Test Automation

libs, tips and tricks for testing your code

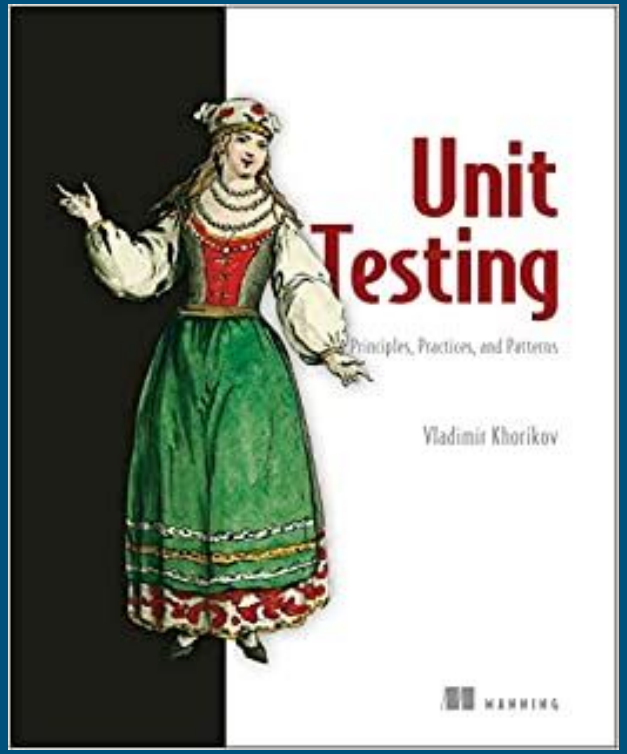
Outline

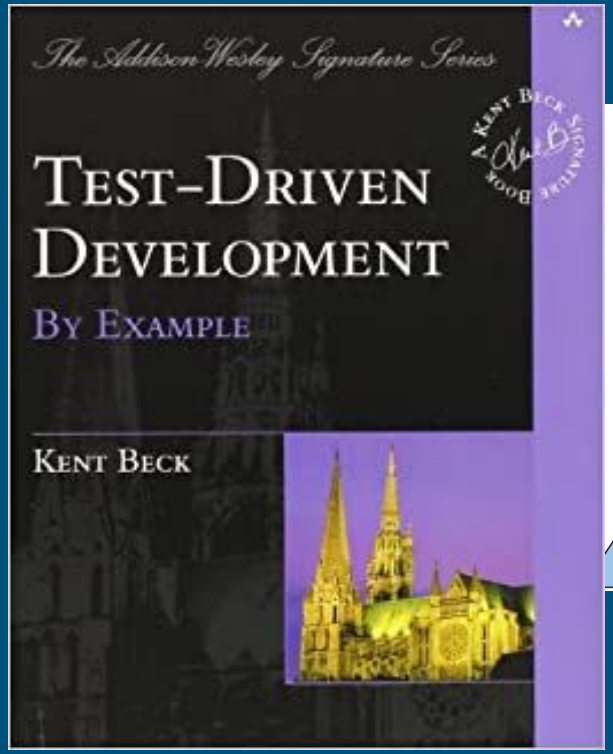
- Why testing
- Testing Pattern and Methods
  - Test Double
  - TDD, BDD, TCR
  - Snapshot testing
  - Smoke Test
  - Fuzzy Test
  - Koans
  - CI
- Testing Libs
  - Functions
  - Classes
  - Spec
  - Sentece Style
  - Test Double
  - Coverage
- Ide Tools
- IJava Jupyter Notebook
- Code Time

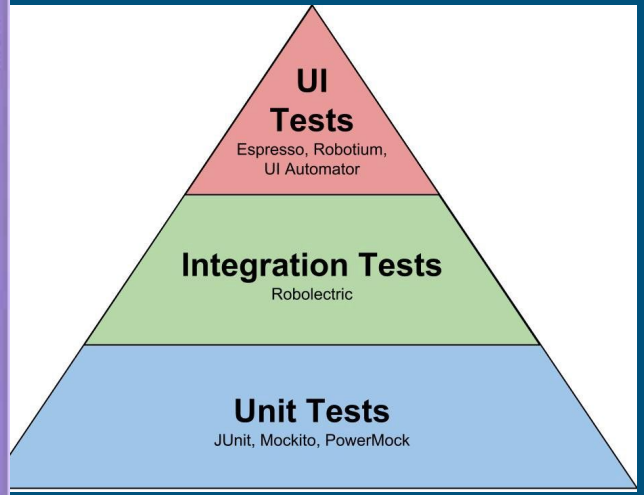
Why testing

- time preserving vs time consuming
- knowledge save
- the testing code is not bored to test again the same functionalities each times
- the testing code do not forget features
- the testing code remember what it should do

Testing Pattern and Methods







Test Double (always called Mock...)

