Tests Logiciels - Master MS2D

Christophe Brun

Campus Saint-Michel IT

21 septembre 2023

Table des matières

1. Table des matières
2. Programme du module
3. Généralités
4. Tests Unitaires
5. Le mutation testing
6. Le BDD
7. Le TDD
8. Tests assistés par l'IA
9. Les tests d'intégration
10. Intégration Continue
    - Généralité
    - GitHub Action
    - Jenkins
    - GitLab CI
    - Conclusion
11. Stubbing et Mocking
    - Mocking
    - Stubbing
12. Conclusion
13. Licence CC

Tests Logiciels

Compétences acquises aucours des 3 jours du module

- Savoir développer et mettre en place des tests unitaires sur une application.
- Savoir développer et mettre en place des tests d'intégration sur une application.
- Savoir configurer un pipeline d'intégration continue sur une application.

Tests Logiciels

Le programme des 3 jours du module

Programme :

1. Les tests unitaires
   - Qu'est-ce qu'un test ?
   - Les tests unitaires
   - Les bonnes pratiques
   - Le Test Driven Development
   - Le code coverage
   - Tests avec langage naturel
2. Les tests d'intégration
   - Les tests d'intégration
   - Le fonctionnement
   - Les bonnes pratiques
3. L'intégration continue
   - Mise en place d'une plateforme d'intégration continue (GitLab CI)
   - Conteneurisation d'une application (API)
   - Configuration d'un pipeline de tests

Intervenant sur le module Tests Logiciels

Christophe Brun, CTO d'In France

- 1ere année d'intervenant à Saint-Michel 🤩.
- 7 ans de conseil en développement au sein d'SSII .
- 6 ans de conseil en développement à mon compte PapIT.
- Directeur technique et associé d'In France.
- Passionné !

L'histoire du testing

- ISTQB, l'International Software Testing Qualifications Board a été fondée en 1998[1]
- Dans son livre Extreme Programming Explained, Kent Beck parle de Test-first Programming en 1999
- Les standards modernes datent de la fin des années 90, début des années 2000
- JUnit, le framework de test le plus utilisé en Java date de 2002[2]
- Début du BDD (Behavior-Driven Development) avec JBehave, censé remplacer JUnit en utilisant le behavior au lieu de test, date de 2003[3]

[1] ISTQB, About Us, https://www.istqb.org/about-us/who-we-are

[2] Steven J Zeil, Unit Testing Frameworks, https://www.cs.odu.edu/~zeil/cs350/latest/Public/junit/index.html

[3] Cucumber, https://cucumber.io/docs/bdd/history/

Qu'est-ce qu'un test ?

Définition de l'International Software Testing Qualifications Board et IBM

L'ISTQB définit les termes suivants dans son glossaire[4] :
Test : Un ensemble d'un ou plusieurs cas de tests.
Cas de test : Un ensemble de conditions préalables, de données d'entrée, d'actions (le cas échéant), de résultats attendus et de postconditions, élaboré sur la base des conditions de test.

Selon IBM[5] :
"Le test logiciel est le processus qui consiste à évaluer et à vérifier qu'un produit ou une application logicielle fait ce qu'il ou elle est censé(e) faire."

---

[4] ISTQB, Glossaire des termes utilisés en tests de logiciels, https://www.cftl.fr/wp-content/uploads/2018/10/Glossaire-des-tests-logiciels-v3_2F-ISTQB-CFTL-1.pdf

[5] IBM, Qu'est-ce que le test logiciel ?, https://www.ibm.com/fr-fr/topics/software-testing

Qu'est-ce qu'un test unitaire ?

Définition de la taverne du testeur[6]

Il obéit au principe "F.I.R.S.T." :

- **Fast :** S'exécute rapidement et est donc automatisé
- **Isolated :** Est indépendant des facteurs externes et des autres tests
- **Repeatable :** Isole les bugs automatiquement
- **Self-validating (autonome) :** N'est pas ambiguë (pas sujet à interprétation, ne demande pas une action manuelle pour vérifier le résultat)
- **Timely (tôt) :** Écrit en même temps que le code (même avant en TDD)

Si le test échappe à un de ces principes, il n'est probablement pas unitaire.

Pour isoler les bugs automatiquement, un test unitaire doit tester le code unitairement, et isolement. Il doit donc être écrit dans le but de tester la logique d'une ligne de code et si un test unitaire échoue vous pouvez connaitre la ligne de code incriminée.

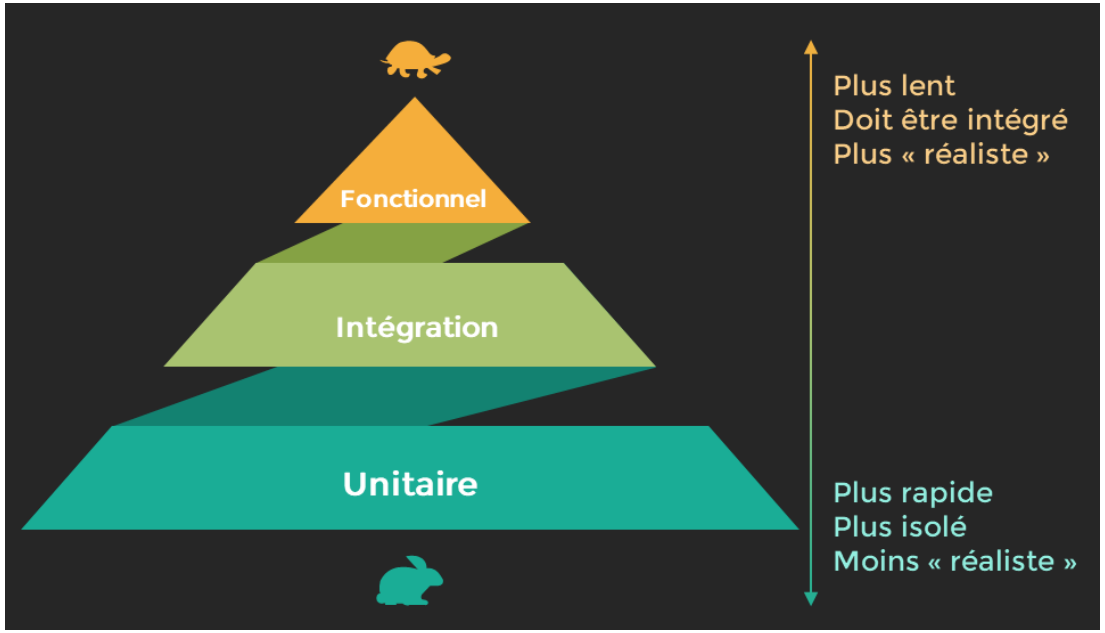
[6] Mais c'est quoi un test unitaire ?, https://lataverneduteseur.fr/2018/04/11/mais-cest-quoi-un-test-unitaire/

Testing, les frameworks de test unitaire

- UnitTest, le framework de test standard de Python
- PyTest, un framework pour tout faire en Python
- JUnit, le framework de test le plus utilisé en Java
- PHPUnit, le framework de test standard de PHP
- Google Test, pour le C++
- Jest pour JS, Babel, pour TypeScript, Node, React, Angular, Vue et plus encore
- etc

La Pyramide des tests

Différents types de tests peuvent être identifiés. La classification la plus classique étant la suivante[7] :



Mais d'autres valent le coup d'être découvertes...

[7] OPENCLASSROOMS, Testez votre code Java pour réaliser des applications de qualité, https://openclassrooms.com/fr/courses/6100311-testez-votre-code-java-pour-realiser-des-applications-de-qualite/6616481-decouvrez-les-tests-dintegration-et-les-tests-fonctionnels

Testing

- Un programme a pour entrée la standard input, et sorties les standard output et standard error, ce sont les standard streams
- Un programme se termine avec un code retour, le plus souvent un entier différent de 0 en cas de problème
- Les frameworks de tests retournent 3 statuts, OK, FAILED et ERROR
- Les OK si aucune erreur n'a lieu et FAILED en cas d'erreur d'assertion

```
chrichri@chrichri-HKD-WXX:~/Documents/Campus St Michel IT$ cat test_stdout_stderr.py
import sys

def test_hello():
    print("hello testing")
    print("stderr during testing", file=sys.stderr)
    assert False
chrichri@chrichri-HKD-WXX:~/Documents/Campus St Michel IT$ python3 -m pytest test_stdout_stderr.py  1> /dev/null
chrichri@chrichri-HKD-WXX:~/Documents/Campus St Michel IT$ python3 -m pytest test_stdout_stderr.py  2> /dev/null
============================================= test session starts ==============================================
platform linux -- Python 3.11.4, pytest-7.2.1, pluggy-1.0.0+repack
rootdir: /home/chrichri/Documents/Campus St Michel IT
collected 1 item

test_stdout_stderr.py F                                                                                  [100%]

================================================= FAILURES =================================================
_____________________________________________________ test_hello _____________________________________________________

    def test_hello():
        print("hello testing")
        print("stderr during testing", file=sys.stderr)
>       assert False
E       assert False

test_stdout_stderr.py:6: AssertionError
------------------------------------------- Captured stdout call -------------------------------------------
hello testing
------------------------------------------- Captured stderr call -------------------------------------------
stderr during testing
========================================== short test summary info ==========================================
FAILED test_stdout_stderr.py::test_hello - assert False
============================================== 1 failed in 0.10s ===============================================
chrichri@chrichri-HKD-WXX:~/Documents/Campus St Michel IT$ echo $?
1
```

Brief sur les outils de test logiciel

- Les frameworks de test retournent 3 statuts, OK, FAILED et ERROR .
- OK si aucune erreur n'a lieu et FAILED en cas d'erreur d'assertion, `AssertionError` en Python et Java.
- Pas de standard universellement accepté mais un format domine, le XML JUnit. Un XML le définit par la grammaire https://windyroad.com.au/dl/Open%20Source/JUnit.xsd.
- Vient de l'écosystème Java avec le package du même nom, JUnit, mais est présent dans la plupart des frameworks de test comme PHPUnit, Pytest.
- L'interopérabilité entre les outils est permise par cette XSD commune, mais pas plus de contrainte. Par exemple, l'interopérabilité entre les frameworks de test et les outils de CI/CD .
- Certains outils, souvent des solutions propriétaires, rechignent toujours à exporter un JUnit contenant les résultats de test pour fermer leur environnement.

L'assertion error

En Python

L'instruction `assert` vérifie une condition. Si la condition est vraie, cela ne fait rien et votre programme continue simplement à s'exécuter. Mais si la condition d'assertion est fausse, elle lève une exception `AssertionError`.

```
assert 23 % 2 == 0, "Le restant de la division est différent de 0"
```

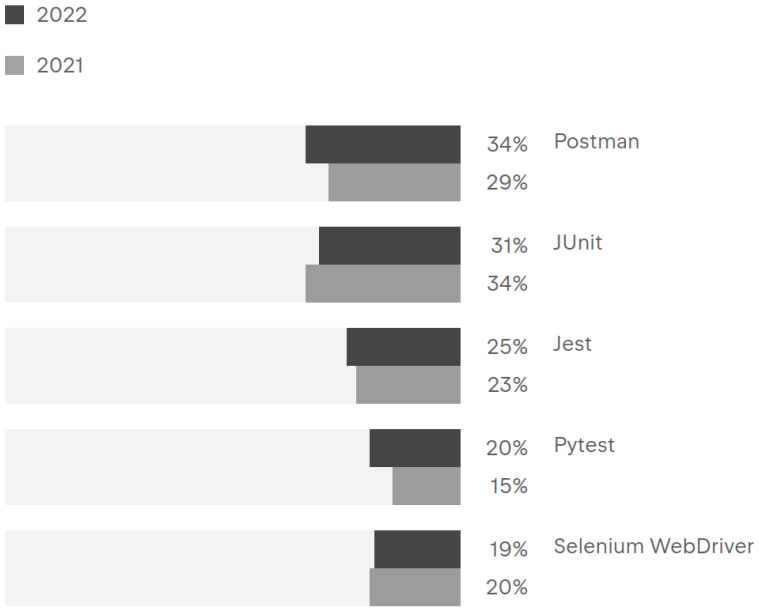
Étant toujours fausse, le programme crash.

```
chrichri@chrichri-HKD-WXX:~$ python3 -c 'assert 23 % 2 == 0, "Le restant de la
    division est différent de 0"'
Traceback (most recent call last):
  File "<string>", line 1, in <module>
AssertionError: Le restant de la division est différent de 0
```

Pourquoi Pytest pour le testing

"Pytest is a mature full-featured Python testing tool that helps you write better programs.[8]"

Pytest est le framework de test en Python le plus utilisé selon les sondages de JetBrains[9]. Seuls JUnit et Pytest sont pour tous usages, les autres sont orientés web.



[8] pytest: helps you write better programs , https://docs.pytest.org

[9] JetBrains, Which test frameworks , https://www.jetbrains.com/lp/devecosystem-2022/testing/

Pytest

Rédiger un test

- Un module de test Pytest est un module Python préfixé par `test_`.
- Toutes les méthodes préfixées par `test_` sont exécutées par Pytest. Qu'elles soient dans une classe ou non. Les autres méthodes ne sont exécutées uniquement si les tests les appellent.

Ici par exemple, la précédente assertion est intégrée dans une méthode nommée `test_divide` dans le module de test `test_division.py`.

```
def test_divide(): # Un test Pytest est préfixé par test_
    assert 23 % 2 == 0, "Le restant de la division est différent de 0."
```

Pytest

Lancer les tests

Lancer avec la commande `python -m pytest test_divison.py`, Pytest affiche le rapport de test.

```
...
def test_divide(): # Un test Pytest est préfixé par test_
> assert 23 % 2 == 0, "Le restant de la division est différent de 0."
E AssertionError: Le restant de la division est différent de 0.
E assert (23 % 2) == 0

test_divison.py:2: AssertionError
...
```

De très utiles et nombreuses options peuvent compléter cette ligne de commande. Pour les découvrir, lancez `python -m pytest --help` et “RTFM”.

Pytest

Les tests paramétriques

Les tests paramétriques sont des tests qui prennent un ensemble de paramètres en entrées.
Ils permettent de tester plusieurs cas avec un seul test.
Le rapport génère un cas de test par paramètre. Il est donc plus détaillé et permet de trouver les tests en échec plus facilement que si l'`assert` était dans une boucle, ce qui ne donne qu'un statut OK ou Fail dans le rapport.

```
import pytest
...
@pytest.mark.parametrize("dividend", range(100)) # Paramétrage du test
def test_divide_from_0_to_99(dividend): # Doit avoir un argument présent dans le
     paramétrage
   assert dividend % 2 == 0, "Le restant de la division est différent de 0."
```

Génère 100 cas de tests. Met en évidence les dividendes dont le restant de la division par 2 est différent de 0, ici 1 par exemple.

```
collecting ... collected 100 items

test_divison.py::test_divide_from_0_to_99[0] PASSED [ 1%]
test_divison.py::test_divide_from_0_to_99[1] FAILED [ 2%]
test_divison.py:7 (test_divide_from_0_to_99[1])
1~!= 0

Expected~:0
Actual ~:1
<Click to see difference>

dividend = 1

    @pytest.mark.parametrize("dividend", range(100)) # Paramétrage du test
    def test_divide_from_0_to_99(dividend): # Doit avoir un argument présent dans
          le paramétrage
> assert dividend % 2 == 0, "Le restant de la division est différent de 0."
E AssertionError: Le restant de la division est différent de 0.
E assert (1 % 2) == 0
```

Pytest

Plantage dans un test, quel statut?

Ce test crash avant l'assert.

```
def test_fail_or_error(): # Une erreur donne un fail ou error ?
    dividende = 23 / 0
    assert dividende % 2 == 0, \
        "Le restant de la division est différent de 0."
```

Malgré l'absence d'`assertionError`, le test est en échec et non en erreur.

```
test_divison.py::test_fail_or_error FAILED [100%]
test_divison.py:12 (test_fail_or_error)
def test_fail_or_error(): # Une erreur donne un fail ou error?
> dividende = 23 / 0
E ZeroDivisionError: division by zero

test_divison.py:14: ZeroDivisionError
```

Pour éviter les faux négatifs, un test doit tendre, tant que faire se peut, vers un `assert`. Pour cela pensez à utiliser les fonctions `parametrize` et `fixture`.

Pytest

Les fixtures

Cf. https://github.com/St-Michel-IT/testing/blob/main/test_customer_database.py

Un `return` ou un `yield` envoie l'objet au test dans l'état voulu.

```
def customer_without_table():
    """...
    """
    customer = Customer() # Avant le test, c'est le setup
    yield customer # Un yield évite de sortir de la fonction
    customer.con.close() # Après le test, le teardown
```

Le test n'a qu'une seule ligne, qu'un `assert` l'échec ne pourrait venir que de là.

```
def test_instantiation(customer_without_table):
    """...
    """
    assert isinstance(customer_without_table.con, Connection)
```

En cas de plantage dans la fixture, i.e., avant ou après le test, la sanction serait erreur et non échec.

Le rapport reflétera fidèlement le test et sera exempt de faux négatif.

