

Test Driven Development

La prospérité des charlatans

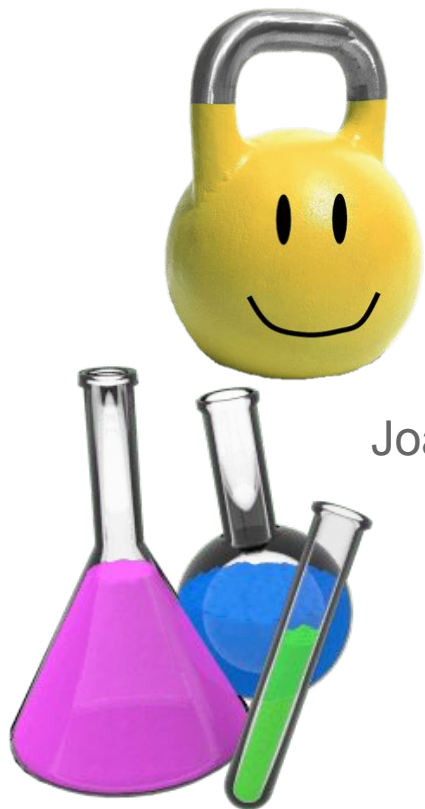
風

Par

Xiang LI <xiangfr007@gmail.com>

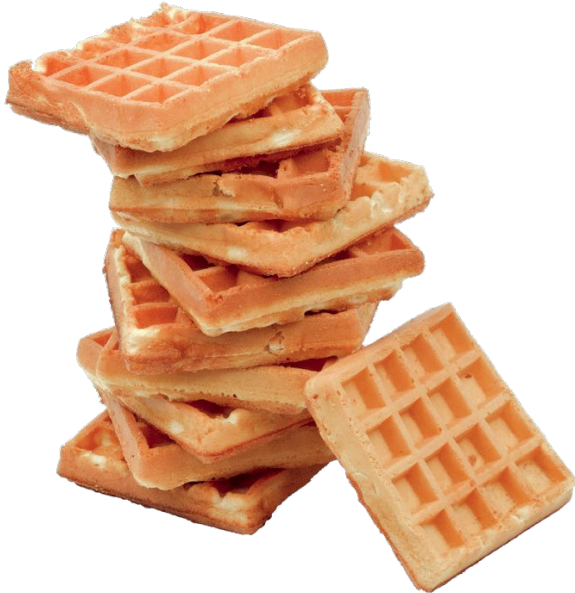
Jérôme SKODA <contact@jeromeskoda.fr>

Joaquim LEFRANC <lefrancjoaquim@gmail.com>

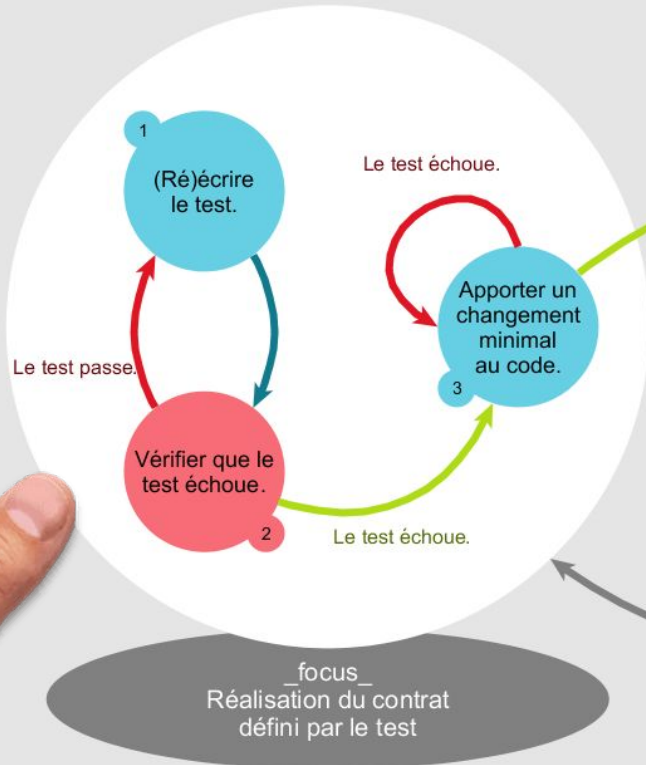


C'est quoi?