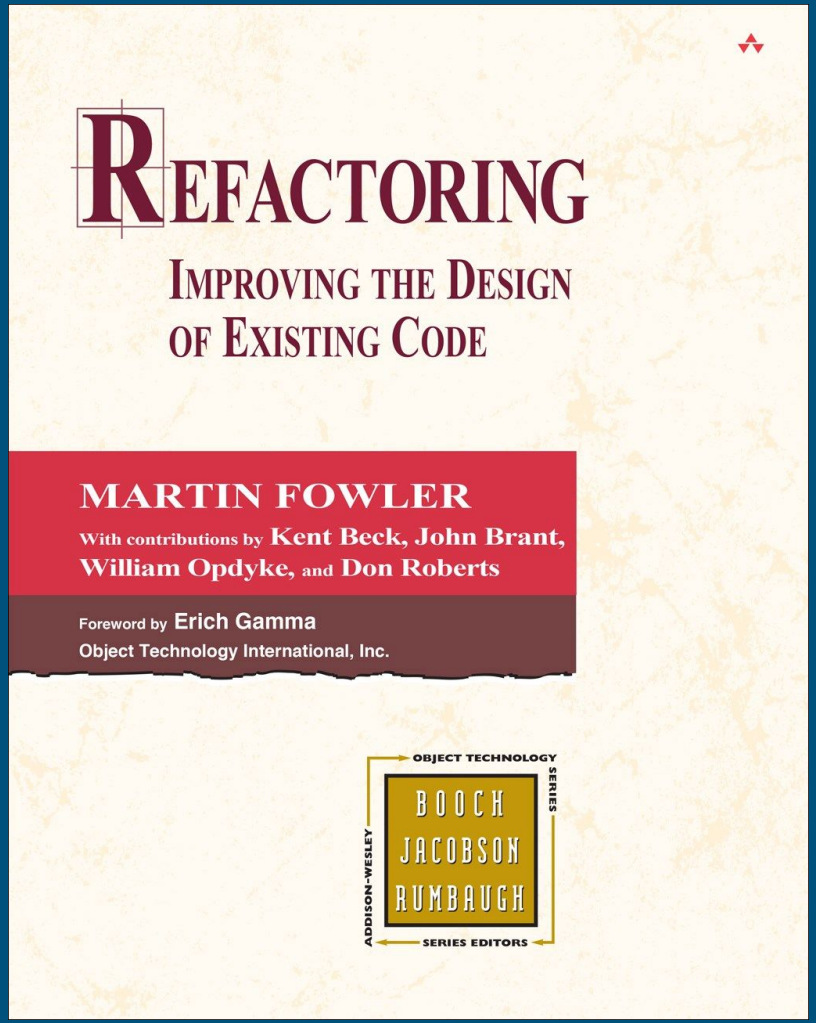
- object oriented
- the test double is an object with the same interface of another but with simplified behaviour
- dependency injection is welcome
- Types:
  - Dummy: no behaviour
  - Stub: fixed behaviour (without computation)
  - Spy: it memorize the call to it so you can assert the used params or what method it is called
  - Mock: A specific version of Spy, in which the spy assert is explicitly defined
  - Fake: simulate the original behaviour

Test Double

- advantages:
  - quick
  - test something that use components which not exist
  - remove external dependencies
- disadvantages:
  - possible errors
  - no check that the fake is real similar to the original