Pytest

Les fixtures, le moyen du pattern A.A.A[10]

Le pattern du A.A.A, Arrange, Act, Assert, est facile à implémenter avec les fixtures. Arrange et Act sont isolés dans la fixture, Assert seul est dans le test.

```
@pytest.fixture()
def customer_without_table():
    """...
    """
    customer = Customer() # Arrange et Act dans le setup avant yiel ou return
    yield customer
    customer.con.close() # Pas de nom en A, ils l'ont oublié ?
```

Le test tend vers un `assert` quasi seul.

```
def test_instantiation(customer_without_table):
    """...
    """
    assert isinstance(customer_without_table.con, Connection) # Assert
```

Le B.D.D. suit le pattern A.A.A sous un autre nom : Given-When-Then. Le langage Gherkin utilise les étapes Given-When-Then pour spécifier les comportements dans les scénarios.

[10] Arrange-Act-Assert: A Pattern for Writing Good Tests, https://automationpanda.com/2020/07/07/arrange-act-assert-a-pattern-for-writing-good-tests/

Pytest

Exercice 1, écrire un test avec une fixture

Exercice : Écrivez votre premier test Pytest avec une fixture en respectant les exigences suivantes :

- Le test doit utiliser une fixture avec setup et teardown.
- Le test doit respect le pattern A.A.A .

Pour le setup et teardown, pensez à un fichier 😉.

Pour être évalué, créez un seul repo contenant tous les exercices de ce cours.

Pytest

Le scope des fixtures

L'argument `scope` du décorateur `pytest.fixture` défini la durée de vie de la fixture. Il peut prendre 3 valeurs :

- `function` : La valeur par défaut si on ne met rien. La fixture est exécutée à chaque fonction de test `def test_...` qui l'appelle directement ou indirectement, donc les éventuels setup et teardown de cette dernière aussi.
- `module` : Une fois par module `test_....py`.
- `session` : Une seule fois durant la session de test quel que soit le nombre de modules et de tests exécutés.

```
@pytest.fixture(scope="function") # Scope de la fixture, par default function
def customer_without_table():
    """Connection to in memory database using the Customer class"""
```

Pour tester une API en étant sûr le token n'est pas périmé, on peut utiliser le scope `function` pour en avoir un nouveau à chaque appel.
Si préparer les conditions initiales, le setup, prend du temps, on évitera si possible de le répéter à chaque test avec les scopes `module`, `session`.

Pytest

Le conftest.py

Il n'a qu'une seule particularité, c'est d'être importé automatiquement par Pytest lorsqu'il est présent dans le dossier courant des tests.
Il permet de factoriser les fixtures et les paramétrages de tests utilisés dans plusieurs modules de tests, car ces derniers seront automatiquement disponibles sans même un import dans le module.

```
./
├── conftest.py
├── test_no_directory.py
├── unit
│   └── test_integration.py
└── functional
    └── test_functional.py
```

Pour tous ces tests, s'ils sont lancés depuis la racine avec la commande `pytest`, le `conftest.py` sera importé automatiquement.

PyCharm intègre par défaut Pytest comme lanceur de tests et fournit l'autocomplétion pour les fixtures du `conftest.py`.

Pytest

Le coverage

Pas de support natif du coverage par Pytest, il faut installer le plugin `pytest-cov`[11].

Pour appeler le plugin, il faut passer en argument le module à tester et le(s) test(s) de ce dernier.

```
pytest --cov=\textcolor{flatgreen}{customer_database} ./test_customer_database.
    py
...
rootdir: /home/chrichri/Documents/Campus-St-Michel-IT/testing
plugins: cov-4.1.0
collected 6 items

test_customer_database.py ...... [100%]

---------- coverage: platform linux, python 3.11.4-final-0 -----------
Name Stmts Miss Cover
-----------------------------------------
customer_database.py 9 0 100%
-----------------------------------------
TOTAL 9 0 100%
```

[11] Welcome to pytest-cov's documentation!,

Pytest

Le coverage

Coverage for **customer_database.py**: 100%

9 statements | 9 run | 0 missing | 0 excluded

« prev ^ index » next coverage.py v7.3.1, created at 2023-09-28 22:38 +0200

```
1  import sqlite3
2
3
4  class Customer:
5      """
6      Customer class containing all the method to interact with the customer table
7      """
8
9      def __init__(self, path=":memory:") -> None:
10         """
11         Connect to in memory database by default, or to a database file if
12         specified.
13         """
14         self.con = sqlite3.connect(path)
15
16     def create_table(self) -> None:
17         """
18         Create the customer table if not exists
19         """
20         self.con.execute("""
21             CREATE TABLE IF NOT EXISTS customer (
22                 id INT PRIMARY KEY NOT NULL,
23                 email TEXT NOT NULL
24             )
25             """)
26
27     def insert(self, customer_id: int, email: str) -> None:
28         """
29         Insert a new customer in the table.
30
31            :param customer_id: The customer id as an int
32            :param email: The customer email as a string
33         """
34         self.con.execute("""
35             INSERT INTO customer (
36                 id,
37                 email
38             ) VALUES (
39                 ?, ?
40             )""", (customer_id, email))
41         self.con.commit()
```

« prev ^ index » next coverage.py v7.3.1, created at 2023-09-28 22:38 +0200

L'option `--cov-report=html` permet de générer un rapport HTML plus détaillé qui met en lumière quelles parties du code sont couvertes par les tests et lesquelles ne le sont pas.

Exercice : Trouvez quel test du module https://github.com/St-Michel-IT/testing/blob/main/test_customer_database.py couvre quelle partie de ce code:

Pytest

Le secret de la réussite

Pytest est un framework complet voir complexe.
Mais quasi toutes les fonctionnalités auxquelles on peut penser sont déjà implémentées et décrites une documentation de qualité.
Comme avec toutes les grosses libraries, il faut avoir confiance en leur design et chercher dedans avant de réinventer la roue.
La liste des fonctionnalités est trop longue, nous venons juste d'en découvrir les principales.

Donc une fois encore, comme pour toutes les bonnes libraires à connaître, le secret est "**RTFM**" !

Exercice 2 : Toujours à partir de la librairie du repo https://github.com/St-Michel-IT/testing/, où le module `customer_database.py` est testé par `test_customer_database.py`. Ajoutez une méthode qui modifie les données d'un client et les tests qui vont avec.

Exercice 3 : Toujours dans le même repo, le module `witness_number.py` est un début implémentation d'une généralisation du petit théorème de Fermat expliquée sur la chaîne Youtube Numberphile par Matt Parker[12]. Cette librairie n'est pas déjà libre sur le web. Finissons son développement, en implémentant **toutes les fonctionnalités de la vidéo** de M. Parker avec des standards de qualité élevés grâce à nos connaissances en testing.

[12] Numberphile, Witness Numbers (and the truthful 1,662,803), https://www.youtube.com/watch?v=_MscGSN5J6o

Mutation testing

Vérifier la qualité des tests unitaires

Si on prend la suite de Fibonacci comme exemple :

```
def fibonacci(n: int) -> int:
    """
    According to https://www.dcode.fr/nombres-fibonacci.
    This should return the n-th Fibonacci number.

      :param n: The n-th
      :return: The corresponding Fibonacci number
    """
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a
```

Imaginez un test sans assert, il ne fait qu'exécuter le code. Tout le code est couvert mais le test ne sert à rien.

```
def test_fibonacci():
    fibonacci(5)
```

Quel peut être le coverage ?

Mutation testing

Vérifier la qualité des tests unitaires

Il peut être de 100%. Encore une fois cette métrique n'est pas suffisante pour juger de la qualité des tests.

La mutation est une étape qui intervient **quand tous les tests unitaires passent uniquement.** Cette étape consiste à modifier le code source pour voir si les tests échouent. C'est un raisonnement par l'absurde. Ils doivent en effet échouer pour prouver qu'ils testent le comportement du code et ne font pas que l'exécuter.

Mutation testing

Fonctionnement des librairies comme `mutmut`[13]

La librairie de mutation essaie de modifier le code avec par exemple les règles suivantes :

- Changer les opérateurs mathématiques, comme `+` en `-`, `>` en `>=`
- Changer les opérateurs logiques : `and` en `or` et inversement
- Intervertir les mots-clés, comme `True`/`False`, `in`/`not` in
- Intervertir `copy` et `deepcopy`
- Remplacer `""` par `None`
- Changer les valeurs numériques en ajoutant `1`
- Changer les valeurs des chaînes de caractères en ajoutant `"xx"`
- Changer les noms des arguments des fonctions en ajoutant `"xx"`

[13] mutmut - python mutation tester, https://mutmut.readthedocs.io/en/latest/

Mutation testing

Example avec mutmut

Dans le dossier `mutmut_fib` il y a 11 tests unitaires qui passent :

```
$ python -m pytest tests/test_fib.py
================================================================= test session
     starts ======
platform linux -- Python 3.11.6, pytest-7.4.2, pluggy-1.3.0
rootdir: /home/chrichri/Documents/Campus-St-Michel-IT/testing/mutmut_fib
plugins: cov-4.1.0
collected 11 items

tests/test_fib.py ........... [100%]

================================================================= 11 passed in
     0.04s ======
```

Le framework de mutation testing créé un ou plusieurs mutants, i.e, code source modifié.

- S'il est détecté par les tests, qu'une sanction au moins change, il est dit "tué".
- Si aucune sanction ne change, que tous les tests passent encore, c'est un "survivants". Cela indique que les tests ne sont pas suffisants.

Mutation testing

Example avec mutmut

Ici `mutmut` créé 5 mutants, donc 5 versions différentes de la fonction `mutmut_fib.src.fib.fibonacci`, tous tués.

Exemple de mutant :

```
$ mutmut show 3
--- src/fib.py
+++ src/fib.py
@@ -1,6 +1,6 @@
 def fibonacci(n: int) -> int:
     """Get the n-th Fibonacci number"""
- a, b = 0, 1
+ a, b = None
     for _ in range(n-1):
         a, b = b, a + b
     return a
```

Mutation testing

Example avec mutmut

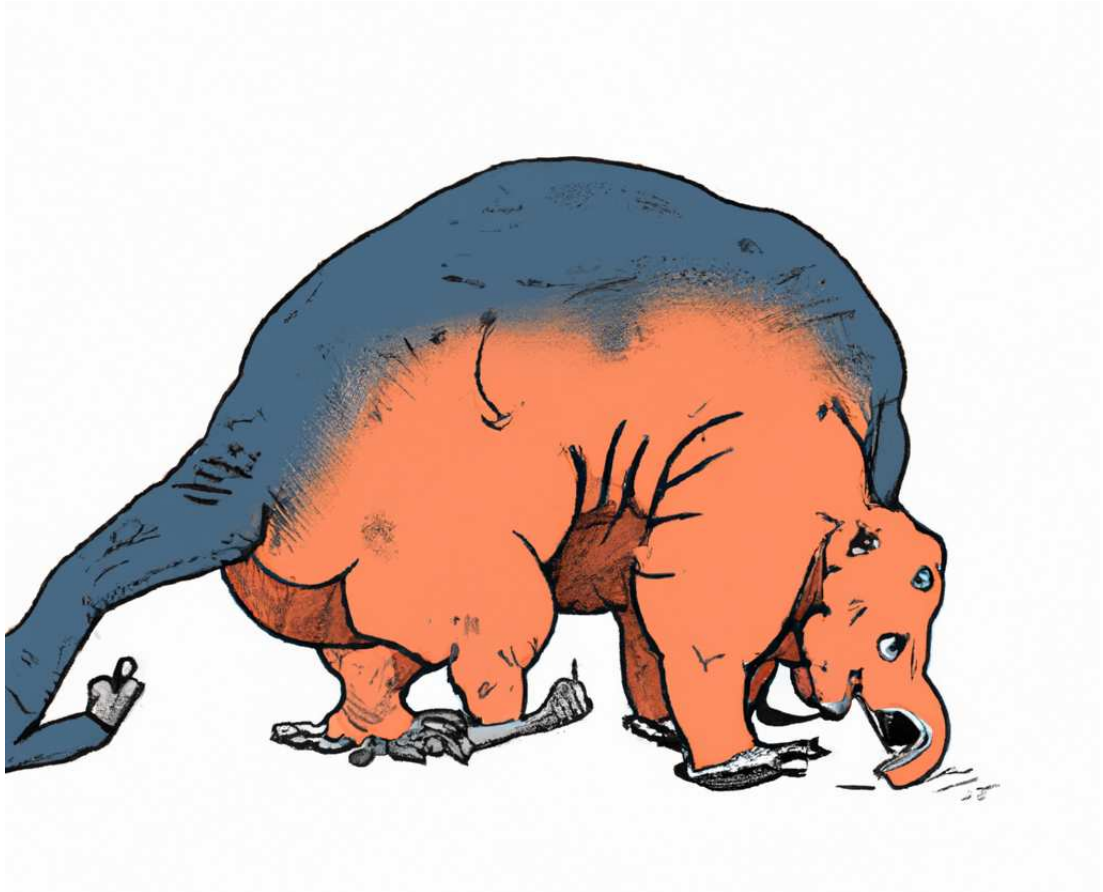
```
$ mutmut run

- Mutation testing starting -

These are the steps:
1. A full test suite run will be made to make sure we
   can run the tests successfully and we know how long
   it takes (to detect infinite loops for example)
2. Mutants will be generated and checked

Results are stored in .mutmut-cache.
Print found mutants with `mutmut results`.

Legend for output:
🎉 Killed mutants.   The goal is for everything to end up in this bucket.
⏰ Timeout.          Test suite took 10 times as long as the baseline so were killed.
🤔 Suspicious.       Tests took a long time, but not long enough to be fatal.
🙁 Survived.         This means your tests need to be expanded.
🔇 Skipped.          Skipped.

1. Running tests without mutations
⠿ Running...Done

2. Checking mutants
⠿ 5/5  🎉 5  ⏰ 0  🤔 0  🙁 0  🔇 0
```

Mutation testing

Conclusion

Un rapport de test unitaire devrait contenir :

- Les rapports de test avec les résultats des tests unitaires
- Le rapport de test coverage
- Le rapport de mutation testing



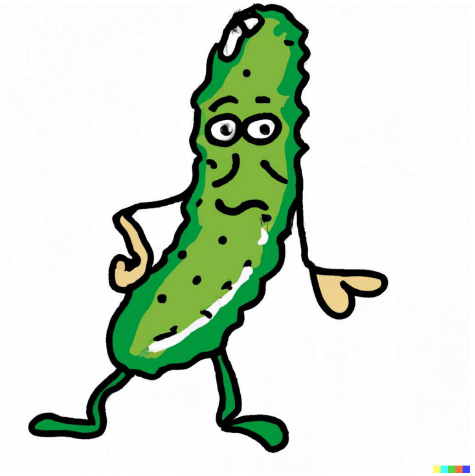
Tests avec langage naturel

A.K.A le BDD

Les slides qui suivent sont truffés de biais négatifs sur le BDD suite à de mauvaises expériences 😅.

Le BDD est le sigle de Behavior-Driven Development.

Vous remarquerez qu'il n'y ici aucune mention liée au testing ?
Alors pourquoi est-il régulièrement associé au testing ?



Parce que le BDD est un outil de communication entre les développeurs et les métiers qui permet à ceux qui spécifient le produit de le décrire par rapport au comportement (behavior) attendu. Le tout dans une syntaxe proche du langage naturel. Ce langage peut être utilisé pour générer des tests. On pourrait presque finir par croire que le métier peut écrire les tests.

Tests avec langage naturel

Cucumber ne serait même pas un outil de test

Dans cette discussion sur Ycombinator, le créateur de Cucumber affirme plusieurs choses intéressantes dont :

- "The biggest problem with Cucumber is that most people trying it out don't understand what it is."
- "Cucumber is not a tool for testing software."
- "Cucumber is a tool that facilitates collaboration and software design (especially domain-driven design)."
- "The purpose of Cucumber is to bridge the communication gap between business and IT by providing a small set of essential scenarios to illustrate core behaviour of unwritten software."

Mais comme c'est au programme et que ça ne choque plus grand monde, on va s'en servir d'outil de test quand même. 😇

Spécifier avec le BDD

Exemple de Cucumber, le plus répandu

Développé en début des années 2000 par Daniel Terhorst-North, il fait entre autre le constat que : “Behaviour” is a more useful word than “test”[14].
Il utilise les mots clés “Given”, “When” et “Then” pour décrire le comportement attendu du produit en développement dans un scénario.

```
Given some initial context (the givens),
When an event occurs,
Then ensure some outcomes.
```

Par exemple :

```
Given I just openned a new private window in my browser
When I browse the https://in-france.fr website
Then I wan to descover the french economy by activity
And by region.
```

On comprend facilement que ce sont des critères d’acceptation haut niveau.
Les tests liés seront donc des tests d’acceptation/fonctionnels.