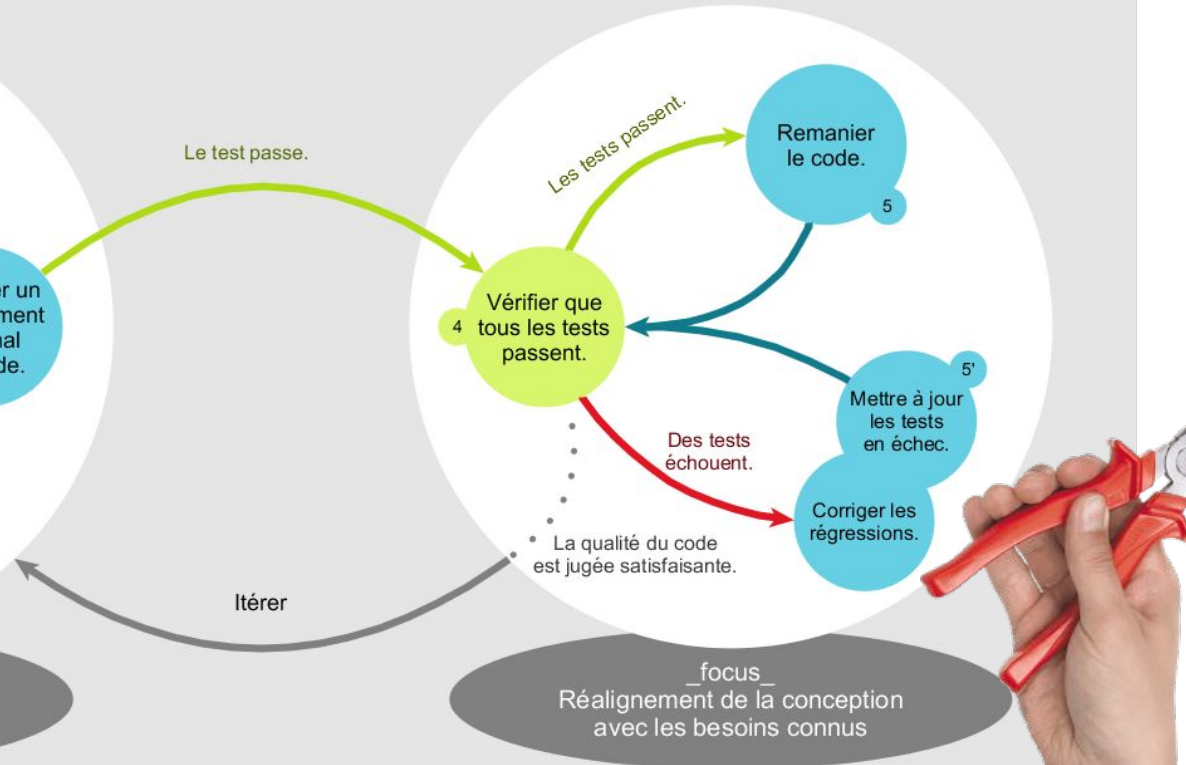
- *Développement itératif*
- *Validation par les test unitaires*



TEST-FIRST DEVELOPMENT



REFACTORING



Exemple simple



Problème: on veut une fonction **formatName(name)** qui nous permet de formaliser les noms.

Par exemple:

xiang li

>> Xiang LI

jérômE skoDa

>> Jérôme SKODA

jOAquIM leFranc

>> Joaquim LEFRANC



Exemple simple

1. Ecrire un premier test
2. Vérifier qu'il échoue (car le code qu'il teste n'existe pas), afin de vérifier que le test est valide ;



```
contraintes.org x t1.py x
1  #!/usr/bin/python3
2
3  from format import *
4
5  test_list = ["xiang li","jérômE skoDa","j0AquIM leFranc"]
6  expected = ["Xiang LI","Jérôme SKODA","Joaquim LEFRANC"]
7
8
9  def test():
10     flag = True
11     for i in range(0,len(test_list)):
12         result = formatName(test_list[i])
13         if result == expected[i]:
14             print("test["+str(i)+"] :"+test_list[i])
15             print("PASS")
16         else:
17             flag = False
18             print("test["+str(i)+"] :"+test_list[i])
19             print("FAIL "+expected[i]+" but: "+result)
20     print(" "*20)
21     if flag:
22         print("RESULT: ALL PASS")
23     else:
24         print("RESULT: FAIL :( :( :(")
25
26 test()
```

```
File Edit View Search Terminal Help
xiangfr007@u17:TDD$ ./t1.py
test[0] :xiang li
FAIL expected: Xiang LI but:
test[1] :jérômE skoDa
FAIL expected: Jérôme SKODA but:
test[2] :j0AquIM leFranc
FAIL expected: Joaquim LEFRANC but:
*****
RESULT: FAIL :( :( :(
xiangfr007@u17:TDD$
```

```
~/Desktop/pcomp-2018/groupe/la_prospérité_des_charlatans/TDD/format
File Edit Selection Find View Goto Tools Project Preferences Help
format.py x
1  #!/usr/bin/python3
2
3  def formatName(name):
4      return ""
```