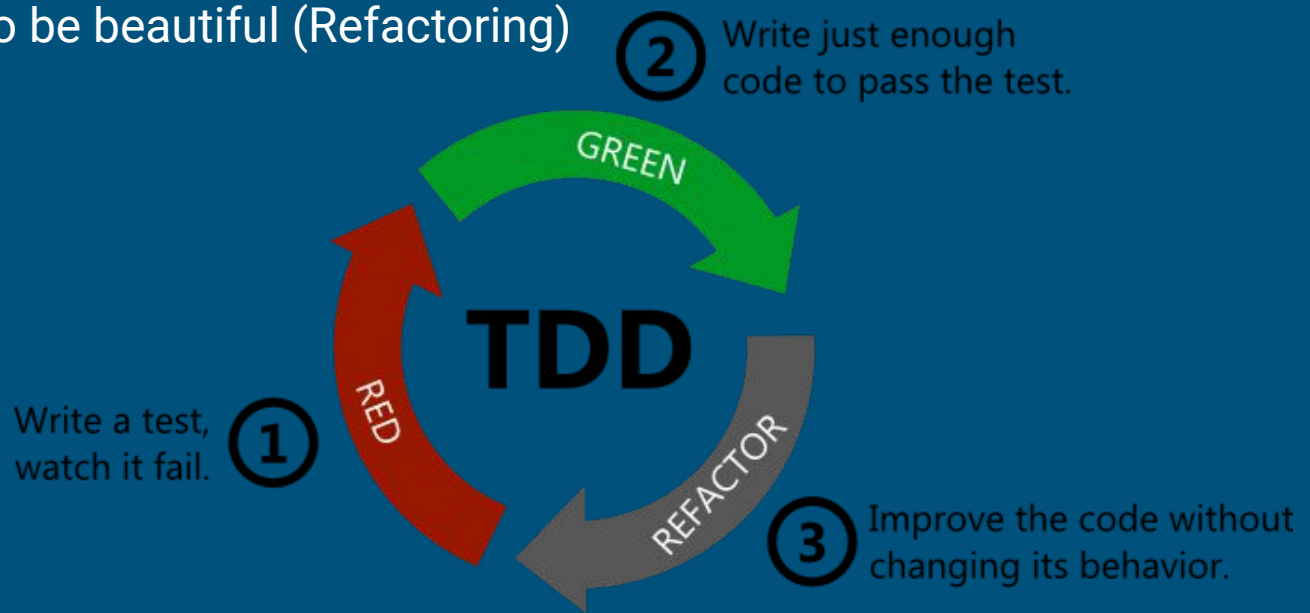
Refactoring

- refactoring the code is important also without test
- is the process of restructuring existing



Test Driven Development (TDD)

- write unit test before the code (Red phase)
- code the smallest to pass the dest (Green phase)
- change your code to be beautiful (Refactoring)



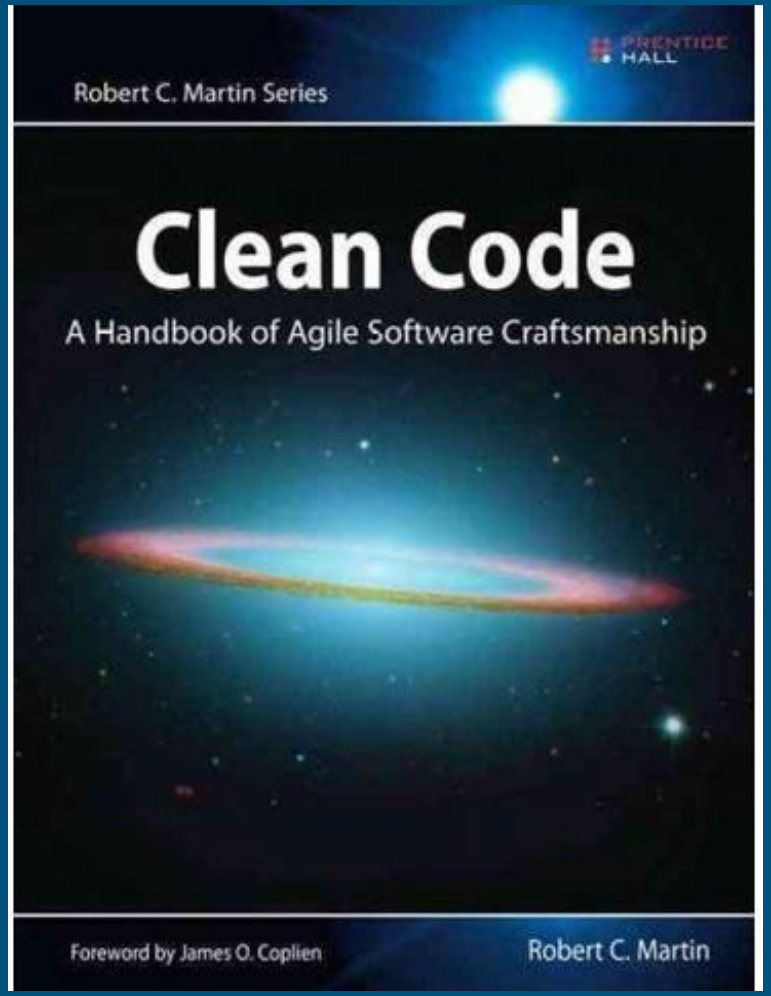
TDD Example

- Test1:
  res = 5 + 5
  assert 10 == res
- Code:
  sum(a, b): return 10

- Test1:
  res = 5 + 5
  assert 10 == res
- Test2:
  res = 2 + 2
  assert 4 == res
- Code:
  sum(a, b): return a + b

Clean Code

- easy to understand
- easy to change
- you can test it
- it is possible to reach the structure of famous pattern iterating with refactoring and testing