[14] Daniel Terhorst-North, Introducing BDD, https://dannorth.net/introducing-bdd/

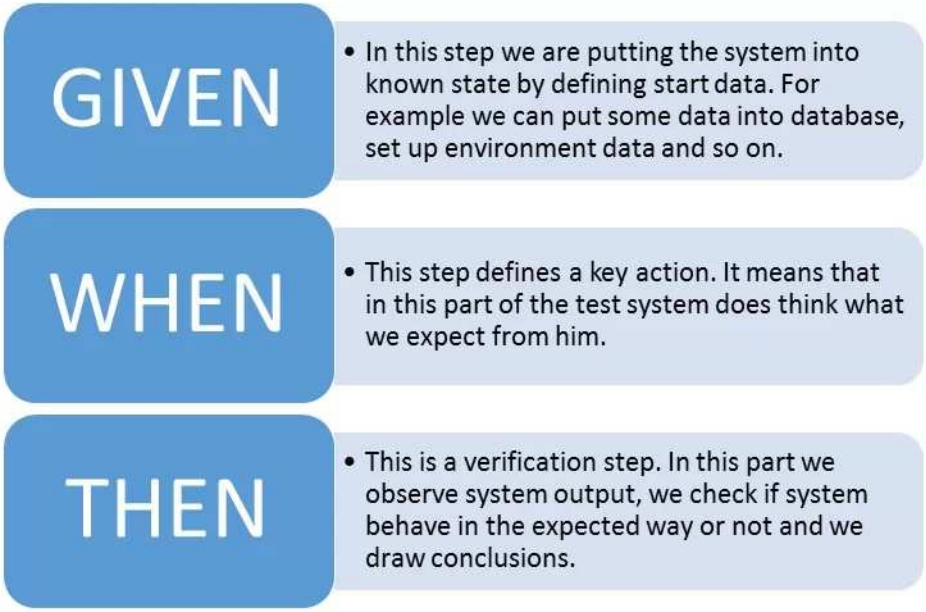
Tests avec langage naturel

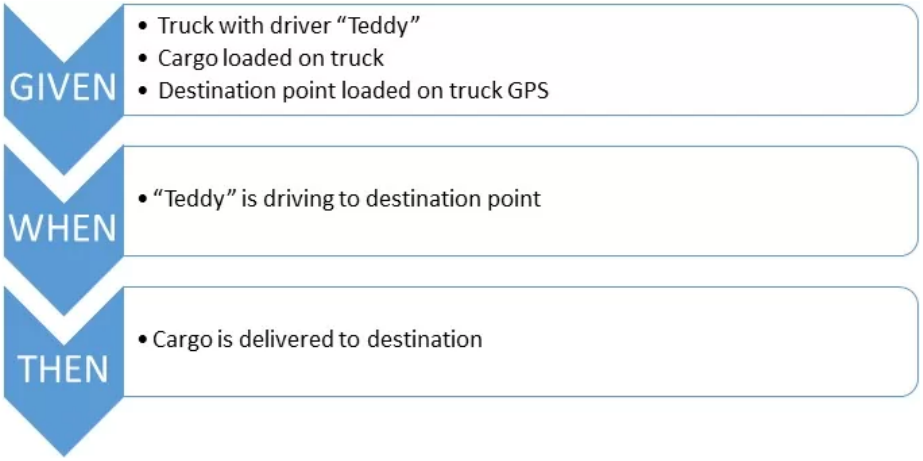
Définitions

- “Feature” : Une fonctionnalité à développer et tester. Les tests correspondant à cette fonctionnalité sont regroupés dans un fichier `.feature`.
- “Scenario” : Un scénario est une suite d’étapes qui décrivent un comportement attendu. Ils peuvent être plusieurs dans un fichier `.feature`.
- “Scenario Outline” : Un scénario paramétrique, les données d’entrée sont définies dans un tableau.
- “Given” : Les conditions initiales du scénario.
- “When” : L’action qui déclenche le scénario.
- “Then” : Les résultats attendus.
- “And” : Pour ajouter des conditions supplémentaires aux 3 mots clés précédents.

Spécifier avec le BDD

Exemple de Cucumber[15]





Bonnes pratiques :

- Un scenario peut s'exécuter sans 1 de ces 3 mots clés, mais il est considéré comme une bonne pratique de toujours avoir les 3.
- Provoquer un crash dans le "then" avec un `assert` ou tout autre helper similaire si le résultat attendu n'est pas vérifié.

[15] Given-When-Then pattern in unit tests, https://www.j-labs.pl/blog-technologiczny/given-when-then-pattern-in-unit-tests/

Spécifier avec le BDD

Exemple de Cucumber, le plus répandu

Les données d'un test paramétrique sont définies dans un tableau[16].

```
Example
    [Arguments] ${periodClosed} ${periodOpenAndModified} ${importDay} ${oldManagerValidUntil} ${
          newManagerValidFrom}

    Given initialized criteria for bonus commercial
    And a branch B with branch manager M_OLD and employee E1
    And evaluation for E1 for period ${periodClosed} which is closed
    And evaluation for E1 for period ${periodOpenAndModified} which is open and modified

    When M_NEW becomes new manager of branch B
    And import service is called on ${importDay}
    Then the new branch manager of branch B is M_NEW valid from ${newManagerValidFrom}
    And branch manager M_OLD manages employee E until ${oldManagerValidUntil}
    And branch manager M_NEW manages employee E from ${newManagerValidFrom}
    And Evaluations for E1 still have the same content

| *Test Case* | | *Closed Period* | *Open Period* | *Run Import On* | *Old Manager Stops* | *New Manager
      Starts* |
| 1 | Example | 1.11.2009 - 30.11.2009 | 1.12.2009 - 31.12.2009 | 11.11.2009 | 30.11.2009 | 1.12.2009
| 2 | Example | 1.11.2009 - 30.11.2009 | 1.12.2009 - 31.12.2009 | 1.11.2009 | 31.10.2009 | 1.11.2009
```

[16] Given/When/Then And Example Tables Using the Robot Framework, https://www-codecentric-de.translate.goog/wissens-hub/blog/givenwhenthen-and-example-tables-using-the-robot-framework?_x_tr_sl=en&_x_tr_tl=fr&_x_tr_hl=fr&_x_tr_pto=sc&_x_tr_hist=true

BDD qui a-t-il derrière ce langage ?

Quelques exemples en Python et Java

Structure d'un projet Behave en Python[17] :

```
features/
features/everything.feature
features/steps/steps.py
```

Derrière une phrase :

```
@then('the result page will include "{text}"')
def step_impl(context, text): # context is a global object that is passed between all steps
    if text not in context.response:
        fail('%r not in %r' % (text, context.response))
```

Avec Cucumber en Java[18] :

```
@When("the stock is traded at $price")
public void theStockIsTradedAt(double price) {
    stock.tradeAt(price);
}
```

C'est une illusion de simplicité, derrière le langage naturel, il y a quasiment la même algorithmie qu'en Pytest.

[17] Tutorial, https://behave.readthedocs.io/en/latest/tutorial/#features

[18] Cucumber Java, https://cucumber.io/docs/guides/10-minute-tutorial/

Exercice 4 de BDD

Exemple de Cucumber, le plus répandu

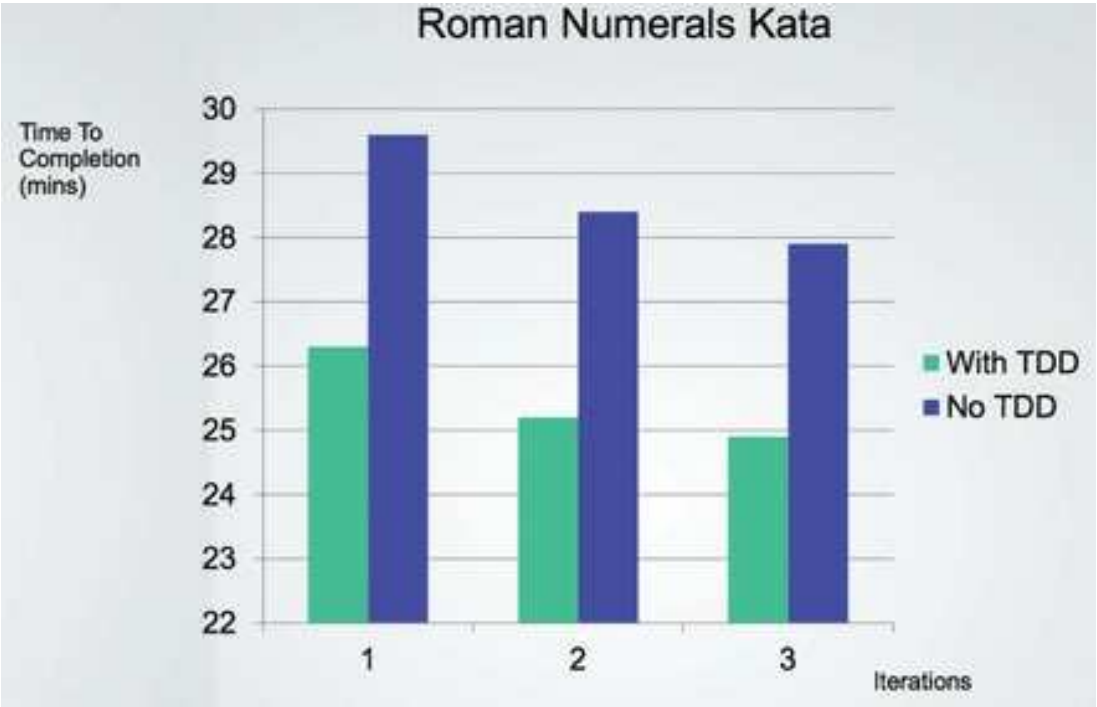
Exercice : En utilisant un maximum du code de l'exercice précédent sur les nombres témoins, réécrivez l'équivalent des tests pytest ou autre en scénarios Behave.

Le TDD

Test Driven Development

Vouloir aller plus vite sans faire de test semble être une mauvaise idée.
Ce qui pratiquent le TDD vont plus vite dès le début d'un projet[19] !
Entre autre, les tests unitaires font réduire la taille des classes, functions et des méthodes, ce qui améliore architecture.

"The only way to go fast, is to go well[19]."



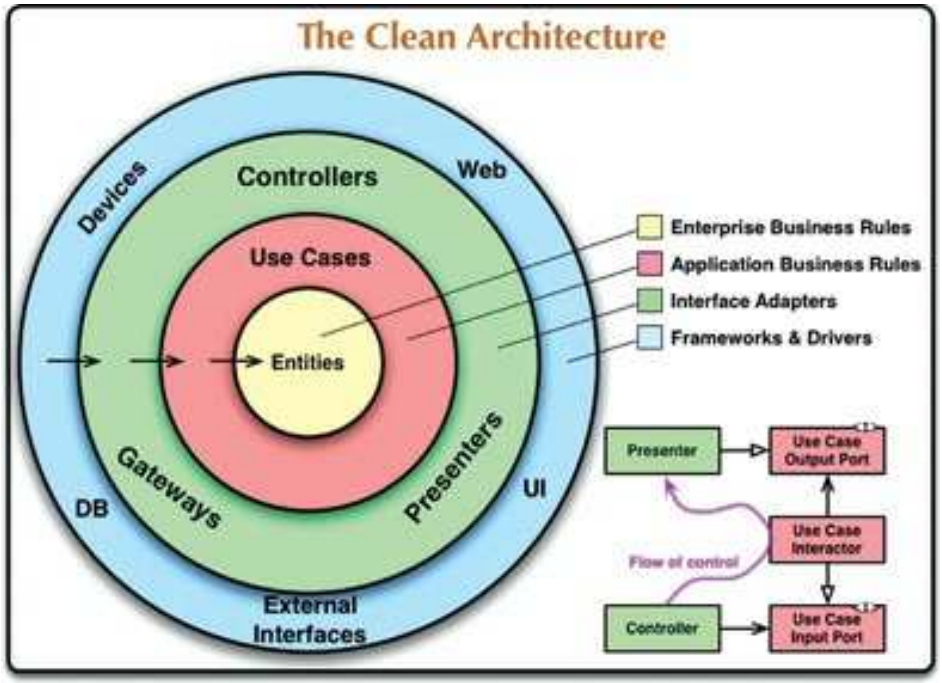
[19] Clean architecture, Robert C. Martin

Le TDD

Quel type(s) de test ?

Tous les types de tests possibles sont à écrire dès que possible avant même de commencer à coder. Les tests unitaires sont en général les plus simples à écrire et les plus rapides à exécuter.

Les éléments centraux, les règles de gestions, les “business rules” doivent être testables sans les éléments périphériques[19] :



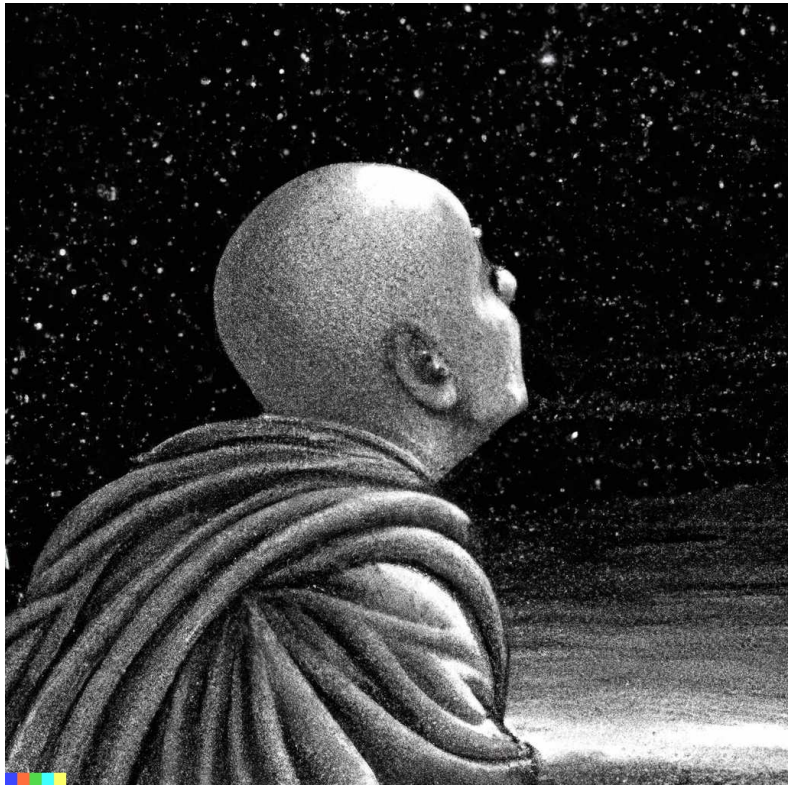
Le TDD

Quel types de test ?

Edsger Dijkstra : “Testing shows the presence, not the absence of bugs.”

Autrement dit, l’univers des tests est infini, on n’a jamais la certitude de suffisamment tester.

Même le coverage est une mauvaise mesure.



Le TDD

Exercices 5

De nombreux exercices de TDD sont disponibles sur le web comme par sur GitHub, https://github.com/gabbloquet/entrainement-au-tdd.

En 30 minutes, sans assistant du type ChatGPT, réaliser l'un des plus classique d'entre eux, le "FizzBuzz", https://github.com/gabbloquet/entrainement-au-tdd/tree/master/src/main/java/io/github/gabbloquet/tddtraining/FizzBuzz . Respecter l'esprit du TDD qui veut que l'on écrive les tests avant le code :

1. Écrire un test par requirement.
2. Écrire l'algorithme.
3. Refactoriser.
4. Revenir à l'étape 2 jusqu'à ce que tous les tests passent.

Tests assisté par l'IA

Test de GitHub Copilot le 30 mai 2024 Based on your request, here are some unit tests for the `Customer` class in the `customer_database.py` module. These tests cover the happy path and some edge cases for the `create_table` and `insert` methods. The test names are self-explanatory and do not start with "test". The tests are written using the `unittest` module, which is a built-in module in Python for writing tests.

```
import sqlite3
import unittest
from unittest.mock import Mock

from customer_database import Customer

class CustomerDatabaseTests(unittest.TestCase):
    def setUp(self):
        ...
    def creating_customer_table(self):
        self.customer.create_table()
        ...
```

These tests use the `unittest.mock` module to mock the `sqlite3.connect` function, allowing us to test ...

Tests assisté par l'IA

Test de GitHub Copilot le 30 mai 2024

```
$ python -m unittest test_customer_database_with_copilot.py

----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
```

"All tests should start with def test_ no ? To be compliant with unittest?"

Yes, you're correct. In Python's `unittest` module, test methods should start with `test_`. This is how the test runner identifies which methods represent tests. Here's how you can modify the test methods in your `CustomerDatabaseTests` class:

Tests assisté par l'IA

Test de GitHub Copilot le 30 mai 2024

```
class CustomerDatabaseTests(unittest.TestCase):
    def setUp(self):
        ...
    def test_creating_customer_table(self):
        ...
```

Now, each test method starts with `test_`, making them compliant with `unittest`.

```
Error
Traceback (most recent call last):
  File "/home/chrichri/Documents/Campus-St-Michel-IT/testing/ble_to_rpc_client.
      py", line 9, in setUp
    self.customer = Customer(self.mock_db)
                    ^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/chrichri/Documents/Campus-St-Michel-IT/testing/customer_database.
      py", line 14, in __init__
    self.con = sqlite3.connect(path)
               ^^^^^^^^^^^^^^^^^^^^^
TypeError: expected str, bytes or os.PathLike object, not Mock
```

En conclusion, on a encore besoin de l'humain...

Tests assisté par l'IA

Cover Agent Un framework open source qui utilise la puissance de l'IA pour générer des tests unitaires. Compatible avec la plupart des LLMs grâce à LiteLLM handles loadbalancing, fallbacks and spend tracking across 100+ LLMs. All in the OpenAI format. Voir sa page GitHub[20].

Exercice 6 : Tester le code `https://github.com/St-Michel-IT/testing/blob/main/test_customer_database.py` avec Cover Agent.

