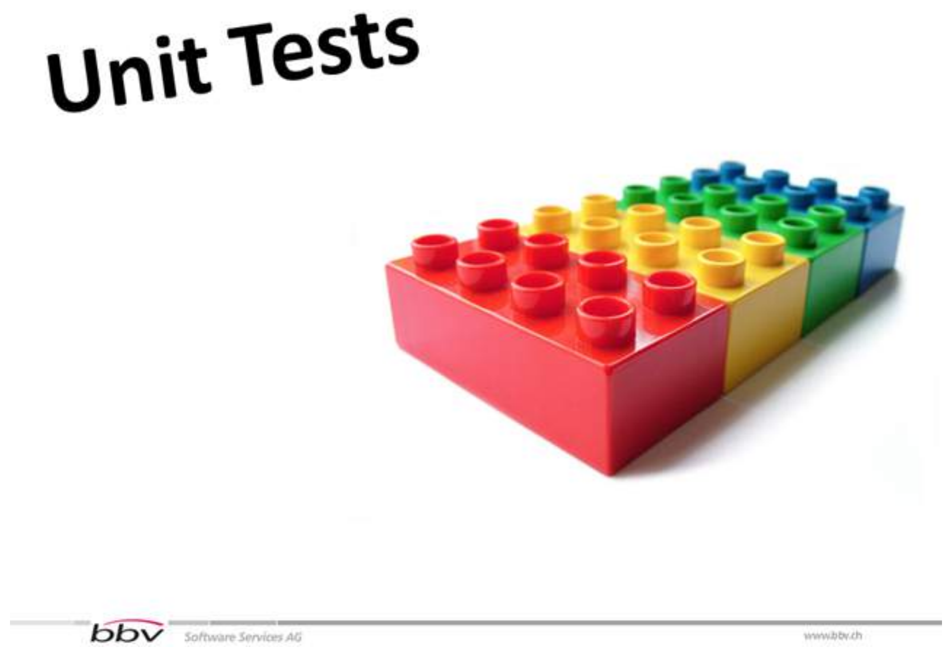
Unit Testing and Test-Driven Development

Unit Testing

What’s Unit Testing



Unit Tests

bbv Software Services AG

www.bbv.ch

Unit Testing

Unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use.

5 Benefits

1. Find Problems Early

2. Facilitates Changes

3. Simplifies Integration

4. Documentation

5. Design

Separation of interface from implementation

Separation of interface from implementation is when you’re doing a unit test but some classes may have references to others classes or some class that depend on database.

This is a mistake! because a unit test should usually not go outside of its own class boundary.

Crossing own unit boundaries turns unit test into integration test.

How to avoid Separation of interface from implementation

Create
MOCK OBJECT
or
FAKE !!

Unit test & Extreme Programming(XP)

Unit test & Extreme Programming(XP)

Unit testing is a cornerstone of extreme programming , which relies on an automated unit testing framework.

Limitations

- Testing will not catch every error in the program.
- it will not catch integration errors or broader system-level errors.
- This obviously takes time and its investment may not be worth the effort.

Tools

Java

- JUnit Test
- TestNG

Looking for other languages ?

http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#Java

Example

Simple Calculator

Test-Driven Development

What’s TDD ?

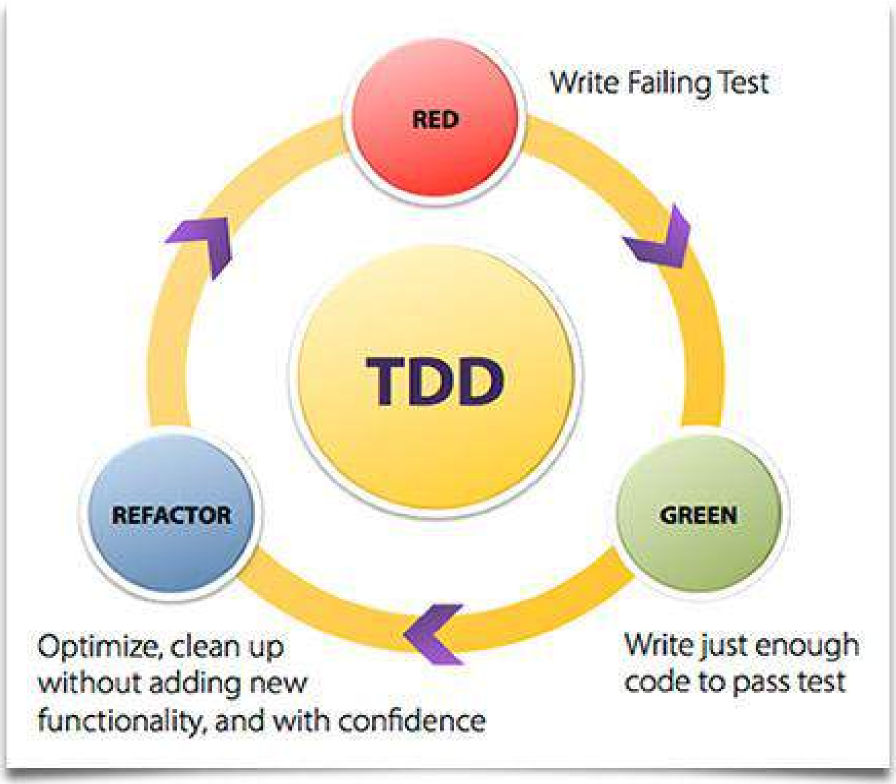
Test-Driven Development (TDD)

is a software development process that relies on the repetition of a very short development cycle.