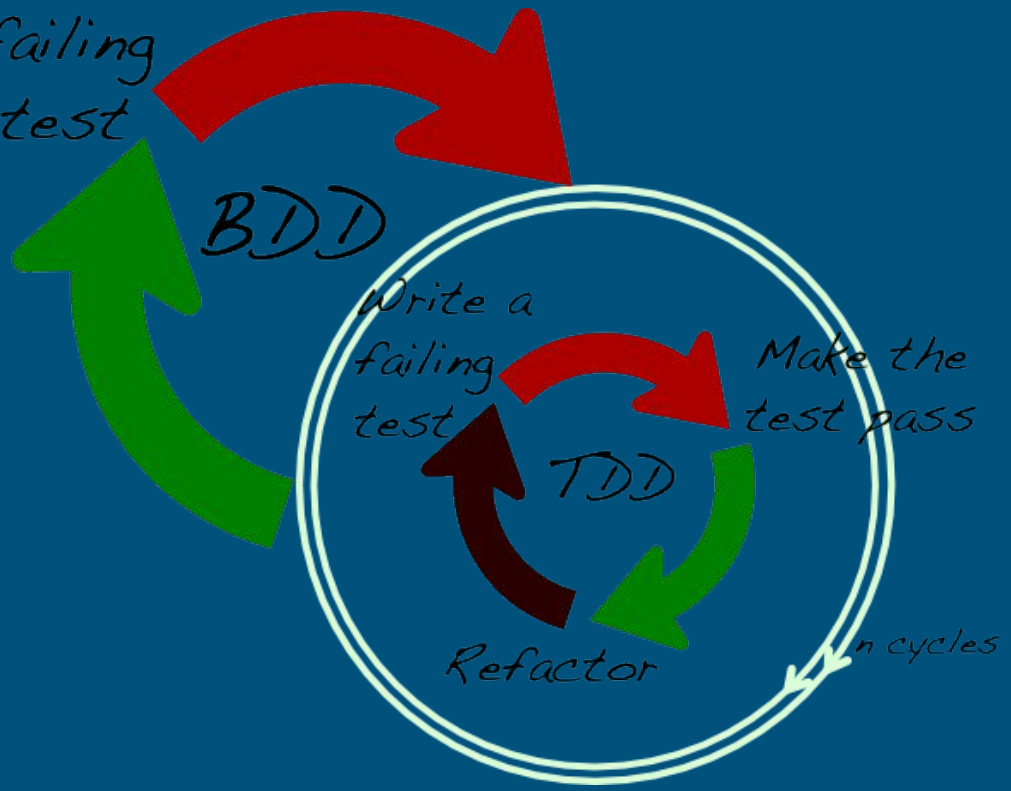


Kata

- a dummy exercise to practice TDD
- https://kata-log.rocks/tdd
- SOLID : important object oriented principles that can be learned by Kata
- GRASP
- ...

Behaviour Driven Development (BDD)

- TDD but not only unit test
- feature, ui, acceptance test in the red phase

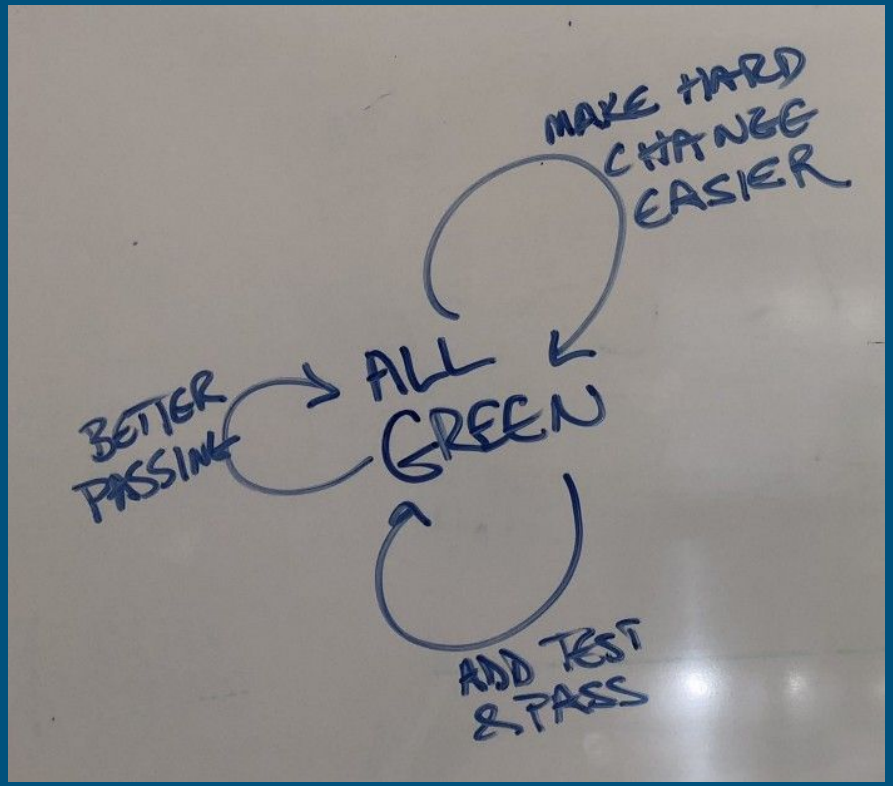


Snapshot testing

- UI testing method
- Assert that the actual view is equal to the old screen shot
- This can be a snapshot with the body representation of the UI but usually it is a .jpg
- Used when you rewrite an application or when you port to another system
- The test is hard to maintain