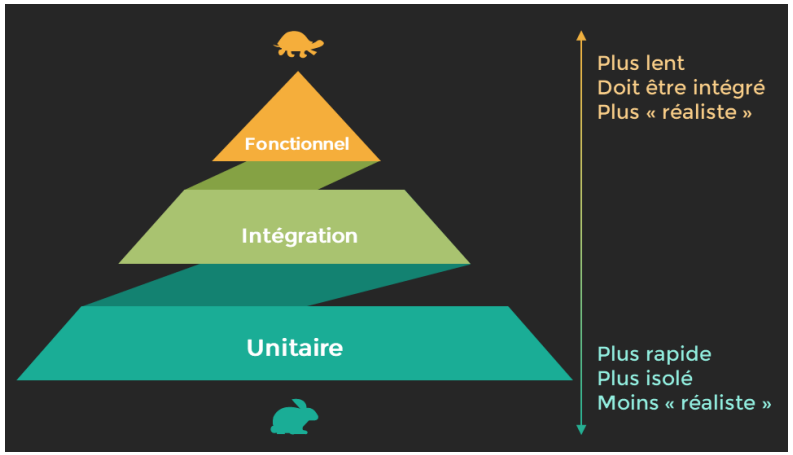
[20] Cover Agent, `https://github.com/Codium-ai/cover-agent`

Les tests d'intégration

définition

Les tests d'intégrations sont des tests effectués entre les composants afin de s'assurer du fonctionnement des interactions et de l'interface entre les différents composants. Ces tests sont également gérés, en général, par des développeurs[21].

C'est aussi la définition de l'ISTQB[22] .



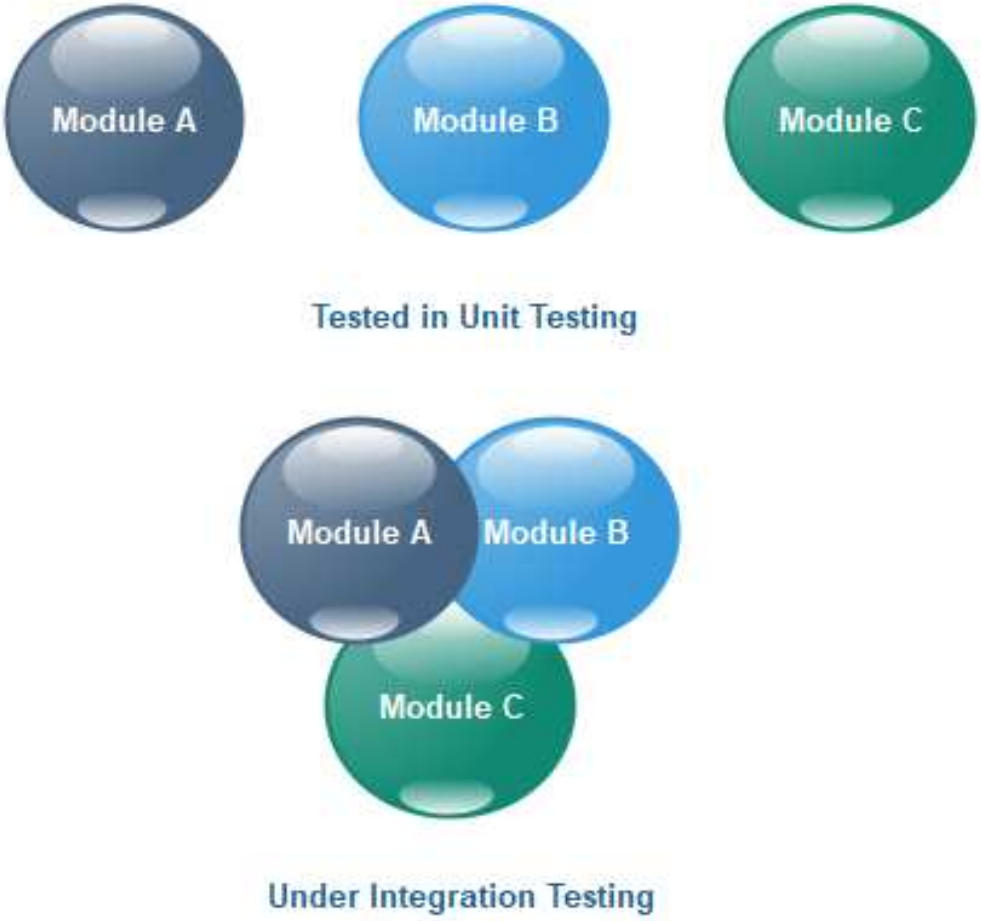
[21] Les niveaux de test, https://lataverneduteseur.fr/2017/11/03/les-niveaux-de-test/

[22] Glossaire CFTL/ISTQB des termes utilisés en tests de logiciels, https://www.cftl.fr/wp-content/uploads/2018/10/Glossaire-des-tests-logiciels-v3_2F-ISTQB-CFTL-1.pdf

Les tests d'intégration

définition

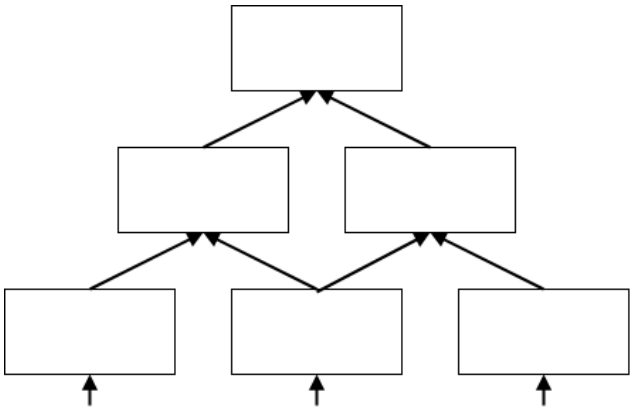
Autrement dit, les tests unitaires ont testé chaque composant, ceux d'intégration testent les interactions entre les composants.



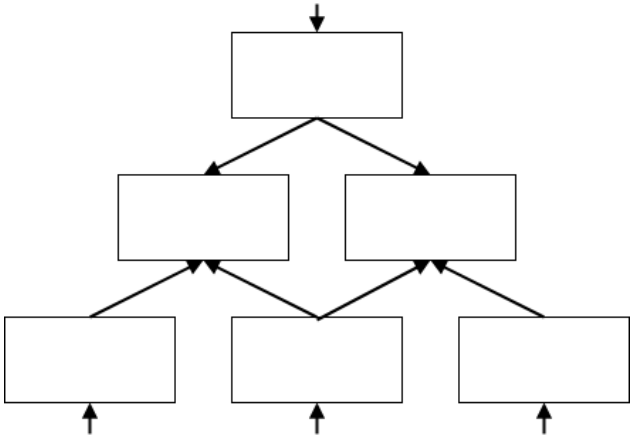
Les tests d'intégration

les différentes approches, les top-down, bottom-up et mixte[23]

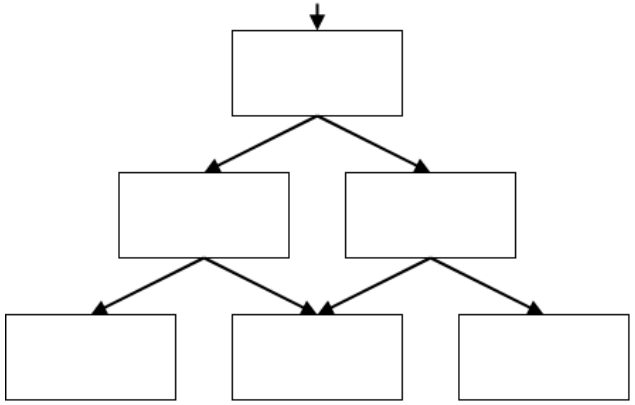
La top-down consiste à tester les composants les plus hauts de la hiérarchie en premier.
La bottom-up consiste à tester les composants les plus bas de la hiérarchie en premier.



Bottom-up



Mixte



Top-down

[23] Tests d'intégration, https://mbf-iut.i3s.unice.fr/lib/exe/fetch.php?media=undefined:7-testintegration-2016.pdf

Les tests d'intégration

Bonnes pratiques

- Ne pas tester la logique métier, c'est le rôle des tests unitaires.
- Ne pas les mélanger à d'autres tests, unitaire ou autre.
- Ils testent les interfaces.
- Ils sont généralement plus longs à jouer, il est donc conseillé de bien séparer les suites de test pour pouvoir lancer un sous-ensemble précis.

Les tests d'intégration

Exemple

Les 15 lignes de code https://github.com/St-Michel-IT/testing/blob/master/ble_to_rpc.py permettent de récupérer des données du Bluetooth Low Energy (BLE) et les distribuer "over" HTTP au format JSON RPC .

C'est donc l'intégration de 3 librairies :

- `bleak` pour le BLE .
- `Werkzeug` pour le serveur HTTP .
- `json-rpc` pour le format des données.

Le tester est donc bien un test d'intégration de ces librairies.

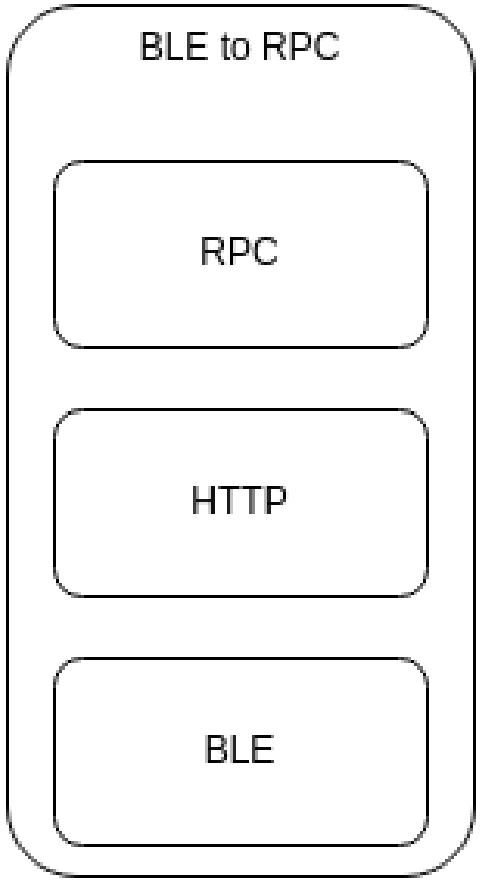
Les tests d'intégration

Exercice 7

Proposer et rédiger des tests d'intégration pour le code `https://github.com/St-Michel-IT/testing/blob/master/ble_to_rpc.py`. Dans la techno de votre choix.

Tester les 3 protocoles intégrés.

Ajouter la fonction de connection BLE à un appareil défini par son adresse et lire la caractéristique "GATT" `2A24`. Rédiger également les tests correspondants.

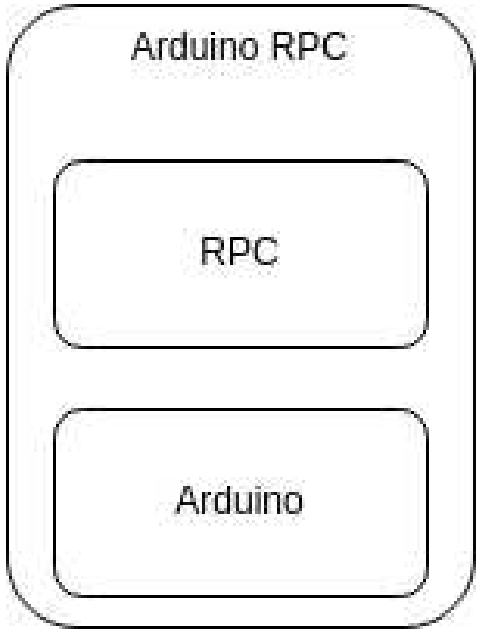


Les tests d'intégration

Exercice 8

Ils peuvent aussi être utilisés en embarqué, IOT, etc.

En lisant la documentation Arduino https://github.com/St-Michel-IT/testing/blob/main/doc/Arduino%C2%AE%20UNO%20R3%20Datasheet.pdf, ajouter des tests d'intégration au module de test https://github.com/St-Michel-IT/testing/blob/main/test_arduino.py.



Intégration continue A.K.A. CI

Les defintions du CI/CD

CI/CD est l'acronyme de "Continuous Integration/Continuous Delivery" ou "Continuous Deployment".

"le but ultime du processus CI/CD qui consiste à supprimer l'intervention humaine pour gagner en temps et en efficacité. Le code est compilé et testé sur l'ordinateur grâce à ce que l'on appelle un build automatisé. S'il n'y a pas de bug le tout est mis en production automatiquement. Cette approche permet aux développeurs de se concentrer sur la conception et le développement et sur les fonctionnalités de l'application."[24]

[24] Qu'est-ce que l'approche CI/CD - Qu'est-ce que le processus CI/CD ?, https://www.oracle.com/fr/cloud/definition-approche-ci-cd/

Intégration continue A.K.A. CI

Les defintions du CI/CD

Plusieurs remarques :

- Une des avancées est l'automatisation de multiples taches.
- Le mot intégration fait penser qu'on a plusieurs systèmes, logiciels, développeurs, équipes, etc.
- L'"ordinateur" n'est pas défini, on verra qu'il peut être celui du développeur ou un serveur distant voir un ordinateur inconnu.
- "S'il n'y a pas de bug le tout est mis en production automatiquement." vous y croyez?
- Le but est toujours le même, éliminer un maximum de bug.

Histoire du CI/CD

Une histoire récente d'une vingtaine d'années

- 2005, première version de Hudson, le précurseur de Jenkins par Kohsuke Kawaguchi.
- 2010, le terme "Continuous Integration" est utilisé pour la première fois dans un livre par Jez Humble[25].
- 2011, Jenkins est libéré.
- 2024, plus de 38 plateformes mainstream[26].

[25] Continuous Delivery, https://www.continuousdelivery.com/

[26] 38 Best CI/CD Tools For 2024, https://www.lambdatest.com/blog/best-ci-cd-tools/

Les avantages du CI/CD

Si tout peut être réalisé en local, pourquoi utiliser un serveur ?

- Les plateformes de CI/CD sont avant tout des outils collaboratifs. Elles permettent le partage des développements dans l'équipe et dans l'entreprise. On peut suivre en direct l'état d'avancement et la qualité des développements de ses collègues.
- Elles s'intègrent à de nombreux outils comme les SSO, Git, les outils de gestion de projet et de qualité comme ceux d'Atlassian (Jira, Confluence, Bitbucket, etc).
- Une UX/UI plus simple que des données un terminal. La prise en main peut-être faite par des métiers autre que celui de développeur (Testeur, Release Manager, Hardware).
- Permet de partager les ressources hardware requises pour la compilation, les tests, entre des machines plus ou moins disponibles. E.g., dans l'embarqué, cela permet de ne pas multiplier le hardware et de limiter le nombre de bancs de test. En général, la compilation qui peut prendre beaucoup de ressources CPU et peut être délocalisée sur un serveur dédié.

Les avantages du CI/CD

Quelles différences avec le DevOps ?

Le CI/CD est un des outils du DevOps.

C'est l'outil dédié au développement et à la livraison de logiciels de qualité. On confond parfois les deux car le terme DevOps étant plus hype que CI/CD, est plus souvent est utilisé.

Le fondamental du CI/CD

Livre et site web de Jez Humble[25]

Jez Humble fait un travail academic depuis les années 2000 pour mesurer l'impact du CI/CD sur les professionnels de l'IT .

En plus de plusieurs paramètres liés à la qualité logiciel, il mesure même un impact positif sur l'ambiance de travail !

Son livre sur le sujet, "Continuous Delivery", est une référence dans le domaine. Son site web `https://www.continuousdelivery.com/` contient déjà beaucoup de ressources.

Le business du CI/CD

Ce n'est pas une simple technologie mais un marché, attention aux coûts

Git, Python, que nous avons vu, sont des technologies open source devenues des standard. Toutes leurs fonctionnalités sont disponibles gratuitement. Contrairement à elles, certaines plateformes de CI/CD sont payantes.

Parmi les business model, on trouve :

- Le freemium, une version gratuite limitée en fonctionnalités.
  - Github est gratuit jusqu'à une certaine consommation de ressources (temps CPU, disque, etc) et est limité en fonctionnalités.
  - Gradle est toujours "full featured" en local mais Devlocity, le SAAS qui va avec et rajoute des fonctionnalités est payant.
  - Gitlab Community Edition VS Entreprise edition.
- L'auto-hébergement d'une solution complète open source et le on-premise.
  - Jenkins.
- Les solutions propriétaires.
  - TeamCity est payant en auto-hébergement et on-premise.
  - Travis CI .

Les différences entre plateformes

À lire absolument, un comparatif des différentes plateformes par Atlassian https://www.atlassian.com/fr/continuous-delivery/continuous-integration/tools.

Les différences sont nombreuses, on peut citer :

- Certains permettent de créer des clusters sur n'importes quelles machines. Par exemple Jenkins avec son cluster formé de "slave(s)" et d'un "master". Tout ce qui se passe sur les "slaves" est visible sur l'IHM du "master".
- AWS CodePipeline permet d'avoir de multiples nodes mais sur des machines AWS uniquement 🤑.
- Devlocity n'a qu'une instance centrale du serveur
- Leur communauté, une communauté open source peut développer des plugins, des scripts, des outils, etc, pour virtuellement toutes les applications qui ne sont pas prévues dans la version de base. De ce point de vue, Jenkins est dans les premiers.
- L'organisation en jobs ou en pipelines de jobs.

