Johan Seland

Introduction to Software Testing

Geilo Winter School 2013

Bio – Johan Seland

Education:

- 2003: MsC (UiO) – Post-Processing of 3D MRI images
- 2008: PhD (UiO) – GPGPU Computing

Work Experience:

- 1997 – 2000: Climbing Instructor Tyrili Klatresenter
- 2000 – 2003: Developer Net Fonds ASA (Financial Industry)
- 2003 – 2004: Developer Lenco Software (Defense Industry)
- 2007 – 2012: Research Scientist SINTEF Applied Mathematics
- 2012 – Today: Research Manager SINTEF Applied Mathematics

Talk to me about: Programming, Sci-Fi, Computer Games, Cats, Kids, Skiing, Cycling,

Overview of my Geilo Lectures

Goal: Describe a set of technical and legal tools to facilitate software testing, sharing and inspection.

- Lecture 1
  - Introduction to testing for scientific software
  - Test Driven Development
  - Exercise: Bowling game Kata
- Lecture 2
  - The test pyramid
  - Test automation
  - Testing scientific software
  - Legacy code
- Lecture 3
  - Install and use Jenkins on your code
  - Laboratory: Add tests to your code
- Lecture 4
  - Using the cloud
  - Software licenses

The Scientific Method

- *"a method or procedure that has characterized natural science since the 17th century, consisting in systematic observation, measurement, and experiment, and the formulation, testing, and modification of hypotheses"*

  Oxford English Dictionary

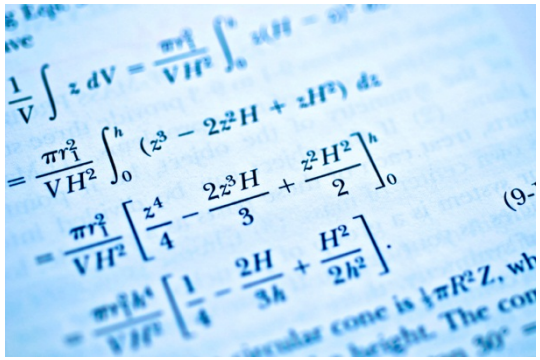
- *Another basic expectation is to document, archive and share all data and methodology so they are available for careful scrutiny by other scientists, giving them the opportunity to verify results by attempting to reproduce them.*

  Wikipedia (on Scientific Method)

- Do your code and numerical experiments live up to this standard?

Test protocols are abundant in science

- Experiments follow protocol
  - Uncertainties, control groups, regression analysis log-books etc.
- Theorems must have proof



- Testing is **MUCH** cheaper for software than any other discipline
  - Testing is cheap to automate

Software Engineering has standards too!

- SWEBOOK (Software Engineering Body of Knowledge) swebook.org
  - IEEE/ACM/ISO Standard
  - New revision in 2013

- ITIL (Information Technology Infrastructure Library)
  - ISO Standard for *Software Management*

- Capability Maturity Model Integration (CMMI)
  - Standard for measuring and improving the software development process

- IEEE Software Document Definitions (SDD)
  - Standards for *documents* describing architecture and application design