Test && Commit || Revert (TCR)

- integrate git as practice
- you commit every time the tests passes
- if you are for much time in red bar you can revert and restart



Smoke Test

- test pre-deploy
- test post-deploy
- production environment
- Es. 1 for a server test if the DB is reachable before deploy, or test if server is alive after deploy
- Es. 2 for a desktop application test that the binary is running

Fuzzy Test

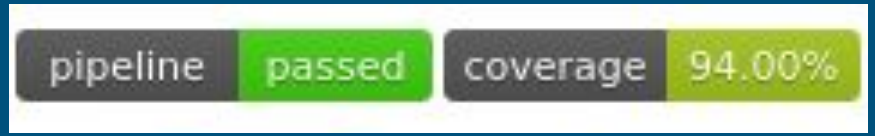
- production or near production environment
- we try randomly to many possible operations
- typically to find bugs into a non tested application
- sometimes also the assert is generated
- often we only now if the app crash and the funnel to reproduce it

Koans

- A test suite with failing test
- You understand a sintax behaviour of the language if you can become green a test by correcting the assertion
- Koans for languages:
  - Java
  - Python
  - Ruby

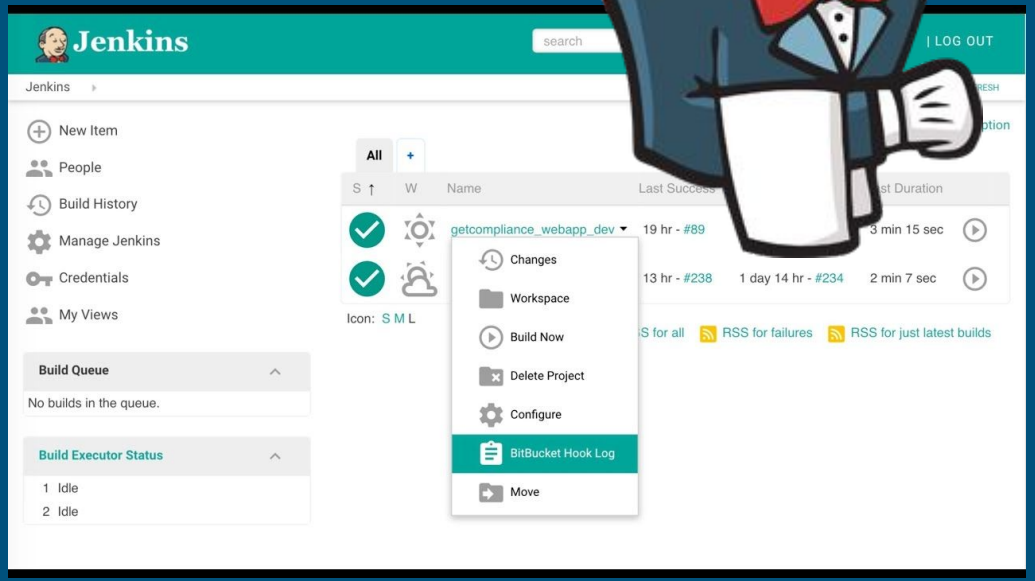
Continous Integration (CI)

- Integration server: run the test (each time you push in all branches or in particular branches)
- You have an interface in which you can see if the test passes and also the coverage
- (Best Practice env: dev, integration, test, prod)
- Tools
  - Classical Server: Jenkins, TeamCity
  - Cloud: GitlabCi/CD, TravisCI, CircleCI



Jenkins

- You must add your codebase and the command to run by a gui, not in the code
- you can self host
- you can add plugins