Les limites du CI/CD

Quelles sont les limites du CI/CD ?

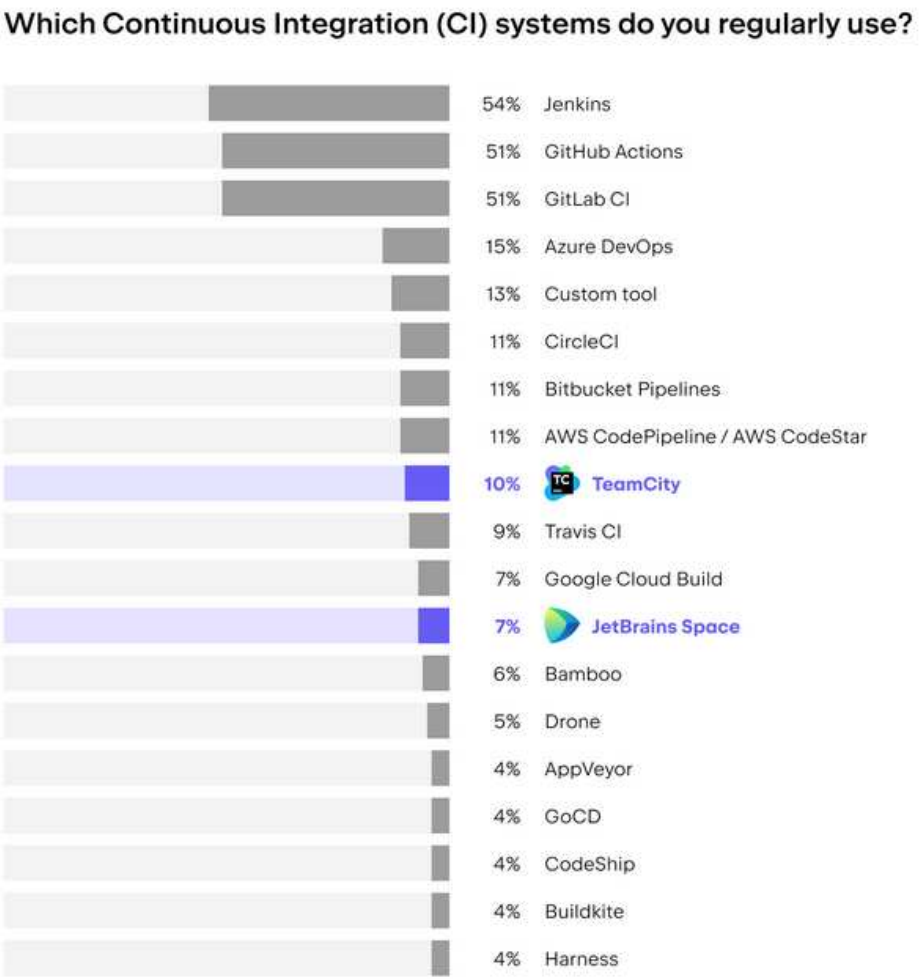
Les limites du CI/CD

Quelles sont les limites du CI/CD ?

- le coût des licences ou des machines.
- Pas de limites sur le type de logiciel, on peut y connecter un banc de test pour l'embarqué et l'IOT, dans le web, la data, etc...
- Les ressources des machines disponibles.
- La complexité des tests, plus ils sont nombreux et plus, ils sont longs à faire tourner et analyser.

Les plateformes les plus utilisées CI/CD

Résultat de l'enquête JetBrains 2023 sur le CI/CD[27]

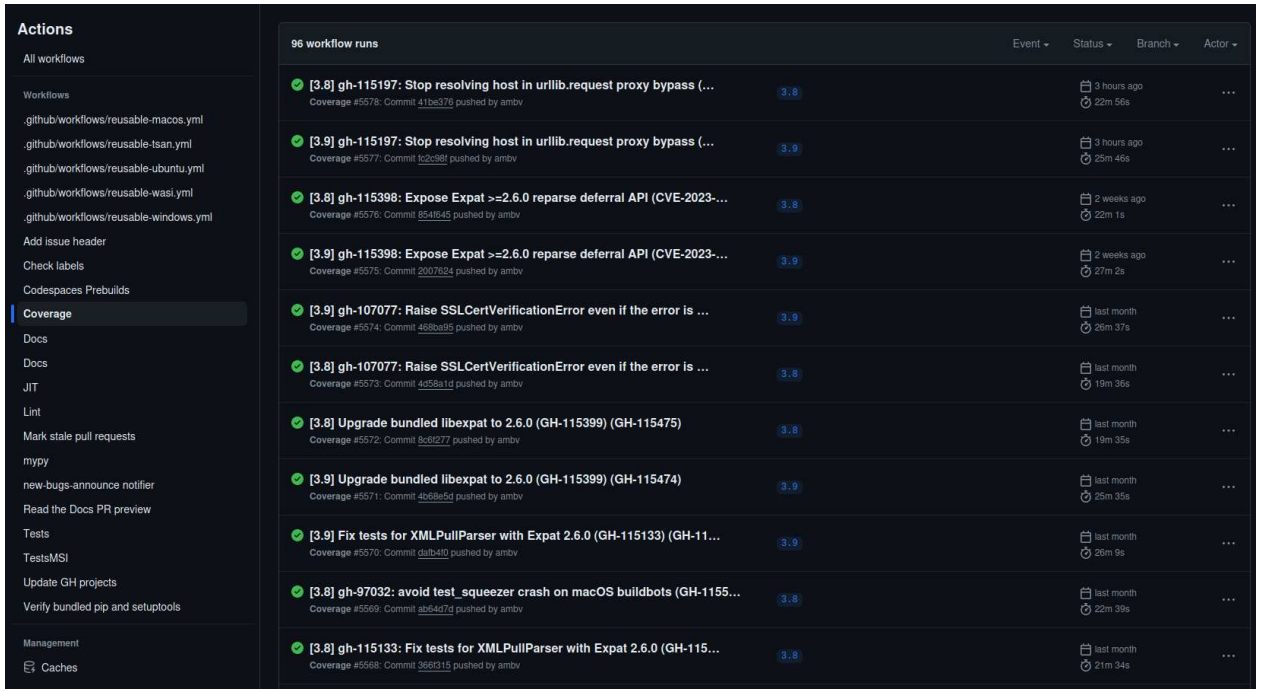


[27] Best Continuous Integration Tools for 2024 – Survey Results, https://blog.jetbrains.com/teamcity/2023/07/best-ci-tools/

Exemple de CI/CD

Exemple de CPython sur GitHub[28] Actions

Une partie seulement de CPython, l'implémentation classique officielle de Python est sur GitHub, il y en a aussi sur Azure DevOps. On y trouve 21 workflows différents !

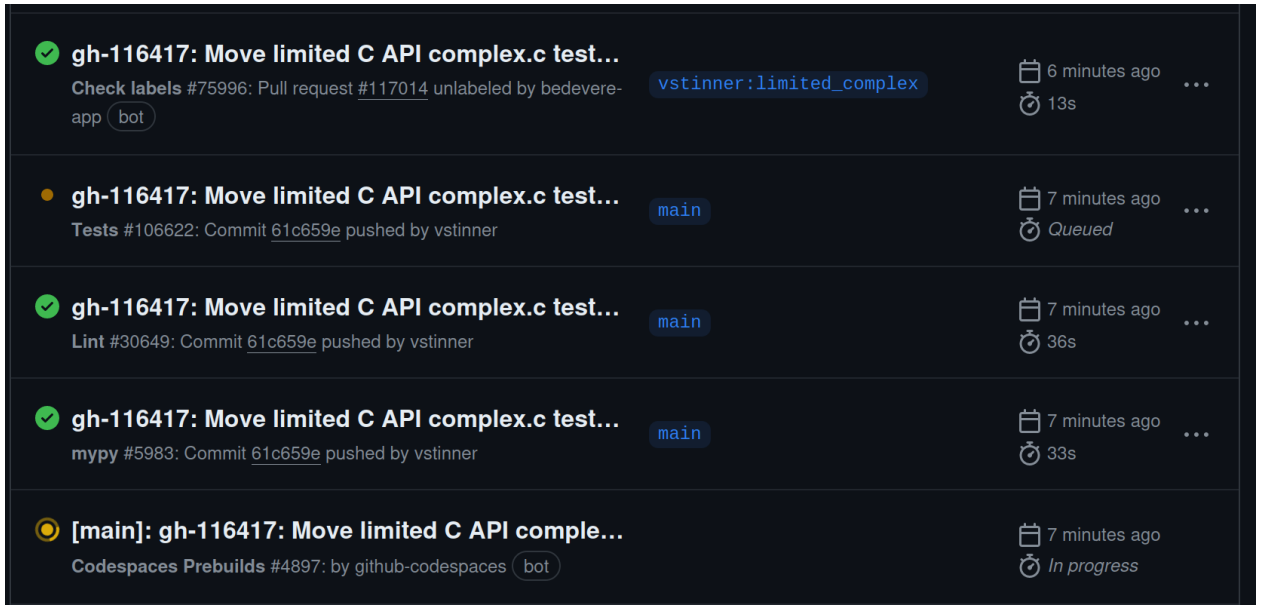


[28] CPython implémentation, https://github.com/python/cpython?tab=readme-ov-file

Exemple de CI/CD

Exemple de CPython sur GitHub[29] Actions

5 workflows “triggered” par un push et une PR !



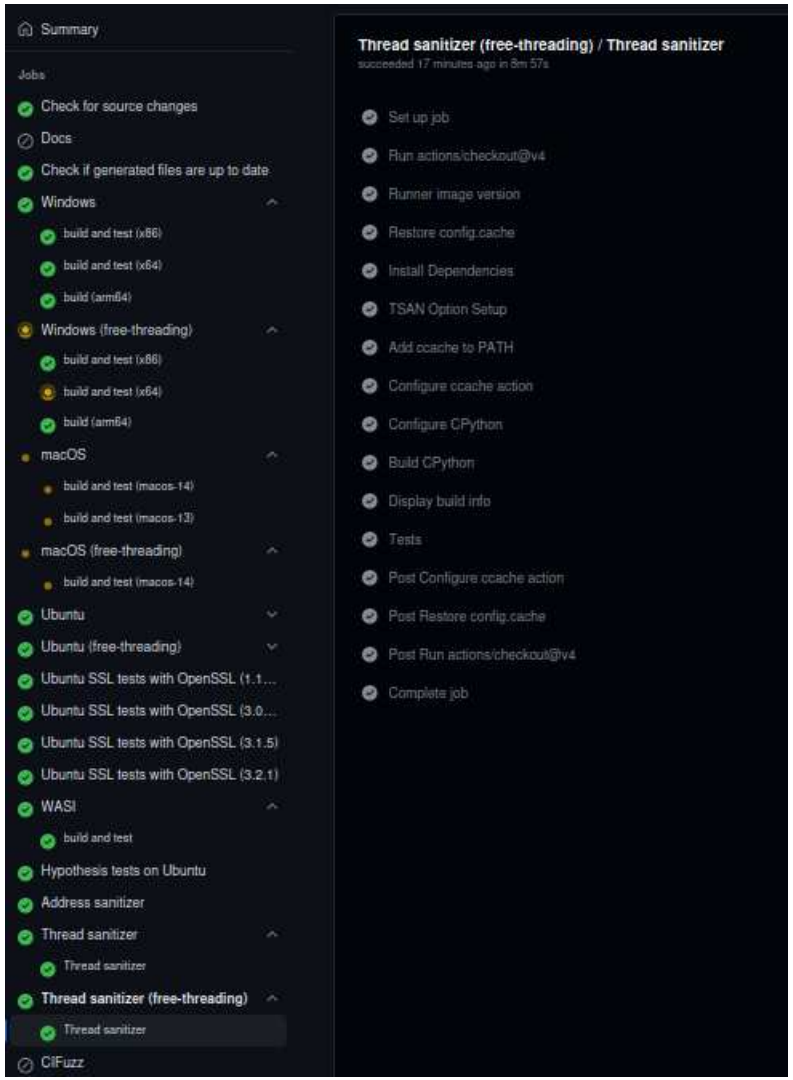
[29] CPython implémentation, https://github.com/python/cpython?tab=readme-ov-file

Exemple de CI/CD

Exemple de CPython sur GitHub[30] Actions

1 seul workflow compte des dizaines de jobs !

1 seul job compte plusieurs steps !



[30] CPython implémentation, https://github.com/python/cpython?tab=readme-ov-file

Exemple de CI/CD

Exemple de worflow GitHub[31]

Definition :

- Un workflow est une suite de jobs.
- Un job est une tâche à réaliser.
- Un step est une étape de la tache.
- Un artefact est un fichier généré par un step.

[31] Automate your workflow from idea to production, https://github.com/features/actions

Exemple de CI/CD

Exemple de worflow GitHub de ce repository, exercice de 15 minutes

Que peut-on faire comme workflow, GitHub Actions, etc, sur les dépôts de ce cours ?

- https://github.com/St-Michel-IT/qualite-code-source/
- https://github.com/St-Michel-IT/testing/

Exemple de CI/CD

Exemple de worflow GitHub de ce repository[32]

Les workflows sont dans les fichiers suivants :

```
$ tree .github/
.github/
--- workflows
    --- build-document.yml
    --- checkmytex.yml
    --- shell-check.yml
```

- `build-document.yml` pour générer le document PDF que vous lisez.
- `shell-check.yml` pour vérifier la syntaxe des scripts Shell.
- `checkmytex.yml` pour lister les erreurs dans le fichier LaTex.

[32] Worflow de génération des documents à partir de Latex, https://github.com/St-Michel-IT/qualite-code-source/blob/master/.github/workflows/

Exemple de CI/CD

Exemple de worflow GitHub du repository testing[33]

Les workflows sont dans les fichiers suivants :

```
$ tree .github
.github
--- workflows
    --- build-document.yml
    --- checkmytex.yml
    --- pylint.yml
    --- shell-check.yml
    --- black-formatter.yml
    --- unit-tests.yml
```

- `unit-tests.yml` lance les tests avec pytest.
- `pylint.yml` un linter qui génère un rapport des erreurs et des warnings.
- `black-formatter.yml` Black formatter qui fait un check sans rien changer.

[33] Worflow de génération des documents à partir de Latex, https://github.com/St-Michel-IT/testing/blob/master/.github/workflows/

Exemple de CI/CD

Des runners mais pas de possibilité d'exécuter l'action directement

Tous les workflows doivent passer par GitHub Actions même s'ils sont exécutés sur la machine du développeur, configurée comme un runner. Il faut donc commiter chaque modification et attendre le résultat, c'est chronophage.

Quelles sont donc les bonnes pratiques pour éviter les pièges des plateformes comme GitHub, Circle CI, Jenkins, etc ?

[34] Accelerate developer productivity, https://gradle.org/

Exemple de CI/CD

Des runners mais pas de possibilité d'exécuter l'action directement

Tous les workflows doivent passer par GitHub Actions même s'ils sont exécutés sur la machine du développeur, configurée comme un runner. Il faut donc commiter chaque modification et attendre le résultat, c'est chronophage.

Quelles sont donc les bonnes pratiques pour éviter les pièges des plateformes comme GitHub, Circle CI, Jenkins, etc ?

- Éviter des fonctionnalités propres à ces plateformes. C'est à dire agnostiques aux plateformes.
- Écrire des scripts qui peuvent être exécutés en local. Par exemple (BASH, PowerShell, Gradle[34], Dockerfiles, Ansible)
- Ces scripts peuvent-être appelés dans les jobs du CI/CD .
- Nettoyer vos environnements après le build pour économiser les ressources.

[34] Accelerate developer productivity, https://gradle.org/

Gradle

Gradle comme une passerelle entre le local et la plateforme de CI/CD

Gradle[34] est un outil de build open source. On script en Groovy[35] ou Kotlin[36], en local des taches de build, équivalentes aux jobs des plateformes de CI/CD .

On débug en local, on teste en local, on commit en local. Cela fait gagner un temps considérable (pas de commit/push...) dans le développement du CI/CD . Et donc, cela fait également économiser des ressources de CI/CD .

> Toutes les plateformes de CI/CD ont un plugin/workflow Gradle. La portabilité est donc assurée.