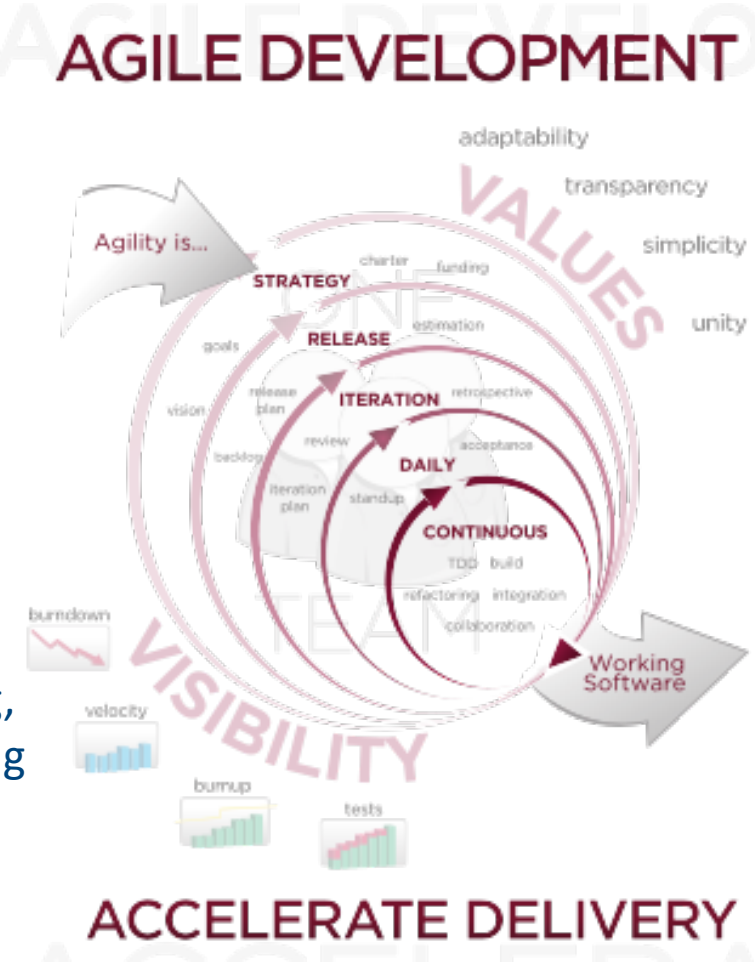
- Not suitable for small teams or projects
- Needs backing from management/institution

Agile Software Development

- Lightweight development processes
- Iterative and incremental development
- Self-organizing teams

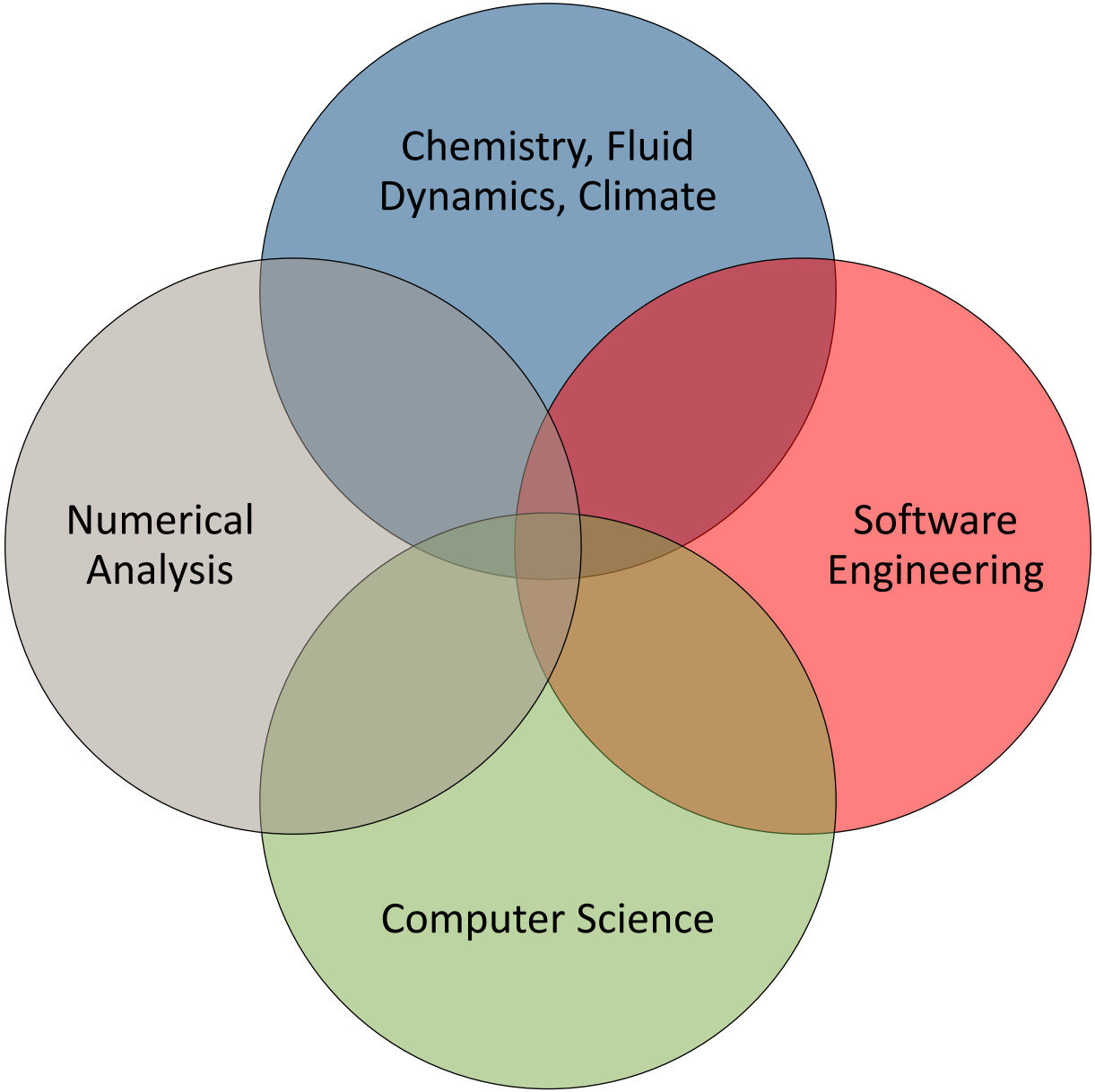
- Relies heavily on best-practices
  - Test-Driven Development, Pair Programming, Code Reviews, Continuous Builds, Refactoring

