Unit testing

Модульне тестування

Що таке тестування?

Тестування програмного забезпечення — техніка контролю якості, що перевіряє відповідність між реальною і очікуваною поведінкою програми завдяки кінцевому набору тестів.

Види тестування:

- Ручне тестування (manual testing)
- Автоматизоване тестування (automated testing)
- Напівавтоматизоване тестування (semiautomated testing)

- Компонентне (модульне) тестування (component/unit testing)
- Інтеграційне тестування (integration testing)
- Системне тестування (system/end-to-end testing)

Unit testing

Метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми.

Модулем називають найменшу частину програми, яка може бути протестованою.

У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно - орієнтованому програмуванні — інтерфейс, клас.

Мета модульного тестування -

ізолювати окремі частини програми і показати, що окремо ці частини працюють коректно.

Коли модульне тестування не ефективне?

- Складний код
- Точний результат не відомий
- Помилки інтеграції та продуктивності

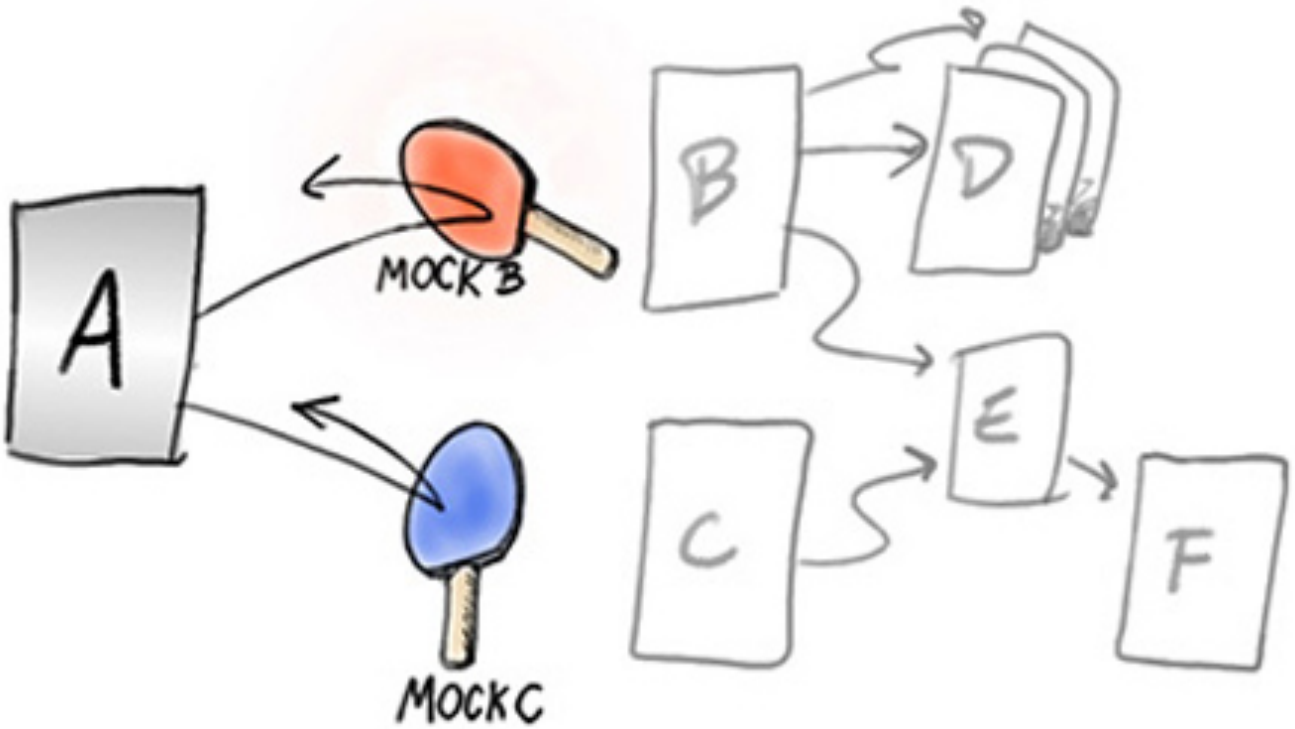
Переваги

- Легкий рефакторинг
- Спрощене інтеграційне тестування
- Документування
- Відокремлення інтерфейсу від реалізації

Mock

Заглушки - замінюють відсутні компоненти, які викликаються модулем і виконують такі дії:

- повертаються до елементу, не виконуючи ніяких інших дій;
- відображають повідомлення і іноді пропонують тестеру продовжити тестування;
- повертають постійне значення або пропонують тестеру самому ввести значення;
- здійснюють спрощену реалізацію відсутньої компоненти;
- імітують виняткові ситуації.