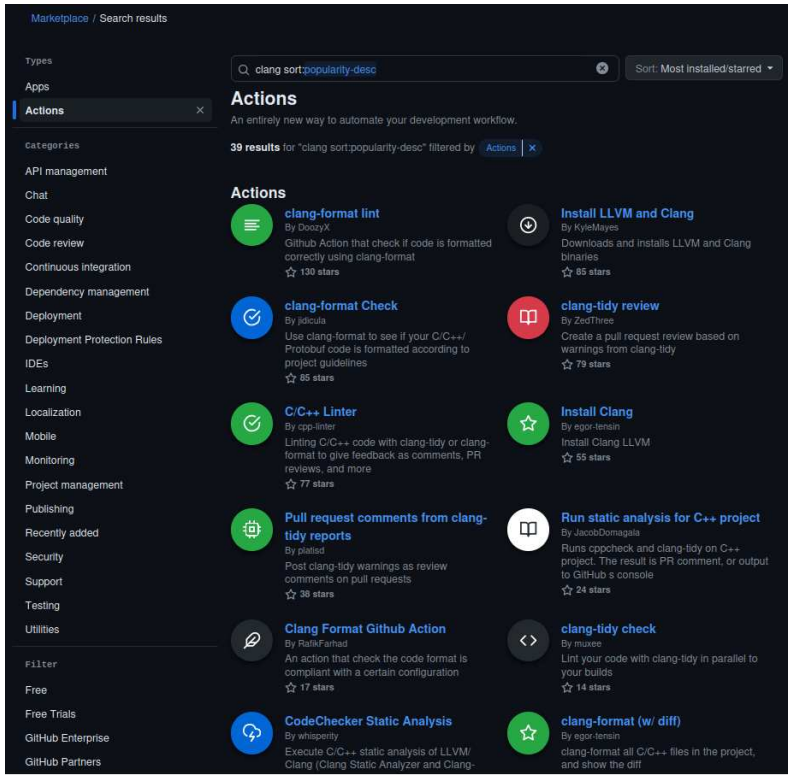
[35] Groovy, A multi-faceted language for the Java platform, https://groovy-lang.org/
[36] Kotlin, https://kotlinlang.org/

Les marketplaces des plateformes de CI/CD

Un “market” gratuit ou payant

Ne pas avoir peur du nom marketplace, beaucoup de plugins Jenkins ou de workflows GitHub sont gratuits.

- Plus de 1 900 plugins Jenkins[37].
- Plus de 22 000 actions GitHub[38].



Ils sont essentiels pour développer plus rapidement les jobs (reuse, comme dans n’importe quel code).

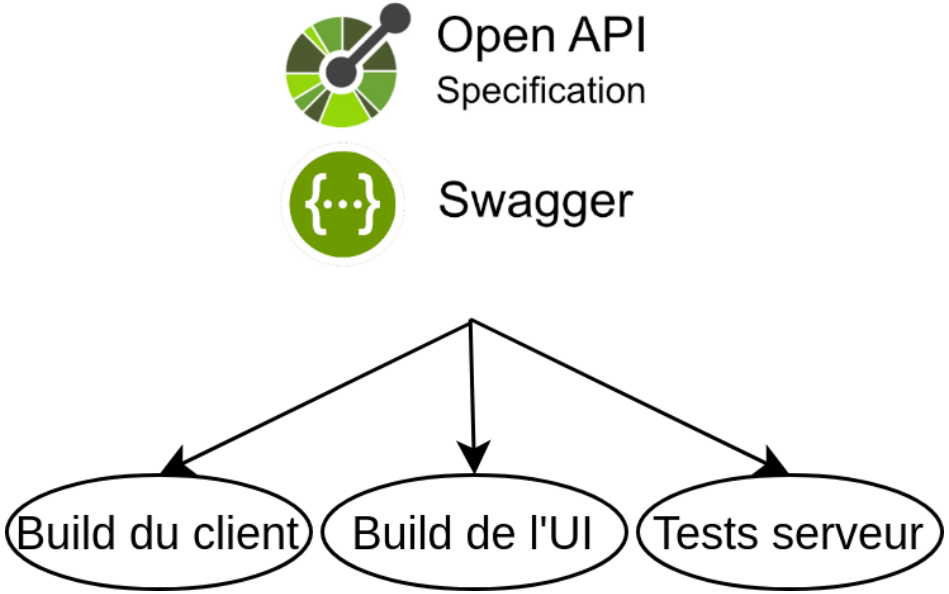
[37] Jenkins Plugins, https://plugins.jenkins.io/

[38] GitHub Actions, https://github.com/marketplace?type=actions

Les exemples de CI/CD dans le développement d'API

Grâce à Swagger et OpenAPI

Swagger est un fichier de définition d'une API . Il est écrit en YAML ou en JSON. Et les données pour décrire l'API sont dans la OpenAPI Specification[39].



[39] API Development for Everyone https://swagger.io/

Les exemples de CI/CD dans le développement d'API

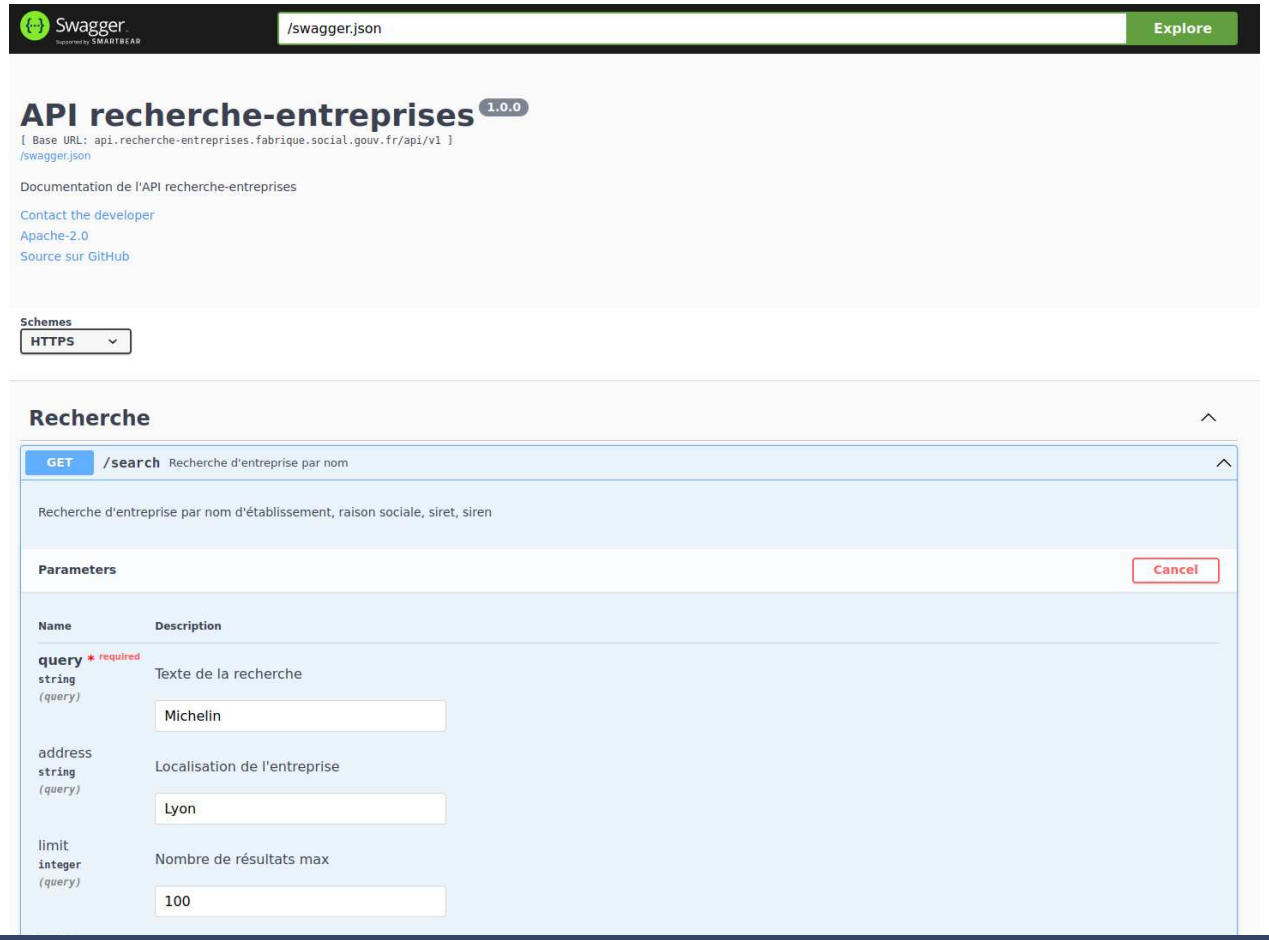
Grâce à Swagger et OpenAPI

```
/lists:
   get:
     summary: Gets all available data lists.
     description: Returns all custom data lists available to the current user.
     tags:
       - Lists
     responses:
       200:
         description: 'OK'
         content:
           application/json:
             schema:
               type: array
               items:
                 # Reuse du schema de DataList !
                 $ref: '#/components/schemas/DataList'
       401:
         # Reuse des responses !
         $ref: '#/components/responses/Unauthorized'
       5XX:
         $ref: '#/components/responses/ServerError'
```

Les exemples de CI/CD dans le développement d'API

Une UI pour découvrir l'API

L'UI est un espace convivial pour découvrir l'API . Il fait aussi office de documentation. Par exemple https://api.recherche-entreprises.fabrique.social.gouv.fr/#/Recherche/search.



Les exemples de CI/CD dans le développement d'API

Toutes les fonctionnalités sont déjà dans la marketplace (ou les plugins pour Jenkins)

Marketplace / Search results

Types
- Apps
- Actions

Categories
- API management
- Chat
- Code quality
- Code review
- Continuous integration
- Dependency management
- Deployment
- Deployment Protection Rules
- IDEs
- Learning
- Localization
- Mobile
- Monitoring
- Project management
- Publishing
- Recently added
- Security
- Support
- Testing
- Utilities

Filter
- Free
- Free Trials
- GitHub Enterprise
- GitHub Partners

swagger sort:popularity-desc

Sort: Most installed/starred

Actions

An entirely new way to automate your development workflow.

23 results for "swagger sort:popularity-desc" filtered by Actions

Actions

rdme Sync to ReadMe
By readmeio
Creator verified by GitHub
Sync OpenAPI/Swagger (OAS) definitions and Markdown documentation from your GitHub repository to your ReadMe developer hub
89 stars

Swagger Editor Validator
By char0n
This GitHub Actions validates OpenAPI (OAS) definition file using Swagger Editor
43 stars

Swagger UI Action
By Legion2
Generate Swagger UI for API documentation
38 stars

OpenAPI Diff
By mvegter
A GitHub Action to identify differences between Swagger or OpenApi specifications
22 stars

swagger ui
By pjoc-team
Generate swagger ui by json files
9 stars

Validate Swagger and OpenAPI using Swagger CLI
By mbowman100
Access to the swagger-cli via a github action
7 stars

Swagger Codegen
By bjerkio
Run Swagger Codegen with Github Actions
5 stars

Sync Swagger to Postman
By salockhart
Syncs a Swagger 2.0 file to a Postman collection
4 stars

Open API Specification Lint action
By vaibhav-jain
Runs Swagger validation on your OAS JSON or YAML file. Supports Swagger 2.0 or OpenAPI 3.0 format
3 stars

api-client-gen
By ajupov
Generate API Clients by Swagger
2 stars

Swagger-cli
By mpetrunic

OpenApi Specification Lint action
By crudo

Les exemples de CI/CD avec le déploiement d'une image Docker

GitHub ou Docker Hub peuvent héberger des images Docker[40]

```
    - name: Log in to Docker Hub
      uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}
    - name: Extract metadata (tags, labels) for Docker
      id: meta
      uses: docker/metadata-action@9ec57ed1fcdbf14dcef7dfbe97b2010124a938b7
      with:
        images: my-docker-hub-namespace/my-docker-hub-repository
    - name: Build and push Docker image
      uses: docker/build-push-action@3b5e8027fcad23fda98b2e3ac259d8d67585f671
      with:
        context: .
        file: ./Dockerfile
        push: true
        tags: ${{ steps.meta.outputs.tags }}
        labels: ${{ steps.meta.outputs.labels }}
```

[40] Publishing Docker images, https://docs.github.com/en/actions/publishing-packages/publishing-docker-images

Les exemples de CI/CD avec la publication de builds

Pour publier un soft compilé ou packagé sur sa page du dépôt GitHub[41]

```
name: Releases

on:
  push:
    tags:
    - '*'

jobs:

  build:
    runs-on: ubuntu-latest
    permissions:
      contents: write
    steps:
    - uses: actions/checkout@v3
    - uses: ncipollo/release-action@v1
      with:
        artifacts: "release.tar.gz,foo/*.txt"
        bodyFile: "body.md"
```

[41] Create Release, https://github.com/marketplace/actions/create-release

Les exemples de CI/CD avec la publication de builds

Publication de VS Code packagé[42]

Releases | Tags | Find a release

2 weeks ago
justschen
1.87.2
863d258
Compare

February 2024 Recovery 2 Latest

The update addresses these issues.

For the complete release notes go to Updates on code.visualstudio.com.

Assets 2

Source code (zip) 2 weeks ago
Source code (tar.gz) 2 weeks ago

56 9 19 14 19 11 82 people reacted

2 weeks ago
justschen
1.87.1
1e790d7
Compare

February 2024 Recovery 1

The update addresses these issues.

For the complete release notes go to Updates on code.visualstudio.com.

Assets 2

60 13 26 26 22 15 87 people reacted

[42] microsoft/vscode, https://github.com/microsoft/vscode/releases

Quelques exemples d'étapes de compilation du CI/CD

Quelles types de jobs pour passer d'un code source à un exécutable?

Quelques exemples d'étapes de compilation du CI/CD

Quelles types de jobs pour passer d'un code source à un exécutable?

- Compilation de C/C++ en binaire avec GCC et Vagrant pour cibler plusieurs les architectures.
- Compilation de Python en bytecode.
- Compilation de Java en Jar.
- Packaging de NodeJS en NPM .
- Packaging de Python en Wheel.
- Compiler LaTex en PDF .
- etc...

Quelques notions de base de Jenkins

Les jobs VS les pipelines

Les jobs sont des taches unitaires.
Un job est plus simple, c'est principalement un ensemble de commandes de "build".
Ce sont des commandes shell, console ou PowerShell en fonction de l'environnement.

Les pipelines sont des enchainements de jobs. Le pipeline est écrit en Groovy[35] et mais le principe est similaire au fichier Gradle ou workflow GitHub[43].

[43] Pipeline Examples, https://www.jenkins.io/doc/pipeline/examples/

Pour être agnostique à une plateforme de CI/CD, on peut exécuter dans la plateforme de CI/CD un build Gradle.
Jenkins, comme les autres plateformes, permet évidement de faire cela dans les builds ou les pipelines[44].

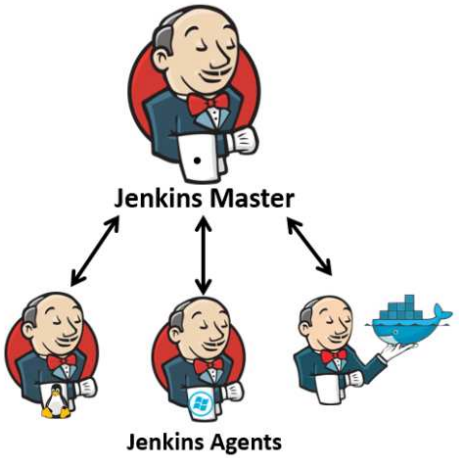
Si un maximum de taches, d'intelligence, de scripting sont dans le build Gradle, il sera plus simple de changer de plateforme de CI/CD .

[44] Jenkins Gradle plugin, https://plugins.jenkins.io/gradle/

Quelques notions de base de Jenkins

Les clusters de slaves et master[45]

Différentes machines esclaves peuvent être reliées à une machine maître. Soit pour répartir la charge, soit pour avoir des machines avec des environnements ou des configurations différentes. Ce sont des Jenkins slave agents aussi appelés node.



[45] Jenkins master-slave setup, https://www.linkedin.com/pulse/jenkins-slave-node-soma-sekhar-k-gusdc/

GitLab

Histoire et présentation de GitLab[46, 47]

GitLab est une plateforme de gestion de développement de logiciels au sens large. Son développement a commencé en 2011 .

Le nombre de ces fonctionnalités ne cesse d'augmenter mais elle est surtout connue comme VCS pour son intégration de Git.

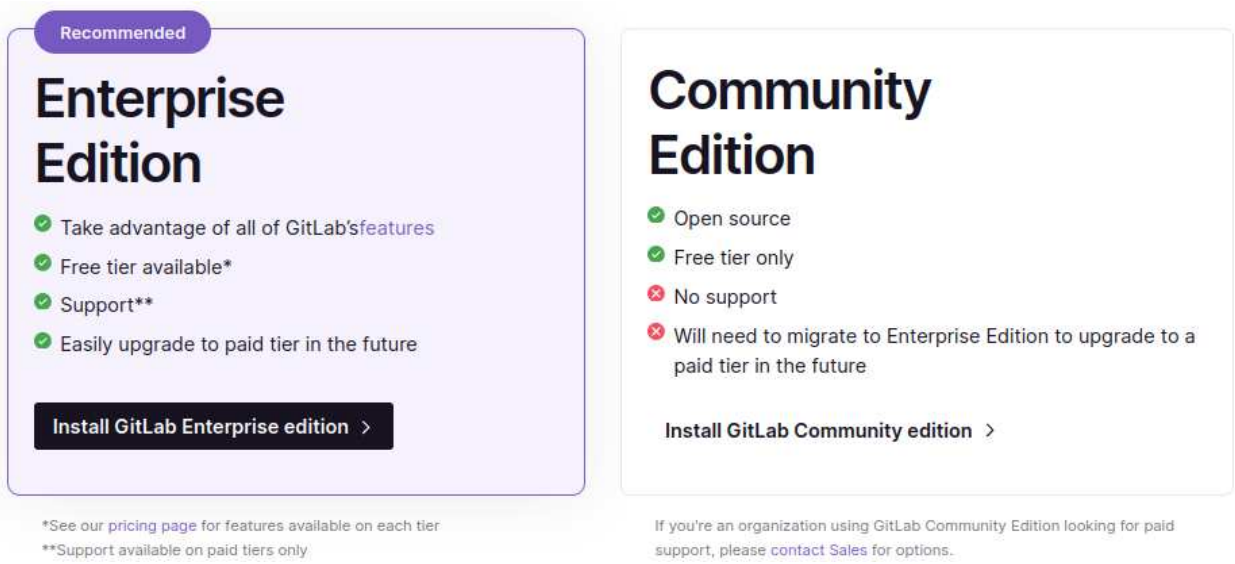
GitLab se tourne vers le CI/CD en 2018.