Exemple simple

3. Ecrire juste le code suffisant pour passer le test

4. Vérifier que le test pass

5. Réusiner le code en gardant les mêmes fonctionnalités



```
format.py x
1  #! /usr/bin/python3
2
3  def formatName(name):
4      nameArray = name.split()
5      return nameArray[0].capitalize()+" "

xiangfr007@u17: TDD

File Edit View Search Terminal Help
xiangfr007@u17:TDD$ ./t1.py
test[0] :xiang li
PASS
test[1] :jérômE skoDa
PASS
test[2] :j0AquIM leFranc
PASS
*****
RESULT: ALL PASS
xiangfr007@u17:TDD$
```

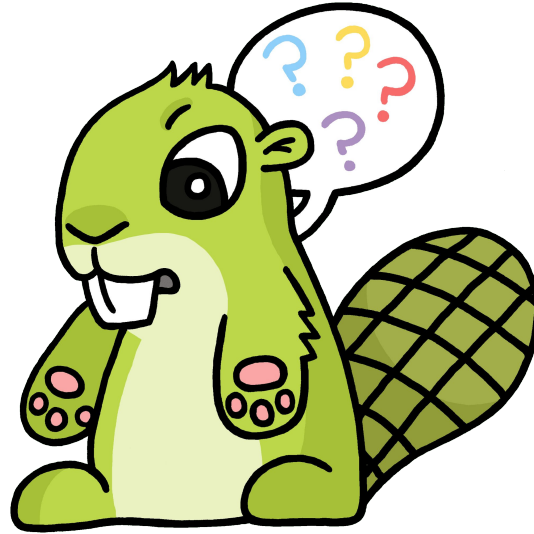
Avantages

- **Les tests unitaires sont réellement écrits**
- **Clarification des détails de l'interface et du comportement**
- **Vérification démontrable, répétable et automatisé**
- **Non présence de régression**

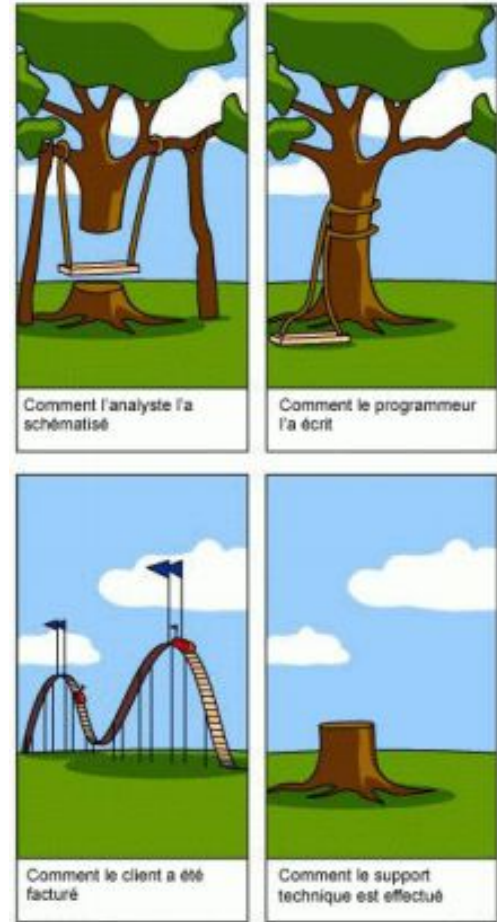


Inconvénients

- Développement de code en avance, inefficacité et trop de codes
- Risque de “Coding yourself into a corner”



Powered by Fuchs Template



Les outils

- Travis CI
- Jenkins / Hudson
- SonarQube
- Tinderbox
- Apache Continuum
- Team Foundation Server
- Chaos Monkey



Conclusion

La méthode TDD apporte de la robustesse au développement. Le code et le projet en général sont à tout instant fonctionnels. Ils peuvent faire l'objet de démonstrations avec un nombre de bugs réduits et surtout sans code dans un état intermédiaire.



