Test Driven Development

La prospérité des charlatans

風

Par

Xiang LI <xiangfr007@gmail.com>

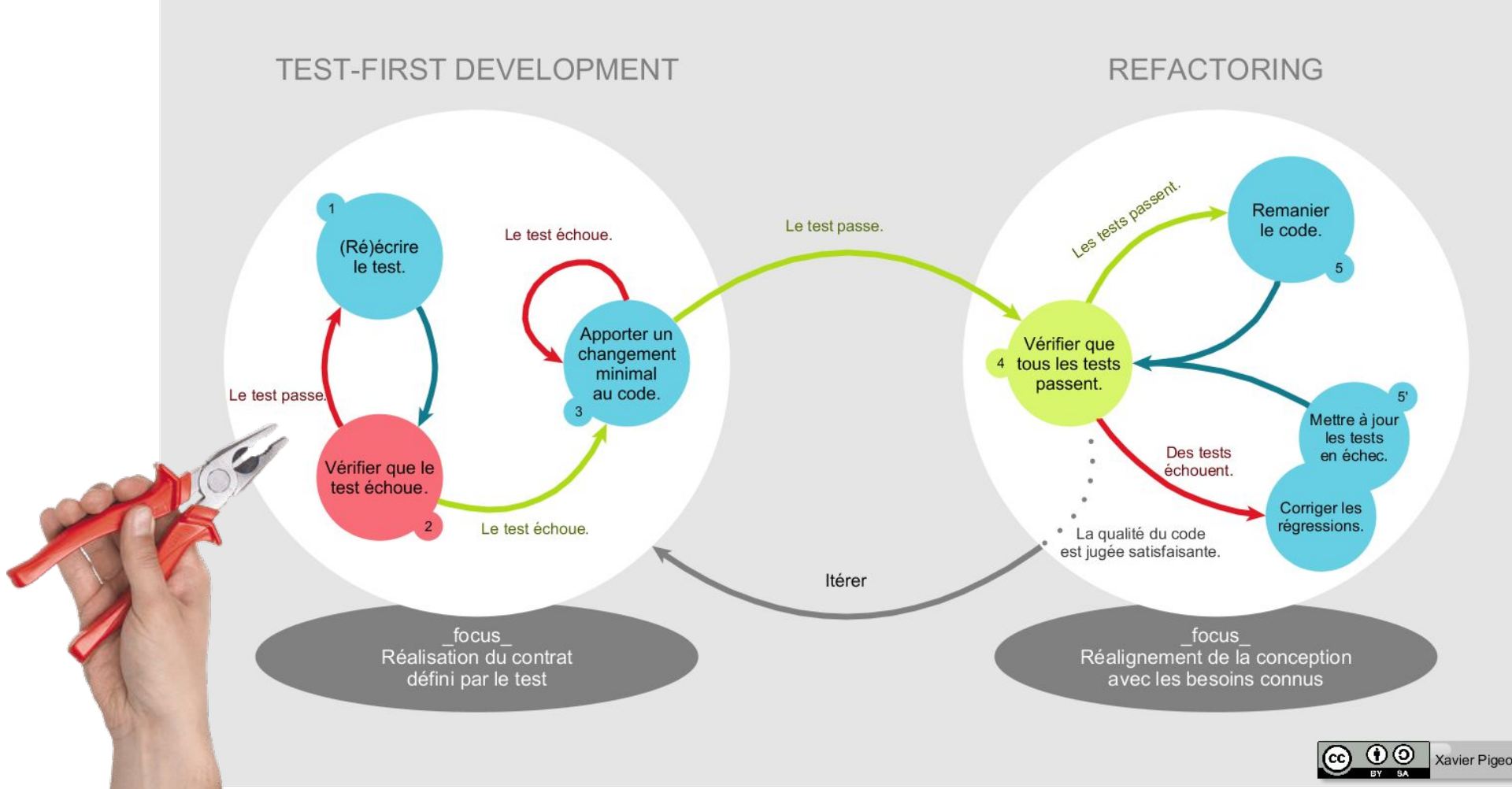
Jérôme SKODA <contact@jeromeskoda.fr>

Joaquim LEFRANC <lefrancjoaquim@gmail.com>

Powered by Fusch Template

C’est quoi?

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



Powered by Fusch Template

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   t1.py
#! /usr/bin/python3

from format import *

test_list = ["xiang li","jérômE skoDa","jOAquIM leFranc"]
expected = ["Xiang LI","Jérôme SKODA","Joaquim LEFRANC"]


def test():
    flag = True
    for i in range(0,len(test_list)):
        result = formatName(test_list[i])
        if result == expected[i]:
            print("test["+str(i)+"] :"+test_list[i])
            print("PASS")
        else:
            flag = False
            print("test["+str(i)+"] :"+test_list[i])
            print("FAIL "+"expected: "+expected[i]+" but: "+result)
    print("*"*20)
    if flag:
        print("RESULT: ALL PASS")
    else:
        print("RESULT: FAIL :( :( :(")

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] :jOAquIM leFranc
FAIL expected: Joaquim LEFRANC but:
********************
RESULT: FAIL :( :( :(
xiangfr007@u17:TDD$
```

```
~/Desktop/pcomp-2018/groupes/la_prosperite_des_charlatans/TDD/format
File  Edit  Selection  Find  View  Goto  Tools  Project  Preferences  Help
format.py
#! /usr/bin/python3

def formatName(name):
    return ""
```

Powered by Fusch Template

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

```
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

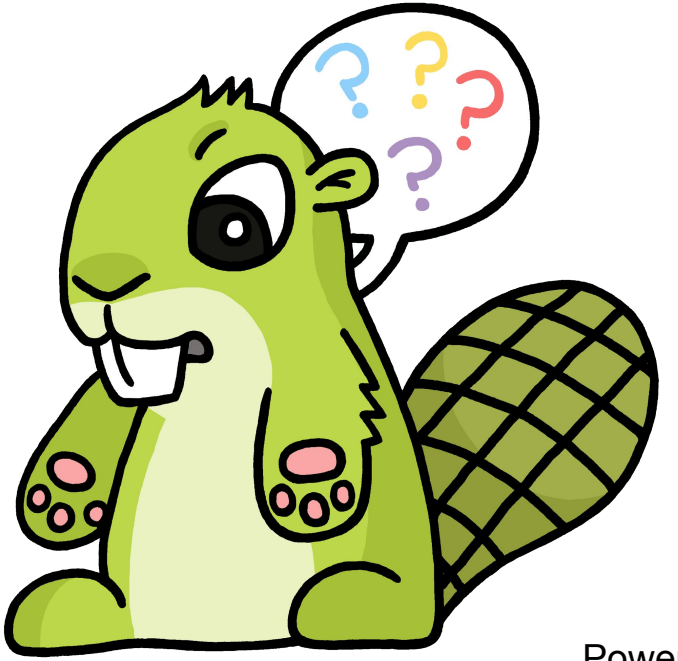
```
xiangfr007@u17:TDD$ ./t1.py
test[0] :xiang li
PASS
test[1] :jérômE skoDa
PASS
test[2] :jOAquIM leFranc
PASS
********************
RESULT: ALL PASS
xiangfr007@u17:TDD$
```

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”**



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.