A
MOCK B
MOCK C
B
D
C
E
F

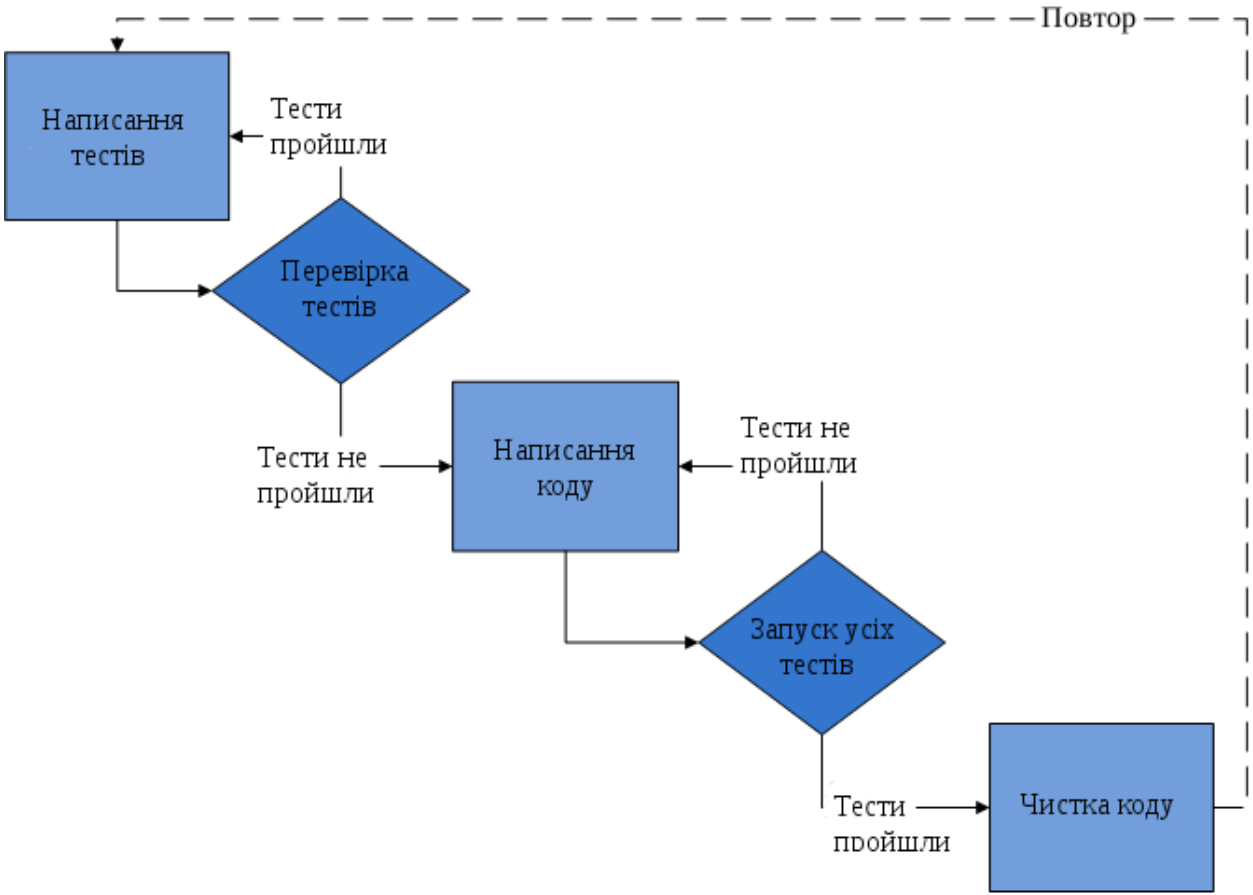
Розробка через тестування

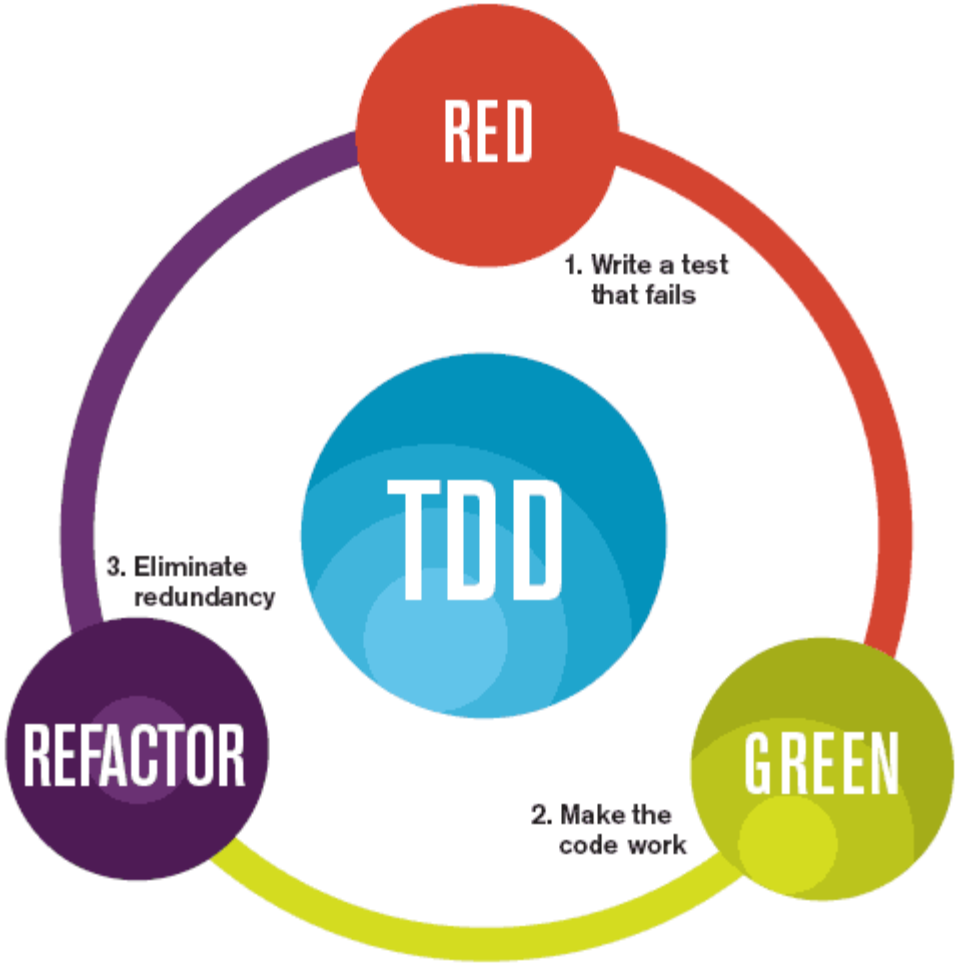
Test-driven development (TDD)

Техніка розробки програмного забезпечення, яка грунтується на повторенні дуже коротких циклів розробки:

- спочатку пишеться тест, що покриває бажану функціональність
- потім пишеться код, який дозволить пройти тест
- і під кінець проводиться рефакторинг нового коду до відповідних стандартів.

Графічна презентація циклу розробки





The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Behavior-driven development (BDD)

- відгалуження TDD
- зв’язок коду з вимогами до функціоналу
- фокус не на тестах, а на поведінці
- запис вимог за допомогою звичайних фраз

Інструменти модульного тестування

Для Java

- JUnit
- TestNG
- JavaTESK
- Spock

Для C

- CUnit
- CTESK
- cfix

Для JavaScript

- Mocha
- Chai ("assertion library")
- Karma (от создателей Angular.JS)
- QUnit (от создателей jQuery)
- JsUnit
- Jasmine

-

simple, flexible, fun

Mocha is a feature-rich JavaScript test framework running on **node.js** and the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on **GitHub**.

Features

- browser support
- simple async support, including promises
- test coverage reporting
- string diff support
- javascript API for running tests
- proper exit status for CI support etc
- auto-detects and disables coloring for non-ttys
- maps uncaught exceptions to the correct test case