- Example Methodologies:
  - SCRUM, Kanban, Extreme Programming,



What is special about scientific software

- Often not the real deliverable
  - It is the paper or thesis (with strict deadlines)
- Developer is often the end-user
  - Requirements are made up as we go
- Correct behaviour is unknown
- Usually written by very small teams
- Life-time of software is unclear
- Usually not written by software engineers


Chemistry, Fluid Dynamics, Climate
Numerical Analysis
Software Engineering
Computer Science

What is software testing?

Software testing is the process of validating and verifying that a computer program:

- Meets the requirements that guided its design and development
- Works as expected
- Can be implemented with the same characteristics
- Satisfies the needs of stakeholders

Two approaches to testing:

- **Black-Box Testing**: Does the software produce correct output for a given input
  - The results you publish

- **White-Box Testing:** Test internal structure of a program

Software Testing

- Is your software correct the first time around?

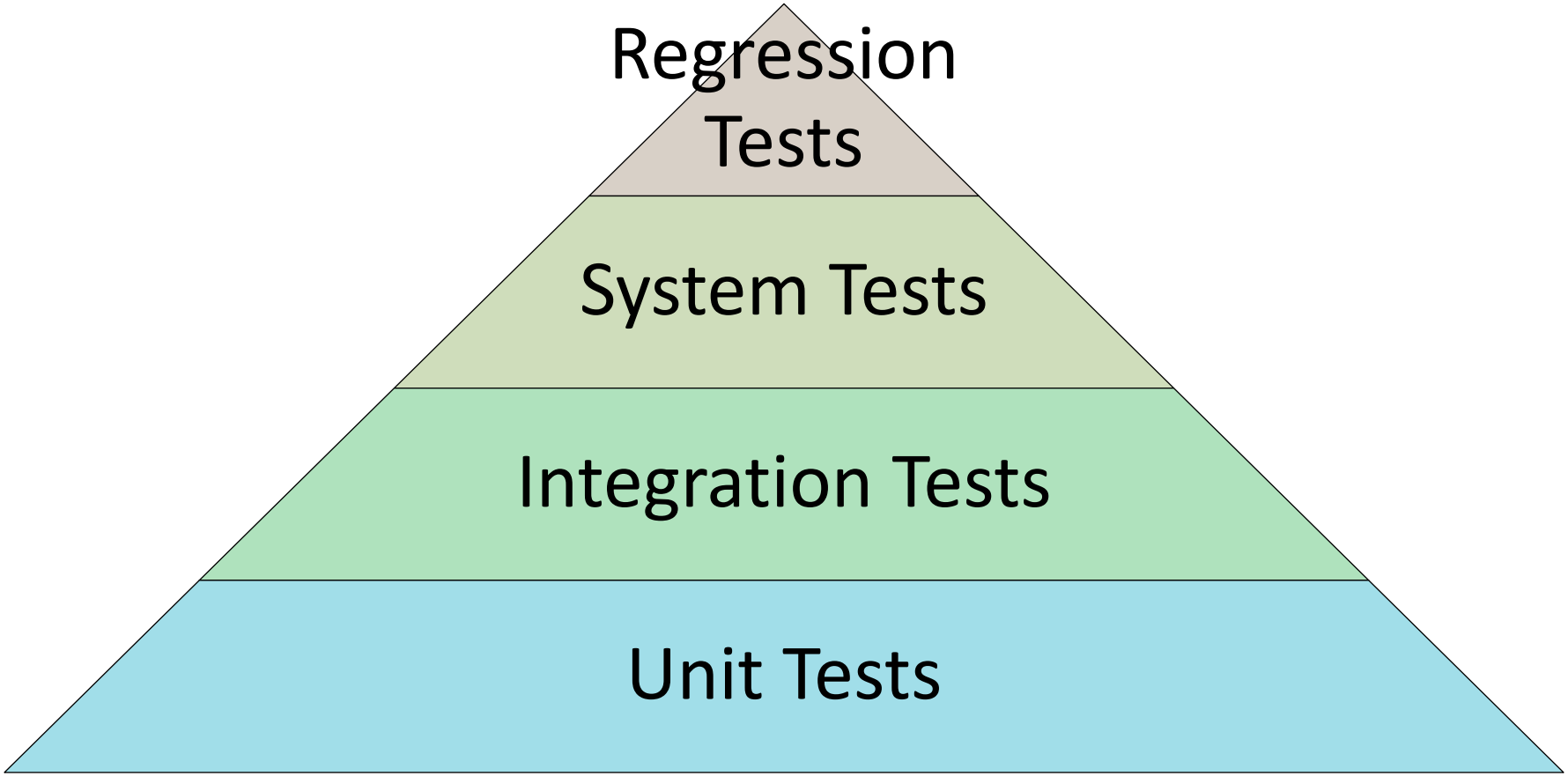
- Scientific Software has many requirements
  - Research goal is often the correctness/suitability numerical methods

  - Correctness of implementation of numerical method
  - Correctness of input parsing and output writing
  - Speed and memory requirements

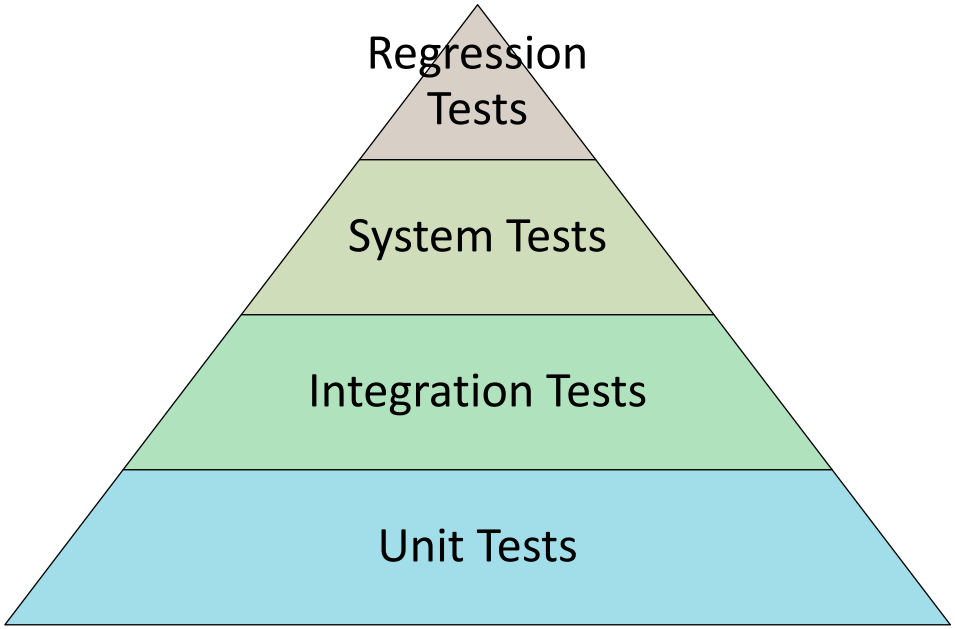
  - Stability/robustness
  - What about hardware failures?