Mais on y trouve encore de nombreuses autres fonctionnalités :

- Gestion de projet
- Bug tracking
- WiKi
- GitLab Pages (déploiement de sites statiques)
- Many more...

[46] What we do, https://about.gitlab.com/company/
[47] GitLab, https://gitlab.com/st-michel/

Community Edition (CE) ou Entreprise Edition (EE) :



La version CE est à installer sur une de ses propres machines.
La version EE se décline en Premium et Ultimate.
Les fonctionnalités de ces 3 versions sont bien différentes, voir `https://about.gitlab.com/pricing/feature-comparison/`.
On ne paie donc pas que pour les ressources utilisées comme chez GitHub mais aussi pour les fonctionnalités.

GitLab

Le pipeline[48]

Très similaire aux workflows GitHub, c'est un ensemble de "Jobs" spécifiés dans un fichier YAML .

Il doit être à la racine du projet et se nommer `.gitlab-ci.yml`.

```
stages:
  - build
  - test

job_build:
  stage: build
  script:
    - echo "Building the project"

job_test:
  stage: test
  script:
    - echo "Running tests"
```

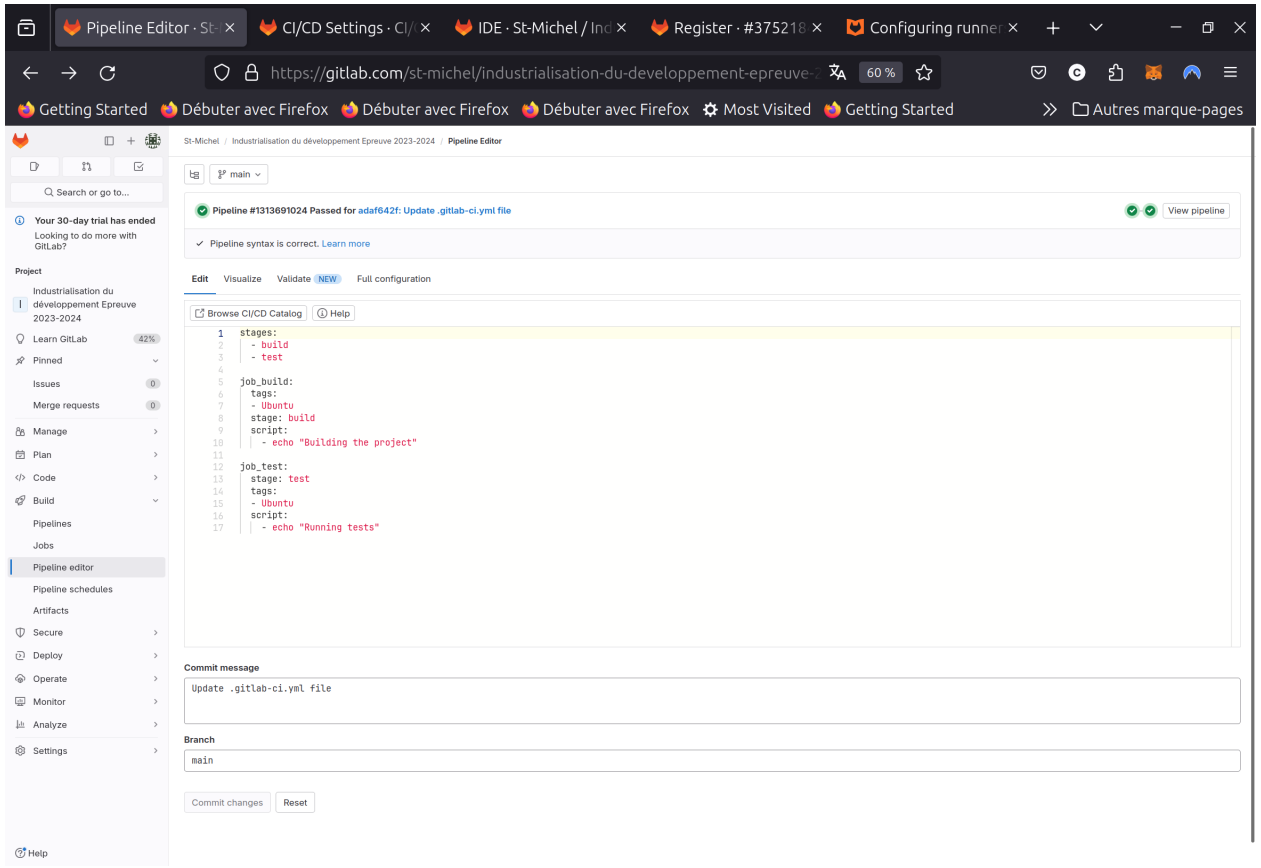
Ce Pipeline doit être exécuté dans un "runner".

[48] Tutorial: Create and run your first GitLab CI/CD pipeline, https://docs.gitlab.com/ee/ci/quick_start/

GitLab

Développement du pipeline dans la web UI de GiLab.com

La web UI de GitLab permet de développer le pipeline et de valider la syntaxe en live dans le “Pipeline editor” :



GitLab

Le runner[49]

Le “runner” est l’équivalent du Jenkins Slave.

Il peut fonctionner via de nombreuses interfaces :

- SSH
- Shell
- VirtualBox
- Docker
- Kubernetes
- ...

La machine peut être un PC, un serveur, un banc de test, Windows, Mac, Linux, etc.

Suivre le tutoriel officiel[49] pour configurer un runner et le lancer. Une fois que le programme `gitlab-runner run` est lancé, le runner est prêt à exécuter les pipelines.

À paramétrer comme service pour qu’il tourne en permanence en tâche de fond.

[49] Tutorial: Create, register, and run your own project runner, https://docs.gitlab.com/ee/tutorials/create_register_first_runner/index.html

GitLab

Exécution d'un pipeline sur un runner donné

Par défaut, si dans le pipeline, aucun runner n'est spécifié, le pipeline est exécuté sur un runner disponible et compatible sans que l'on puisse prédire lequel.

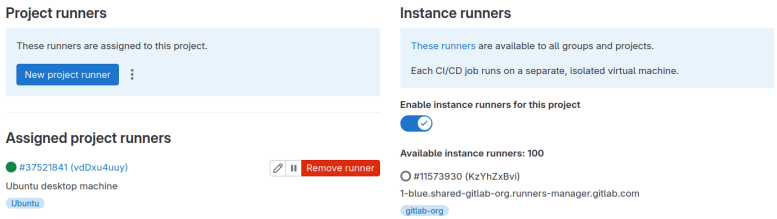
Lors de la configuration du runner, on peut lui donner de(s) tag(s).

Ce tag peut être spécifié dans les jobs du pipeline pour forcer l'exécution sur un/des runner(s) donné(s).

```
job_build:
  tags:
  - Ubuntu
```

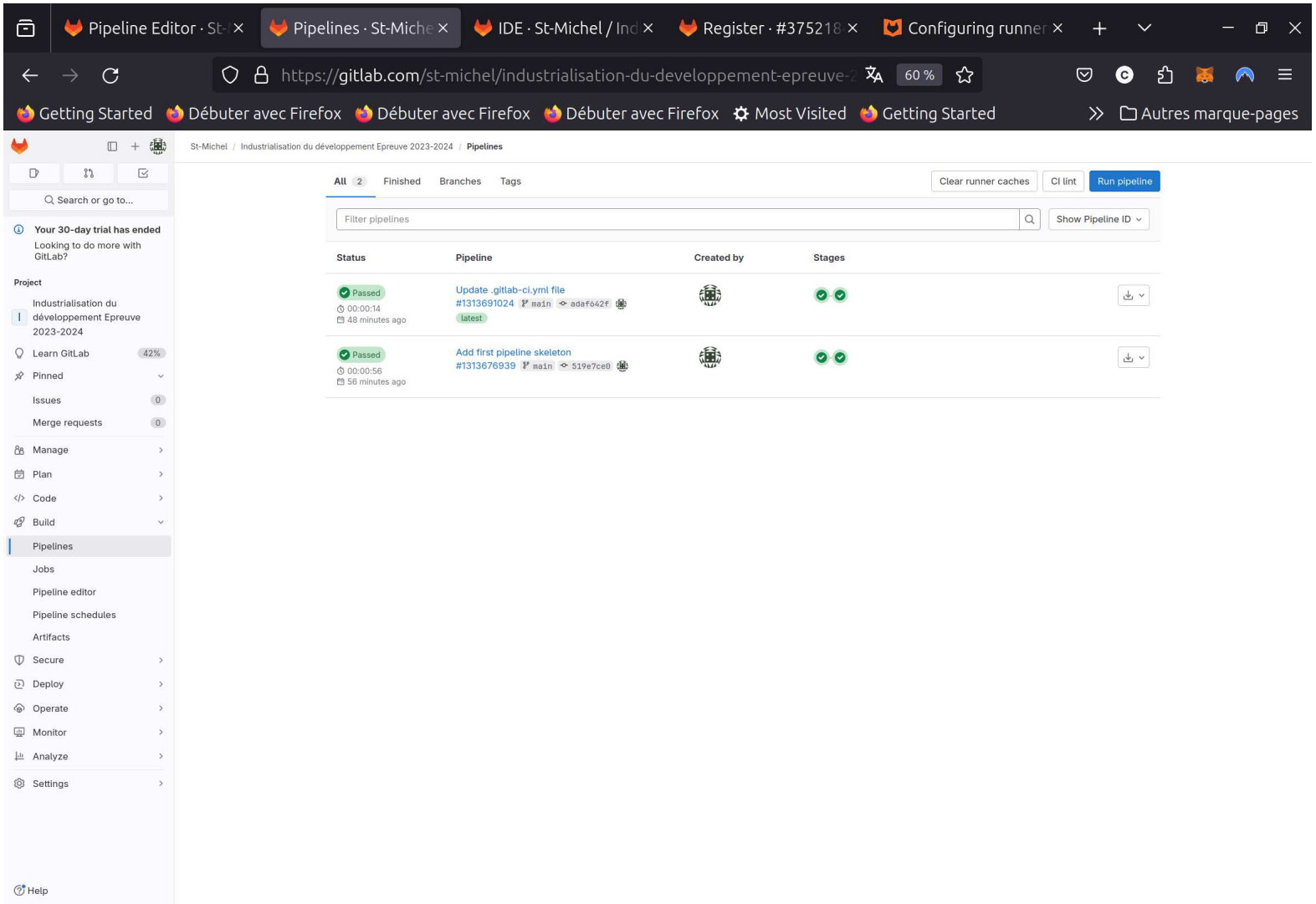
Ou :

```
job_build:
  tags:
  - Ubuntu
  - Docker
```

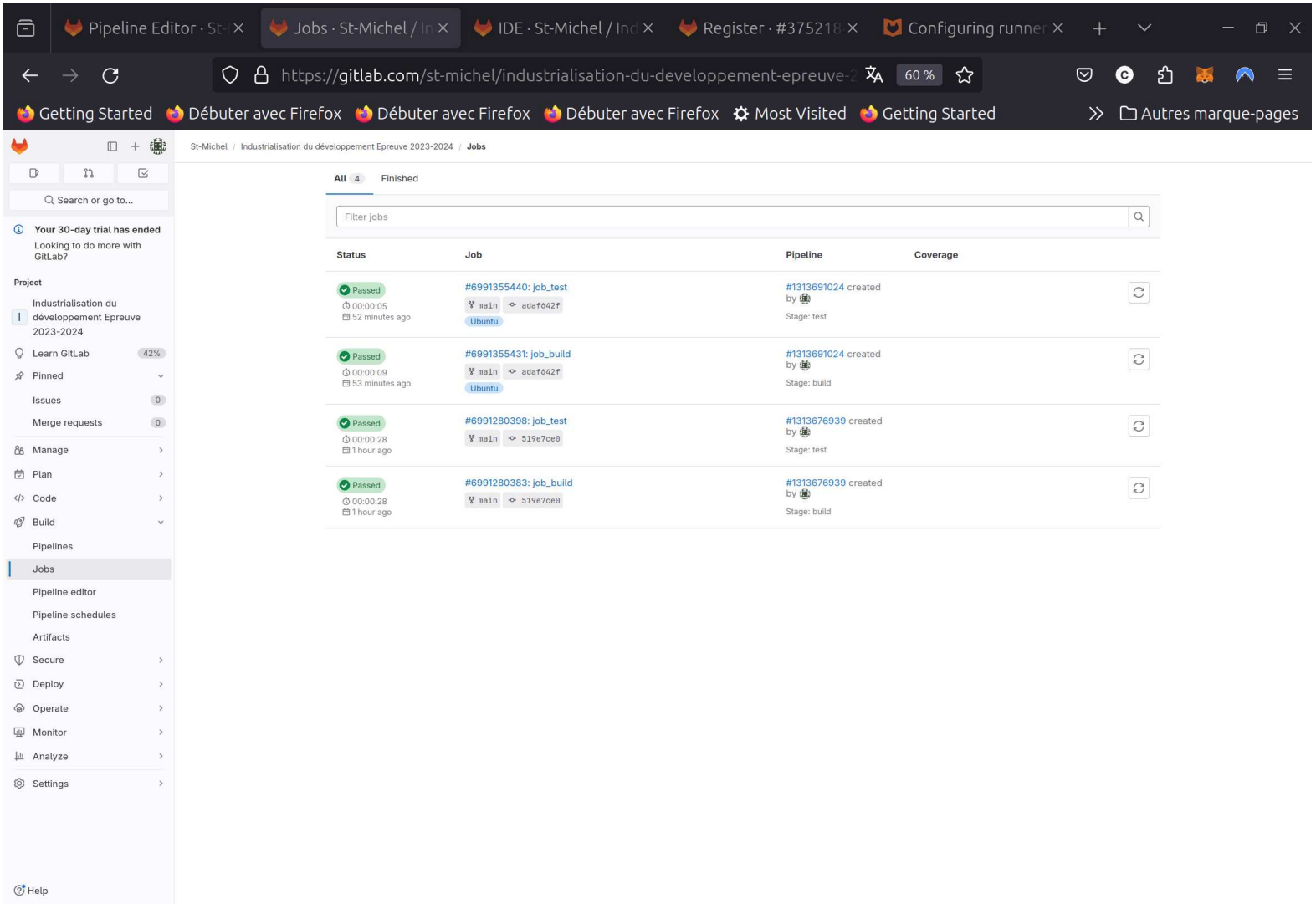


Une machine match avec le tag `Ubuntu`

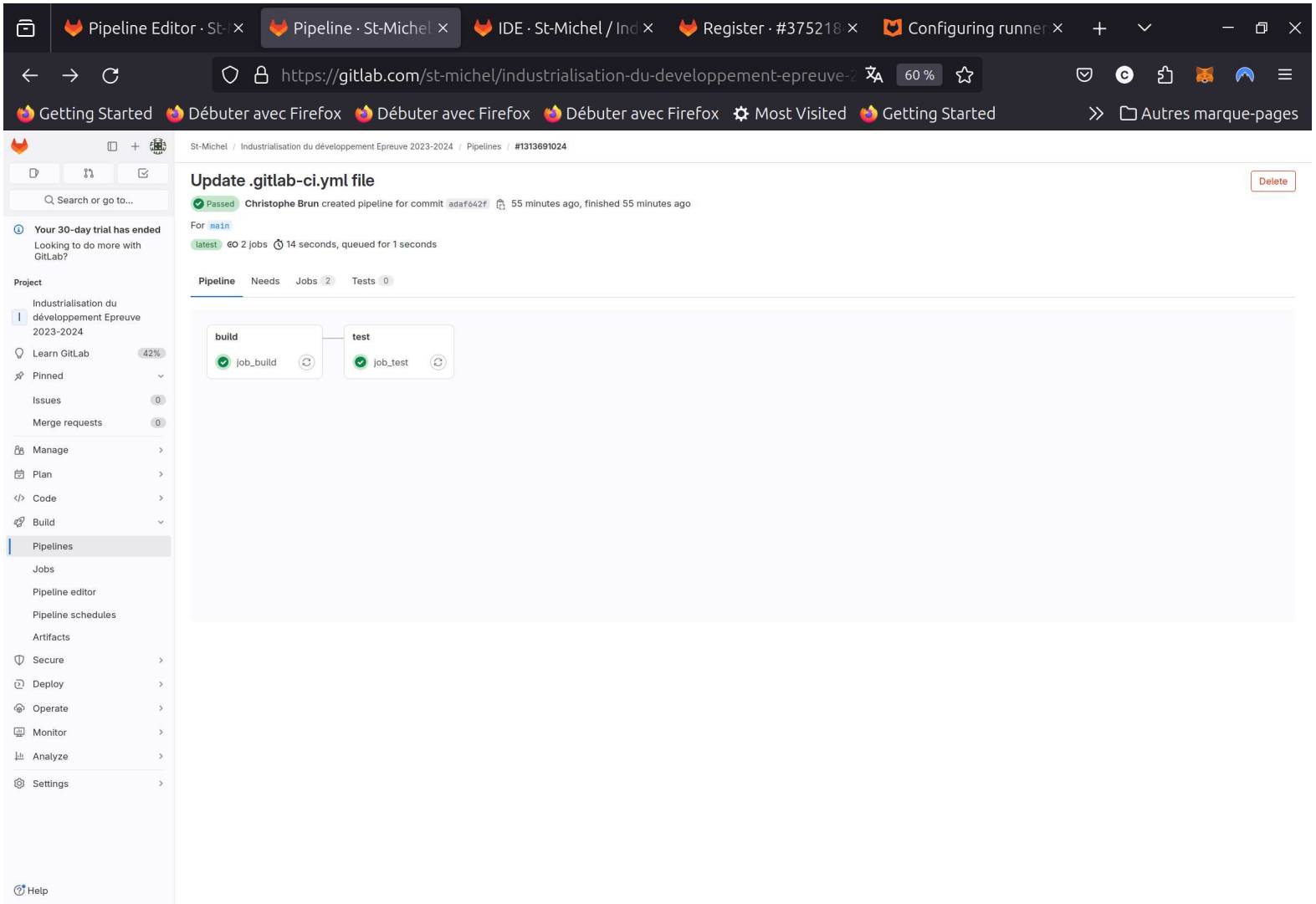
La liste des exécutions des pipelines est dans Build > Pipelines :



