Test Driven Development

Lect. PhD. Arthur Molnar

Babes-Bolyai University

arthur@cs.ubbcluj.ro

Overview

Consultations schedule

- Each professor has appointed consultation hours every week
- This is the time and place to ask for extra help for the course
- There is no grading!
- In order to attend, send an email to your professor using your @scs username and get their confirmation before attending
- Schedule is at http://www.cs.ubbcluj.ro/studenti/tutoriat/orarul-consultatiilor/

My consultation hours

Each Thursday, in Campus C406, starting 12:00 (send email at least 24h beforehand!)

Test Driven Development Steps

Test Driven Development (TDD)

TDD requires developers to create automated unit tests that clarify code requirements before writing the code.

- Steps to apply TDD[1]:
  1. Create automated test cases
  2. Run the test (will fail)
  3. Write the minimum amount of code to pass that test
  4. Run the test (will succeed)
  5. Refactor the code

[1] Kent Beck. *Test Driven Development: By Example. Addison-Wesley Longman, 2002*. See also Test-driven development. http://en.wikipedia.org/wiki/Test-driven_development

Writing functions for TDD

1. Create a test
   - Define a test function (*test_f*()) which contains test cases written using assertions.
   - Concentrate on the **specification** of **f**.
   - Define *f*: name, parameters, precondition, post-condition, and an empty body.

Writing functions for TDD

2. Run all tests and see that the new one fails
   - Your program has many functions, so it will also have many **test functions**
   - At this stage, ensure the new **test_f() fails**, while previously written test function pass
   - This shows that the test is actually executed and that it tests the correct function

Writing functions for TDD

3. Write the body of function **f()**
   - Writing the test before the function obliged you to clarify its specification
   - Now you concentrate on correctly implementing the function code
   - At this point, do not concentrate on technical aspects such as duplicated code or optimizations

Writing functions for TDD

4. Run all tests and see them succeed
   - Re-run the test you created at step 1
   - Now, you can be confident that the function meets its specification

Writing functions for TDD

5. Refactor code
   - **Code refactoring** is a "disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior"[2].
   - **Code smell** is any symptom in the source code of a program that possibly indicates a deeper problem:
     - **Duplicated code**: identical or very similar code exists in more than one location.
     - **Long method**: a method, function, or procedure that has grown too large.

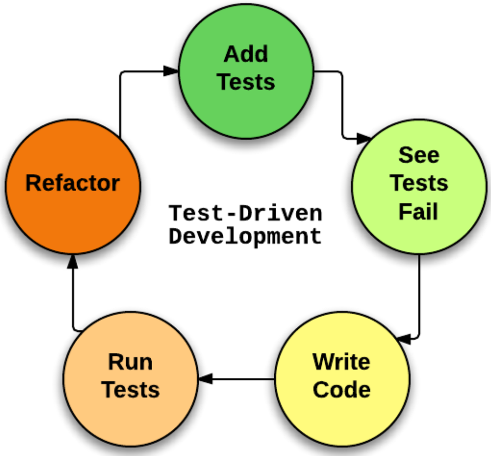
[2] Martin Fowler. *Refactoring. Improving the Design of Existing Code*. Addison-Wesley, 1999. See also http:// refactoring.com/catalog/index. html

Writing functions for TDD

Discussion

How do I know my tests are good enough?

Test Driven Development (TDD)



[3]

[3] http://joshldavis.com/2013/05/27/difference-between-tdd-and-bdd/

Demo

Test Driven Development

07-TestDrivenDevelopment-1.py

Test Driven Development

08-TestDrivenDevelopment-2.py

The assert keyword

- You should only use it within test case functions
- It is followed by an expression that is evaluated to either True or False:
  - If it evaluates to **True**, program execution continues normally.
  - If it evaluates to **False**, program execution halts with an **AssertionError**

Thoughts on TDD

- TDD is designed to take you out of the mindset of writing code first, and thinking later
- It forces you to **think** what each part of the program has to do
- It makes you analyse boundary behaviour, how to handle invalid parameters before writing any code

Thoughts on refactoring

Discussion

What do you think refactoring is good for?

What you can look to refactor

- Eliminate duplicated code by creating a new function
- Make long functions simpler by ensuring that each function only does one thing!
- Rename variables to meaningful names

Demo

Refactoring

09-Refactoring.py

Demo

Code review

The code in the following file implements a calculator program for rational numbers using most of the things we covered until now: **10-CalculatorProcedural.py**