Gitlab CI/CD

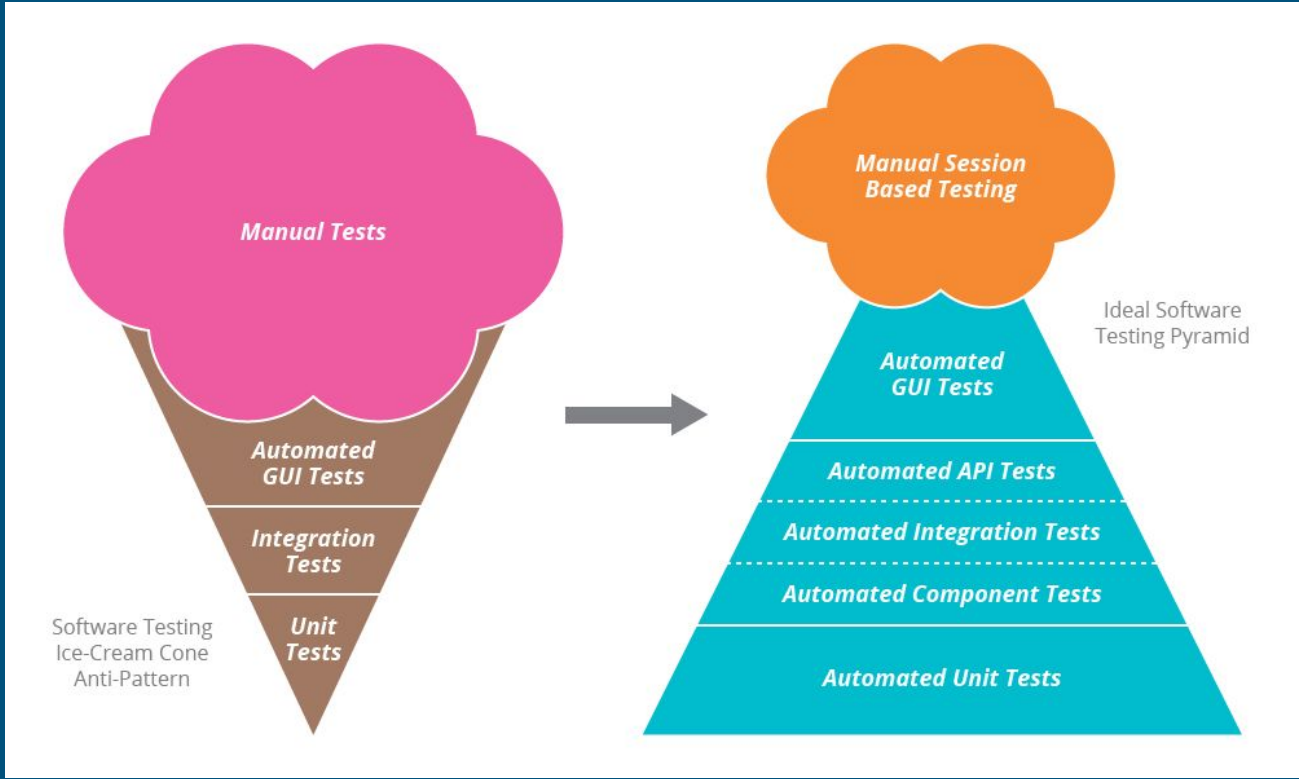
- .gitlab-ci.yml code into the codebase
- testrunner into your machine or gitlab servers

.gitlab-ci.yml 304 Bytes

Edit | Web IDE | Pipeline Editor

```yaml
python_test:
  image: registry.gitlab.com/artelabsuper/raccomandazioni-uninsubria/test_python
  services:
    - docker:dind
  script:
    - pip install --upgrade pip
    - pip install -r requirements.txt
    - coverage run --source=STERbackend/ STERbackend/manage.py test api.tests
    - coverage report
```

Testing Libs & Styles



Functions

- python: pytest
- other languages generally do not have an alternative for that style

```python
client = TestClient(app)


def test_get_root():
    response = client.get("/")
    assert response.status_code == 200
    assert response.json() == {"msg": "Hello World"}
```

Classes

- java: JUnit
- python: unittest
- Ruby: Minitest

```python
class TestApp(unittest.TestCase):

    def setUp(self):
        self.client = TestClient(app)

    def test_get_root(self):
        response = self.client.get("/")

        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.json(), {"msg": "Hello World"})
```

Doc test

- Java: doctest
- Python: doctest

```python
@app.get("/")
async def root():
    """
    Return a message and 200 with doctest
    >>> from fastapi.testclient import TestClient
    >>> client = TestClient(app)
    >>> response = client.get("/")
    >>> response.status_code
    200
    >>> response.json()
    {'msg': 'Hello World'}
    """
    return {"msg": "Hello World"}
```

Spec

- Java: javaspec
- Python: Mamba
- Ruby: Rspec

Spec Guidelines:
https://www.betterspecs.org/

```
with description('FastApi Server root') as self:
  with before.all:
    self.client = TestClient(app)

  with it('should return 200 and message'):
    response = self.client.get("/")

    expect(response.status_code).to(equal(200))
    expect(response.json()).to(equal({"msg": "Hello World"}))
```