- You probably do ad-hoc testing all the time!
- **You need a testing strategy**

The Test Pyramid



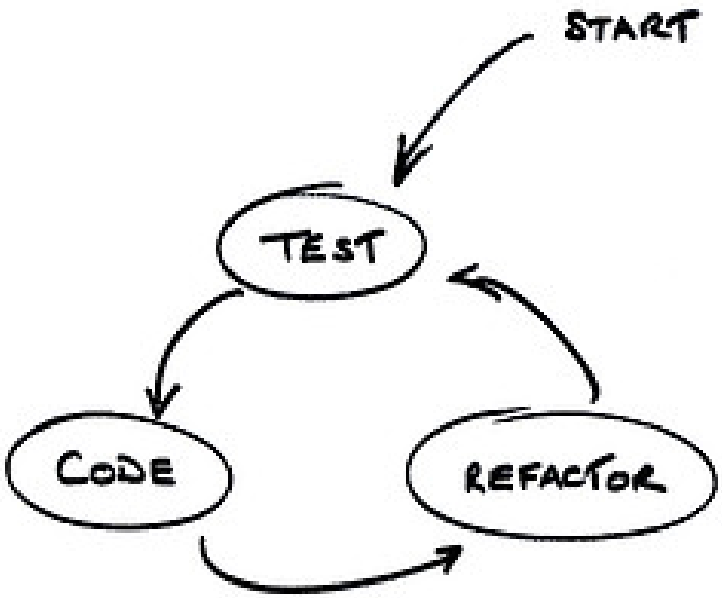
The Test Pyramid



- **Unit Tests:**
  - Tests individual units of code
  - Function/Class level
- **Integration Tests:**
  - Tests how components communicate
  - Example: Is grid correctly initialized from init file?
- **System Tests:**
  - Are results valid?
  - Performance/Scaling Tests
- **Regression Tests:**
  - Test for old bugs

What is Test-Driven-Development

- A *methodology* for software development
- Based on writing tests **FIRST!**
- "Rediscovered" in 2003 by Kent Beck
- Widely adopted by the software development community
  - Part of Agile development
- TDD = Test Driven Development



The rules of TDD

1. You are not allowed to write any production code unless it is to make a failing unit test pass

2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures

3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test

What is a Unit Test?

- A small function testing a single unit of source code
  - Unit = Function or small class

- Typical steps
  1. Create and initialize object
  2. Call method
  3. Check result

- Test are automatically discovered and executed by a *test framework*
  - More on these tomorrow

- **PITFALL: Making too large and slow UNIT tests!**

Example using Google Test for C/C++

```
int factorial(int n); // Returns the factorial of n

// Tests factorial of 0.
TEST(FactorialTest, HandlesZeroInput) {
    EXPECT_EQ(1, factorial(0));
}

// Tests factorial of positive numbers.
TEST(FactorialTest, HandlesPositiveInput) {
    EXPECT_EQ(1, factorial(1));
    EXPECT_EQ(2, factorial(2));
    EXPECT_EQ(6, factorial(3));
    EXPECT_EQ(40320, factorial(8));
}
```

Why?