- async test timeout support
- test-specific timeouts
- growl notification support
- reports test durations
- highlights slow tests
- file watcher support
- global variable leak detection
- optionally run tests that match a regexp
- auto-exit to prevent "hanging" with an active loop
- easily meta-generate suites & test-cases
- mocha.opts file support
- clickable suite titles to filter test execution
- node debugger support
- detects multiple calls to `done()`
- use any assertion library you want
- extensible reporting, bundled with 9+ reporters
- extensible test DSLs or "interfaces"
- before, after, before each, after each hooks
- arbitrary transpiler support (coffee-script etc)
- TextMate bundle
- and more!

1. 2. 3. Mocha!

```
$ npm install -g mocha
$ mkdir test
$ $EDITOR test/test.js

var assert = require("assert")
describe('Array', function(){
  describe('#indexOf()', function(){
    it('should return -1 when the value is not present', function(){
      assert.equal(-1, [1,2,3].indexOf(5));
      assert.equal(-1, [1,2,3].indexOf(0));
    })
  })
})

$  mocha

  .

  ✔ 1 test complete (1ms)
```

TDD

The "**TDD**" interface provides `suite()`, `test()`, `suiteSetup()`, `suiteTeardown()`, `setup()`, and `teardown()`.

```
suite('Array');

test('#length', function(){
  var arr = [1,2,3];
  ok(arr.length == 3);
});

test('#indexOf()', function(){
  var arr = [1,2,3];
  ok(arr.indexOf(1) == 0);
  ok(arr.indexOf(2) == 1);
  ok(arr.indexOf(3) == 2);
});
```