Sentence Style

- Java: Cucumber
- Python: robotframework
- Ruby: Calabash (for android)

```
*** Test Cases ***
Valid Login
    Open Login Page
    Input Credentials    demo    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

Test Double

- Java: Mockito, Power Mock
- Python: mock
- Ruby: Rspec - Test Doubles

Test Coverage

- Java: JaCoCo , Cobertura, JCov
- Python: coverage.py
- Ruby: simplecov

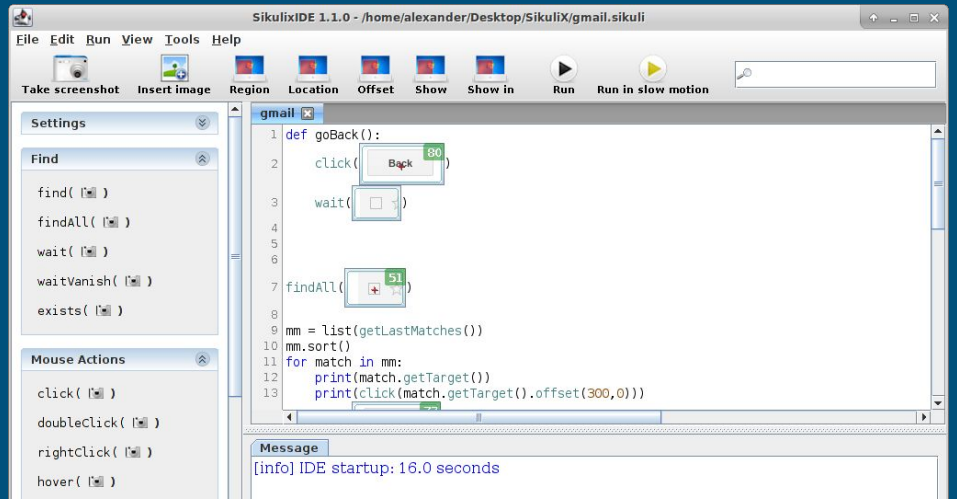
Infrastructure Test

- Test provision script
- Test deploy script
- Test infrastructure creation
- IaC (Infrastructure as Code)
- Virtual Machines
- Containers

- Resources:
  - DevOps Playbook. Testing con Molecule
  - Ansible
  - Goss
  - Terraform
  - Molecule
  - Docker
  - Vagrant

Ide & Tools

- Generally the testing is supported by the ide like Intellij
- sometimes they have normal main so it is supported like a normal run
- They do not have a special ide but SiculiX is an exception



Jupyter and IJava Kernel

- Jupyter notebook
- Ijava
- Into the notebook we can put together code and text (markdown)
- Interactive run
- We can interactive run the code on the fly

Factorial Example

This is a factorial implementation.

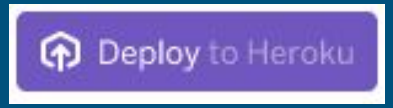
```
public static int factorial( int n ){
        int f = 1;

        for(int i = 1; i <=n; i++){
            f = f*i;
        }
        return f;
}
```

```
factorial(2)
```

```
2
```

IJava notebook Deploy

- Login on heroku: https:/heroku.com
- jupyter with java: https://github.com/nicolalandro/java_junit_jupyter
- Click on the deploy button on the repo
- Add your password
- Wait some minutes…



Markdown

- Github README.md SIntax
- You can use it into the notebook
- # Heading Level 1, ## Heading Level 2…
- * bullet point
- - [ ] unchecked check box, - [x] checked checkbox
- `code inline`
- ```code box, you can use newline```
- latex math $eq...$
- more sintax here: https://www.markdownguide.org/basic-syntax

Code time

Demo: https://gitlab.com/nicolalandro/demp_java_test

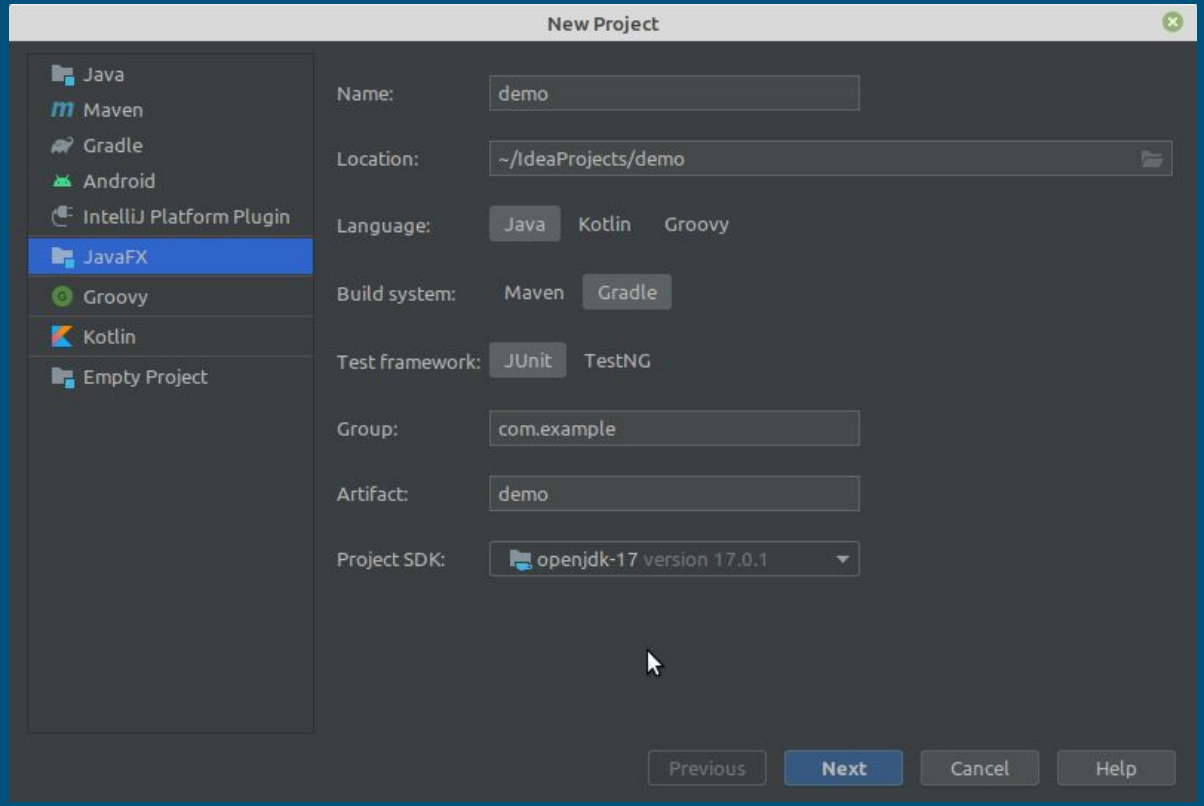
- Java11: programming language
  (retro-compatibility to Java 8, LTS deadline March 2022)
  (remember that new java 17 should be LTS)
- Gradle: dependencies and task manager
- Intellij Idea Community: IDE (Integrated Development Environment)
- git and Gitlab.com: version control system
- Gitlab CI/CD: CI server