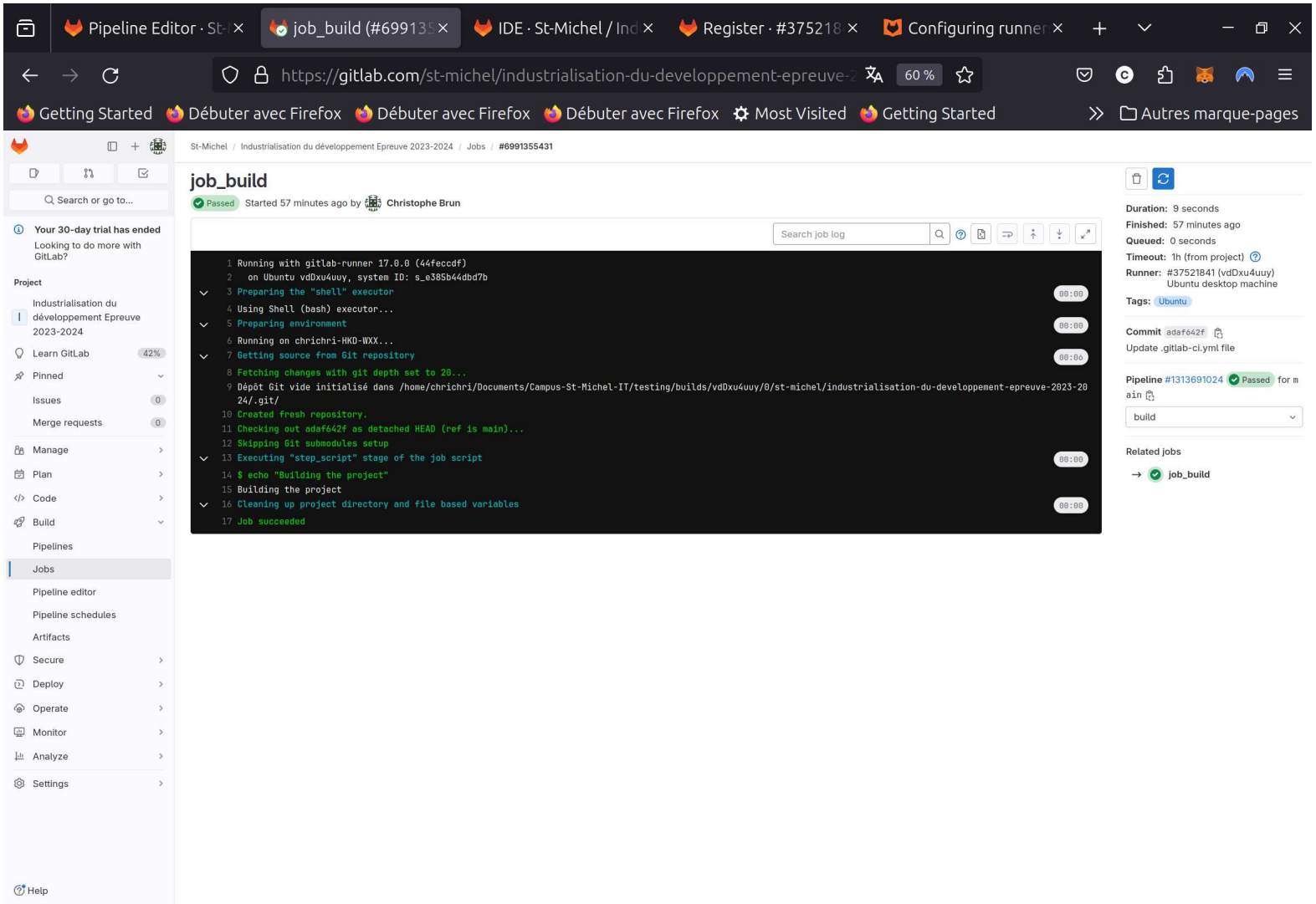
Il y a aussi une vue qui liste les Jobs est dans Build > Jobs :



Une vue détaillée des Jobs d'un Pipeline donné dans Build > Pipelines :



Une vue détaillée d'un Job avec les logs horodatées du runner :



Risques cyber lié au CI/CD

Quels sont les risques cyber lié au CI/CD ?

[50] Echoes of SolarWinds, https://www.scmagazine.com/news/echoes-of-solarwinds-jetbrains-teamcity-servers-under-attack-by-russia-bac

[51] The CircleCI secrets hack is a red flag for security teams on software supply chain risk, https://www.reversinglabs.com/blog/circleci-hack-is-a-red-flag-for-security-teams-on-the-software-supply-chai

Risques cyber lié au CI/CD

Quels sont les risques cyber lié au CI/CD ?

Une mauvaise configuration des droits, une faille non patchée, un secret exposé peuvent permettre d'accéder aux données du CI/CD . Donc probablement le code source.
Exemples de hacks majeurs liés au CI/CD :

- TeamCity non patché dans l'attaque de SolwarWinds[50]. Un fournisseur de services et logiciels de supervision de réseau. Les hackers ont ensuite exploité les failles de Solar Winds et accédé aux réseaux des clients comme MS qui a vu un source de Windows fuiter.
- Circle CI a vu des secrets exposés[51]. Le hack n'a pas touché ceux qui avaient mis en place Open ID Connect, la configuration est donc importante.

[50] Echoes of SolarWinds, https://www.scmagazine.com/news/echoes-of-solarwinds-jetbrains-teamcity-servers-under-attack-by-russia-bac

[51] The CircleCI secrets hack is a red flag for security teams on software supply chain risk, https://www.reversinglabs.com/blog/circleci-hack-is-a-red-flag-for-security-teams-on-the-software-supply-chai

Conclusions sur le CI/CD

- C'est un outil de plus pour améliorer la qualité des livrables (documentation, rapport de tests, librairies, exécutables, etc).
- Pour éliminer du bug.
- Un outil collaboratif pour partager les résultats dans l'équipe et l'entreprise.
- Faire d'abord en local puis exécuter dans la plateforme CI/CD .
- S'inspirer des outils qu'on a vus dans ou l'IDE ou en testing pour écrire des scripts.
- Attention aux ressources sur les machines locales.
- Attention aux coûts sur le cloud.

Stubbing et Mocking

Qu'est-ce que le stubbing et le mocking ?[52]

- Mock et Stub sont parfois appelés des "doubles" de test. Dans le sens où ils sont le double d'un objet, autre que le SUT ("Sytem Under Test").
- En plus de vérifier des données et des comportements du SUT. Le comportement attendu est également vérifié en testant l'interaction avec d'autres objets, ceux à mocker. Les objets mockés sont souvent appelés "collaborateurs" .
- Le stub, A.K.A. bouchon, est un objet qui simule un objet réel. Ses interactions avec le SUT doivent être le proche possible de celles de l'objet réel. Celles avec les autres objets peuvent ne pas exister pour plus de simplicité de développement du stub.
  Il peut être utilisé pour découpler les développements, on peut développer un client et le tester sans avoir le vrai serveur en face.

[52] Mocks Aren't Stubs, https://martinfowler.com/articles/mocksArentStubs.html

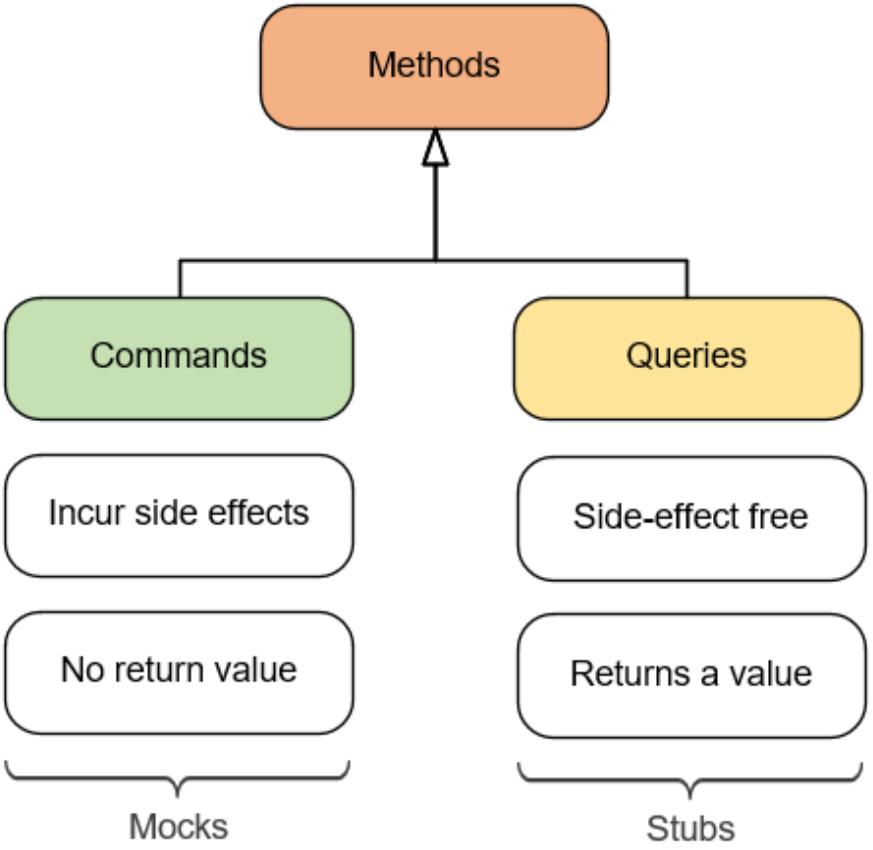
Stubbing et Mocking

Mock VS Stub[53]

S'il n'y a pas de valeur retournée, comme dans une commande par exemple, on se tourne évidemment vers les Mock. Pourquoi ❓

Les stubs ont pour but de retourner une valeur simulée, comme avec une requête.

Voir le pattern CQS...



[53] When to Mock, https://enterprisecraftsmanship.com/posts/when-to-mock/

Stubbing et Mocking

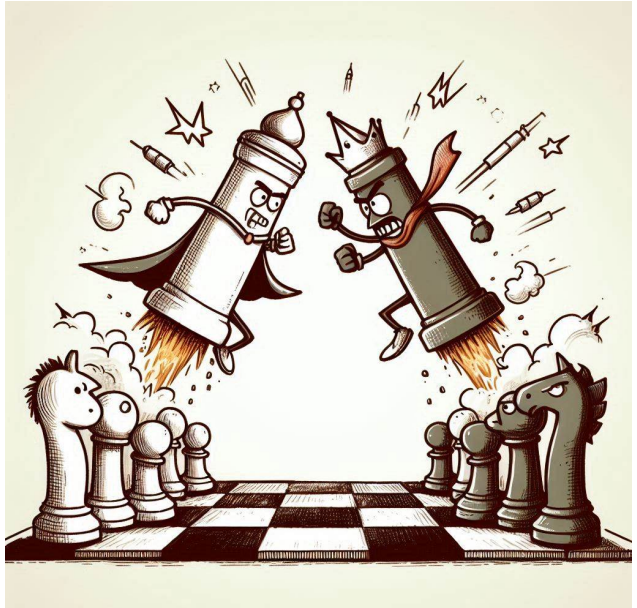
Mock VS Stub

Stub

- Get et Set des propriétés.
- Set des valeurs de retour.
- Test des états.

Mock

- Test les appels de fonctions du SUT au collaborateur.
- Test des arguments passés du SUT au collaborateur.
- Test des intéractions entre SUT et collaborateur.



Stubbing et Mocking

Classique, Mock ou Stub la conclusion de M. Fowler

Selon M. Fowler[52], c'est une histoire de compromis.
Il oppose l'approche du mocking et l'approche classique.
Il ne voit pas de bénéfice crucial au Mocking et reste sur sa manière classique de faire du TDD.

Le mot du formateur : Si vous ne savez pas choisir, ne le faites pas. L'univers des tests est infini et celui des bugs aussi. Écrire les deux voir trois types de tests, classiques et avec doubles pour réduire les bugs, isoler le code et le décorréler les développements (éviter le "tuilage").

Mocking

Equivalent mocké d'un test unitaire classique[52]

Le test unitaire classique :

```
public class OrderStateTester extends TestCase {
    private static String TALISKER = "Talisker";
    private static String HIGHLAND_PARK = "Highland Park";
    private Warehouse warehouse = new WarehouseImpl();

    protected void setUp() throws Exception {
        warehouse.add(TALISKER, 50);
        warehouse.add(HIGHLAND_PARK, 25);
    }
    public void testOrderIsFilledIfEnoughInWarehouse() {
        Order order = new Order(TALISKER, 50);
        order.fill(warehouse);
        assertTrue(order.isFilled());
        assertEquals(0, warehouse.getInventory(TALISKER));
    }
    public void testOrderDoesNotRemoveIfNotEnough() {
        Order order = new Order(TALISKER, 51);
        order.fill(warehouse);
        assertFalse(order.isFilled());
        assertEquals(50, warehouse.getInventory(TALISKER));
    }
}
```

Mocking

Equivalent mocké d'un test unitaire classique[52]

L'équivalent mocké' :

```
public class OrderInteractionTester extends MockObjectTestCase {
  private static String TALISKER = "Talisker";

  public void testFillingRemovesInventoryIfInStock() {
    Order order = new Order(TALISKER, 50);
    Mock warehouseMock = new Mock(Warehouse.class);
    warehouseMock.expects(once()).method("hasInventory")
      .with(eq(TALISKER),eq(50))
      .will(returnValue(true));
    warehouseMock.expects(once()).method("remove")
      .with(eq(TALISKER), eq(50))
      .after("hasInventory");
    order.fill((Warehouse) warehouseMock.proxy());
    warehouseMock.verify();
    assertTrue(order.isFilled());
  }

  public void testFillingDoesNotRemoveIfNotEnoughInStock() {
    Order order = new Order(TALISKER, 51);
    Mock warehouse = mock(Warehouse.class);
    warehouse.expects(once()).method("hasInventory")
      .withAnyArguments()
      .will(returnValue(false));
    order.fill((Warehouse) warehouse.proxy());
    assertFalse(order.isFilled());
  }
}
```

Mocking

Exercice 9, Redaction d'un test mocké

Le but est d'écrire un test mocké qui test le même SUT que `test_witness_number.py` pour venir le compléter.

- Identifier le SUT de ce test.
- Identifier une ou plusieurs libraire(s)/package(s) de mocking compatible(s) avec notre environnement de test.
- Ajouter une dépendance de mocking identifiée au projet.
- Identifier le collaborateur ou double à mocker.
- Rédiger le test mocké équivalent à `test_witness_number.py`.
- Jouer le test et produire le rapport.

Stubbing

Equivalent Stubbing d'un test unitaire classique[52]

Le stub ou bouchon, qui n'interagit pas avec l'extérieur :

```
public interface MailService {
  public void send (Message msg);
}

public class MailServiceStub implements MailService {
  private List<Message> messages = new ArrayList<Message>();
  public void send (Message msg) {
    messages.add(msg);
  }
  public int numberSent() {
    return messages.size();
  }
}
```

Le test unitaire bouchonné :

```
class OrderStateTester...
  public void testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    MailServiceStub mailer = new MailServiceStub();
    order.setMailer(mailer);
    order.fill(warehouse);
    assertEquals(1, mailer.numberSent());
  }
}
```

Stubbing

Exercice 10, Redaction d'un test bouchonné

Le but est d'écrire un test bouchonné qui test le même SUT que `test_customer_database.py` pour venir le compléter.

- Identifier le SUT de ce test.
- Identifier le(s) double(s).
- Développer le(s) bouchon(s).
- Rédiger le test bouchonné équivalent à `test_customer_database.py`.
- Jouer le test et produire le rapport.

Conclusion

"Write tests until fear is transformed into boredom"[54]

[54] Kent Beck, Test-Driven Development: By Example

Licence

Licence Creative Commons

Support de cours sous licence Creative Commons BY-NC-ND .

Vous pouvez donc, partager, copier, distribuer le document.

Attribution requise à PapIT SASU - Pas d'utilisation commerciale - Pas de modification