
- simple design - design as simply as possible, simplify whenever possible
- collective ownership - anyone can change any code anytime
- coding standards - code is written to standard to enhance understanding
- small releases - min product, then release new ver on short cycle
- continuous integration - integrate & build many times a day
- refactoring - restructure sys to improve design, simplicity & flexibility
- planning game - determine scope of each iteration w/ customer
- testing - write unit tests before coding
- on-site customer - include real, live user on team, available full time
- pair programming - all production code is written w/ 2 programmers @ 1 machine

State-Based Testing

- **Steps:**
  - **set up software into a known state**
    - **e.g., initialize variables**
  - **trigger transitions to cause state changes**
    - **e.g., call methods to change variables**
  - **verify the actual arrived state is expected**
    - **e.g., set if actual values in variables meet expectations**

Black Box Testing

- **Example test cases:**
  - **be systematic about what to test,**
    not knowing the internal code

	Addends	Sum	Description (also check commutative)
2	3	5	something simple
99	99	198	large positive pair
99	-14	85	large positive plus negative

- Big Bang testing: test whole completed program
- Top down / Bottom up: Test highest/lowest levels first.

"Dependency Injection"

```
public class ExampleService {
    private DataSource theDataSource;
    ...
    public ExampleService(
        DataSource aDataSource ) {
        theDataSource = aDataSource;
        ...
    }
    public void doService() {
        ...
        ... = theDataSource.getInfo();
        ...
    }
    ...
```

data passed as parameter

alternatively, construct what this class depends on outside the class