Demo References

- Intellij Idea Community
- Gradle for Java
- Jacoco+Gradle
- JavaFX: code examples and slides
- Gluon Scene Builder
- TestFX

Create a Project

- click on file>new>project
- select JavaFX
- select Name, Location, java version
- select Gradle as build system
- Click Next and Finish



Edit the Java version compatibility

- Open build.gradle
- select the source and target compatibility options:

```
sourceCompatibility = '8'
targetCompatibility = '11'
```

Add junit Libs

- Open build.gradle
- Add under dependecies
- create folder src/test/java/<package>
- put your classes in that folder

```
dependencies {
    testImplementation("org.junit.jupiter:junit-jupiter:5.7.1")
    testImplementation("org.junit.jupiter:junit-jupiter-api:5.7.1")
    testImplementation("org.junit.jupiter:junit-jupiter-engine:5.7.1")

    testImplementation("org.testfx:testfx-junit5:4.0.16-alpha")
    testImplementation("org.hamcrest:hamcrest-core:2.1")
    testImplementation("org.hamcrest:hamcrest:2.1")
    testImplementation("org.mockito:mockito-junit-jupiter:4.0.0")

    testImplementation("org.testfx:openjfx-monocle:jdk-11+26")
}
```

Add Jacoco Config 1

- Official Guide: Jacoco+Gradle
- Open build.gradle
- into the plugins add jacoco id

```
plugins{
    ...
    id 'jacoco'
    id 'org.barfuin.gradle.jacocolog' version '2.0.0'
}
```

Add Jacoco Config 2

- change test
```
test {
    useJUnitPlatform()
    finalizedBy jacocoTestReport
}
```
- create the task jacocoTestReport
```
jacocoTestReport {
    dependsOn test
    reports {
        xml.required = false
        csv.required = true
        html.outputLocation = layout.buildDirectory.dir('jacocoHtml')
    }
}
```

Test FX Without Real Screen

- add openfx-monocle to the libraries
- set java env:
  _JAVA_OPTIONS="-Dheadless.geometry=1600x1200-32 -Djava.awt.headless=true -Dtestfx.robot=glass -Dtestfx.headless=true -Dprism.order=sw -Dprism.verbose=true -Dglass.platform=Monocle -Dmonocle.platform=Headless"
- use xvbf into the container
- for gradle container remember to install also libpangoft
- info can be finded at the end of the TestFX Readme and StackOverflow

Gitlab CI/CD

```
1   java_test:
2     image: gradle:jdk11
3     script:
4       - apt-get update && apt-get install -y xvfb libpangoft2-1.0-0
5       - gradle wrapper
6       - export _JAVA_OPTIONS="-Dheadless.geometry=1600x1200-32 -Djav
7       - Xvfb :99 &>/dev/null &
8       - export DISPLAY=:99
9       - ./gradlew test
10      # - ./gradlew test --tests com.example.demo_java_test.StringFc
11    coverage: /Instruction\sCoverage:\s\d+.\d+%/
```

Useful Links

- docs
  - JUnit asserction
  - Mockyto Any
- Exercise
  - jupyter basics
  - Junit basic
  - Mockyto
  - Exercises

```
# Esempio
In questo esempio non dico niente
```
codice
```
*corsivo* **grassetto** _sottolineato_

* una
* serie
* di
* punti


* [ ] uncheck
* [x] check
```

```
In [5]: public static void f(String s){
            System.out.println("ciao " + s);
        }

In [7]: f("xs")

        ciao xs
```