Studies show:

- Total code implementation time is shorter
- Code quality increase
  - Code is more modular and extensible
- Inconclusive if developer productivity increase

- Code and tests are clearly separated
  - Avoids #ifdef DEBUG, printfs etc
  - Tests continue to be run
- Test as communication of intent
- Tests as documentation
- Tests as contract
- Measure of progress

Nagappan, Maximilien, Bhat, Williams: *Realizing quality improvement through test driven development: results and experiences of four industrial teams*
Turhan, Layman, Diep, Erdogmus, and Forrest Shull. *How Effective is Test-Driven Development?*

TDD Excuses

- You are smart and don’t make mistakes
- You want to get your idea into code **NOW!**
- My code is too simple to test
- My code is too complex to test
- I am just making a very small change

TDD Criticism

- Not a silver bullet
  - Does not create good-design
  - Does not measure performance
  - Does not remove the need for higher level tests
- Make large scale changes/redesigns harder
- Tests can take a long time to run
  - So they end up being disabled
- Encourages modular code and late-binding which may be slow

When to **NOT** write tests (Cowboy Coding)

- Graphical User Interfaces?
- When the code is clearly throw-away code
  - Prototyping, Bug-Reports
- For learning languages and frameworks
- When coding for fun

The rules of TDD

1. You are not allowed to write any production code unless it is to make a failing unit test pass

2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures

3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test

Code Katas 型

- Japanese for choreographed patterns of movements
- Widely used for practice in martial arts
- Also used to practice *programming*
  - The goal is the practice
  - Not the resulting software
- We will do the *Bowling Game Kata* to practice TDD

The Bowling Game

- Dr. Holms has a bowling hall in the basement!
  - Will anyone beat André this year?

Requirements:

- Write class **Game** with two methods:
  - **void roll(int pins)** is called for each ball. Argument is the number of pins knocked down.
  - **int score()** is called only at the end of the game. It returns the total score for the game.

- We will implement this as a Python program

Based on Uncle Bobs (Robert Martins) famous exercise

Scoring in bowling

1	4	4	5	6	/	5	/	X	0	1	7	/	6	/	X	2	6
5		14		29		49		60	61		77		97		117	133	

- The game consists of 10 frames as shown above. In each frame the player has two opportunities to knock down 10 pins. The score for the frame is the total number of pins knocked down, plus bonuses for strikes and spares.

- A **spare** is when the player knocks down all 10 pins in two tries. The bonus for that frame is the number of pins knocked down by the next roll. So in frame 3 above, the score is 10 (the total number knocked down) plus a bonus of 5 (the number of pins knocked down on the next roll.)

- A **strike** is when the player knocks down all 10 pins on his first try. The bonus for that frame is the value of the next two balls rolled.

- In the tenth frame a player who rolls a spare or strike is allowed to roll the extra balls to complete the frame. However no more than three balls can be rolled in tenth frame.

The Movements of the Kata

- We will implement five tests in total
  1. test_RollAllZeros
  2. test_RollAllOnes
  3. test_RollSpare
  4. test_RollStrike
  5. test_PerfectGame

- We will **NOT:**
  - Check for valid rolls
  - Check for correct number of rolls and frames
  - Provide intermediate score

- We **WILL:**
  - Strictly be following TDD rules

TDD in Python

- Python comes with the unittest-module
  - **import unittest**
- A test class must inherit from unittest.TestCase
  - **class TestBowlingGame(unittest.TestCase):**
- unitTest.TestCase provides assert-methods
  - **self.assertEqual(0, game.score())**
- Test methods **MUST** start with **test_**
  - **def test_GutterGame(self):**
- Execute tests from the command line
  - **nosetests bowlingGame.py**

- Clone from **git://github.com/johanseland/BowlingGameKataPy.git**

Object Orientation in Python

- Classes are defined by the **class**-keyword
- No private/protected members
  - By convention members starting with an underscore are private
- All methods must take **self** as first parameter

```
class Earth():
    def __init__(self):
        self._answer = 42

    def compute(self, years):
        return self._answer
```

Organization of Exercise

- Make groups of 2-3 people on one laptop
- Complete **ONE** test and pass laptop to next person
  - Next person starts by cleaning up the working code
- **ONLY WRITE ENOUGH CODE TO MAKE TESTS PASS**