BDD

The "**BDD**" interface provides `describe()`, `it()`, `before()`, `after()`, `beforeEach()`, and `afterEach()`:

```
describe('Array', function(){
  before(function(){
    // ...
  });

  describe('#indexOf()', function(){
    it('should return -1 when not present', function(){
      [1,2,3].indexOf(4).should.equal(-1);
    });
  });
});
```

Hooks

Mocha provides the hooks `before()`, `after()`, `beforeEach()`, `afterEach()`, that can be used to set up preconditions and clean up your tests.

```
describe('hooks', function() {
  before(function() {
    // runs before all tests in this block
  })
  after(function(){
    // runs after all tests in this block
  })
  beforeEach(function(){
    // runs before each test in this block
  })
  afterEach(function(){
    // runs after each test in this block
  })
  // test cases
})
```

Mocha example

Jasmine

Powered by PIVOTAL LABS

Jump To: ajax.js boot.js custom_boot.js custom_equality.js custom_matcher.js custom_reporter.js focused_specs.js introduction.js node.js python_egg.py ruby_gem.rb upgrading.js

introduction.js

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests. This guide is running against Jasmine version 2.1.0.

Standalone Distribution

The releases page has links to download the standalone distribution, which contains everything you need to start running Jasmine. After downloading a particular version and unzipping, opening `SpecRunner.html` will run the included specs. You'll note that both the source files and their respective specs are linked in the `<head>` of the `SpecRunner.html`. To start using Jasmine, replace the source/spec files with your own.

Suites: `describe` Your Tests

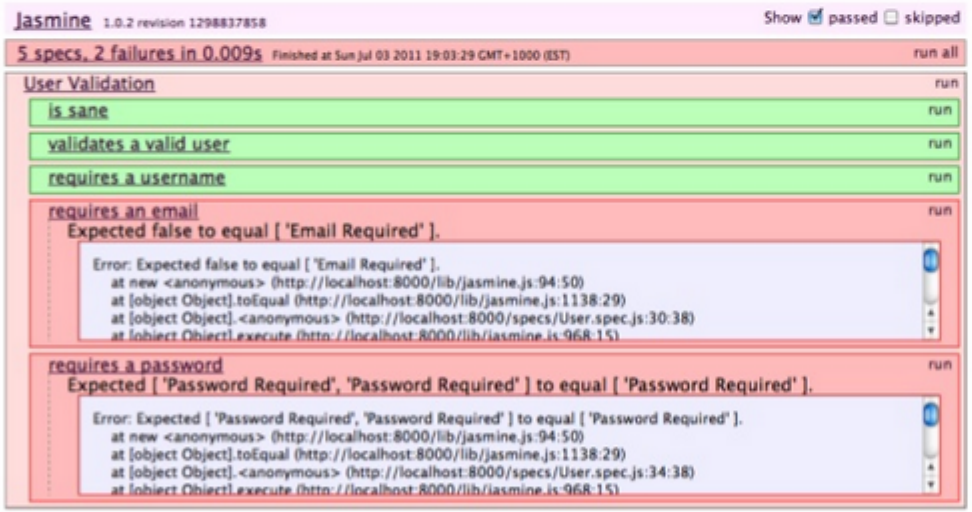
```
describe("A suite", function() {
  it("contains spec with an expectation", function() {
    expect(true).toBe(true);
  });
});
```

Running Jasmine Tests

```
<!DOCTYPE html>
<html>
<head>
  <title>Jasmine Test Runner</title>

  <!-- Jasmine Library -->
  <link rel="stylesheet" type="text/css" href="/lib/jasmine.css">
  <script type="text/javascript" src="/lib/jasmine.js"></script>
  <script type="text/javascript" src="/lib/jasmine.html.js"></script>

  <!-- App Dependencies -->
  <script src="/public/scripts/application.js" type="text/javascript" charset="utf-8"></script>
  <script src="/public/scripts/lib/jquery.js" type="text/javascript" charset="utf-8"></script>

  <!-- Specs -->
  <script src="/specs/User.spec.js" type="text/javascript" charset="utf-8"></script>

</head>
<body>

  <script type="text/javascript">
    jasmine.getEnv().addReporter(new jasmine.TrivialReporter());
    jasmine.getEnv().execute();
  </script>
</body>
</html>
```

Running Jasmine Tests

Use a simple http server to serve the content
eg python -m SimpleHTTPServer
Point browser at jasmine location




THANKS FOR YOUR ATTENTION
NO QUESTIONS, PLEASE
memegenerator.net