What’s TDD ?

Test-Driven Development is related to the test-first programming concepts of extreme programming.

TDD Cycle


RED
Write Failing Test
TDD
GREEN
Write just enough code to pass test
REFACTOR
Optimize, clean up without adding new functionality, and with confidence

TDD Cycle

Red	Add a test and run, if the test fails go to green.
Green	Write production code and run all tests If all tests succeed, go to refactor.
Refactor	Optimize, Clean up code
Repeat	Repeat all steps with new tests.

3 Rules of TDD

1. Do not write any production code unless it is to make a failing unit test pass.
2. Do not write any more of unit test than is sufficient to fail; and compilation failures are failures.
3. Do not write any more production code than is sufficient to pass the one failing unit test.

Benefits

- Better understanding of what you're going to write.
- More modularized, flexible and extensible
- Enforces the policy of writing tests better.

Limitations

In situations where automated unit tests are not applicable, TDD obviously does not apply.

Test written is time consuming.

TDD is highly reliant on refactoring and programmer skills

Example

Guessing Game

How to play ?

- You can guess **only 7 times !**
- The secret number have 4 digits (with no repeated digits)
- **"A"** means you have guess the right digits and the right position
- **"B"** means you have guess the right digits but the position is wrong

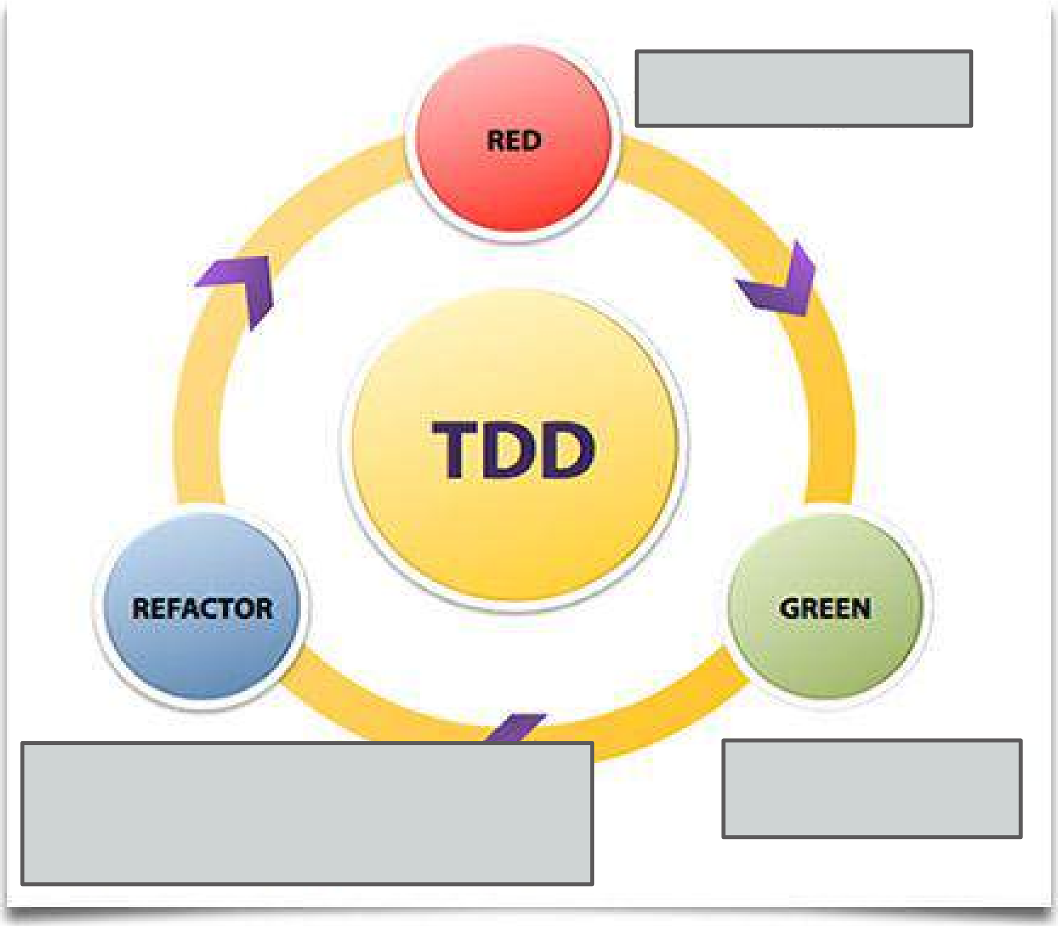
Guessing Game

Example

- Secret number is **“0314”**
- You guess a number **“0438”**
- Result -> A : 1, B : 2
    - In your guess number have 1 digit that contains in the secret number and it’s in the right place (“0”)
    - In your guess number have 2 digits that contains in the secret number but place in the wrong position (“4” and “3”)

Quiz & Exercise

```
 /\   /\
\(OwO!!)/
```


RED
TDD
REFACTOR
GREEN

Any Questions ?