Langage et algorithmique

Rodéric Moitié

2018–2019

ENSTA Bretagne

Sommaire

Exemple : Ariane 501

- Vol de qualification du 4 juin 1996

Exemple : Ariane 501

- Vol de qualification du 4 juin 1996
- 36 premières secondes : vol normal

Exemple : Ariane 501

- Vol de qualification du 4 juin 1996
- 36 premières secondes : vol normal
- H+37s : erreur dans les SRI (système de référence inertielle) 1 et 2

Exemple : Ariane 501

- Vol de qualification du 4 juin 1996
- 36 premières secondes : vol normal
- H+37s : erreur dans les SRI (système de référence inertielle) 1 et 2
- Braquage des tuyères $\rightsquigarrow$ angle $> 20°$

Exemple : Ariane 501

- Vol de qualification du 4 juin 1996
- 36 premières secondes : vol normal
- H+37s : erreur dans les SRI (système de référence inertielle) 1 et 2
- Braquage des tuyères $\rightsquigarrow$ angle $> 20°$
- H+39s : autodestruction déclenchée automatiquement

Exemple : Ariane 501

- Vol de qualification du 4 juin 1996
- 36 premières secondes : vol normal
- H+37s : erreur dans les SRI (système de référence inertielle) 1 et 2
- Braquage des tuyères $\rightsquigarrow$ angle $> 20°$
- H+39s : autodestruction déclenchée automatiquement
- Coût : 500M€

Analyse de l'erreur du vol Ariane 501

- SR2 a transmis des données erronées

Analyse de l'erreur du vol Ariane 501

- SR2 a transmis des données erronées
- Raison : exception logicielle

Analyse de l'erreur du vol Ariane 501

- SR2 a transmis des données erronées
- Raison : exception logicielle
- Erreur lors de la conversion d'un flottant ($>$ 32768) en entier 16 bits

Analyse de l'erreur du vol Ariane 501

- SR2 a transmis des données erronées
- Raison : exception logicielle
- Erreur lors de la conversion d'un flottant (> 32768) en entier 16 bits
- Pas de protection de la conversion...

Analyse de l'erreur du vol Ariane 501

- SR2 a transmis des données erronées
- Raison : exception logicielle
- Erreur lors de la conversion d'un flottant (> 32768) en entier 16 bits
- Pas de protection de la conversion...

$\Rightarrow$ Valeur trop élevée dans la fonction de calcul de biais horizontal

Analyse de l'erreur du vol Ariane 501

- SR2 a transmis des données erronées
- Raison : exception logicielle
- Erreur lors de la conversion d'un flottant (> 32768) en entier 16 bits
- Pas de protection de la conversion...

$\Rightarrow$ Valeur trop élevée dans la fonction de calcul de biais horizontal

Raison

Trajectoires Ariane 4 et Ariane 5 différentes !

Analyse de l'échec

- Pas une erreur de programmation
  - Non protection de la variable justifiée par les marges de sécurité prises
- Pas une erreur de conception
  - Décision justifiée pour Ariane 4
- Réutilisation d'un composant d'Ariane 4 avec une contrainte cachée

$\Rightarrow$ précondition : $|BH| < 32768.0$

- Respectée pour Ariane 4, pas pour Ariane 5

Autres bugs ayant coûté cher

- 1985 : la machine **Therac-25** (traitement cancer) envoie une dose mortelle de radiations $\Rightarrow$ cause : overflow quand le technicien tapait trop vite, 5 décès

Autres bugs ayant coûté cher

- 1985 : la machine **Therac-25** (traitement cancer) envoie une dose mortelle de radiations $\Rightarrow$ cause : overflow quand le technicien tapait trop vite, 5 décès
- 1991 : des missiles **Patriot** ratent l'interception d'un Scud $\Rightarrow$ cause : bug dans l'arrondi de temps, 28 morts

Autres bugs ayant coûté cher

- 1985 : la machine **Therac-25** (traitement cancer) envoie une dose mortelle de radiations ⇒ cause : overflow quand le technicien tapait trop vite, 5 décès
- 1991 : des missiles **Patriot** ratent l'interception d'un Scud ⇒ cause : bug dans l'arrondi de temps, 28 morts
- 1999 : la sonde **Mars Climate Orbiter** entre trop bas dans l'atmosphère de Mars ⇒ cause : problème d'unités entre système métrique et système anglo-saxon, coût : $125M

Autres bugs ayant coûté cher

- 1985 : la machine **Therac-25** (traitement cancer) envoie une dose mortelle de radiations ⇒ cause : overflow quand le technicien tapait trop vite, 5 décès
- 1991 : des missiles **Patriot** ratent l'interception d'un Scud ⇒ cause : bug dans l'arrondi de temps, 28 morts
- 1999 : la sonde **Mars Climate Orbiter** entre trop bas dans l'atmosphère de Mars ⇒ cause : problème d'unités entre système métrique et système anglo-saxon, coût : $125M
- 1996-2013 : LOgiciel Unique à VOcation Interarmées de la Solde (**Louvois**). Logiciel mal conçu, nombreux bugs ⇒ calcul de solde fortement erroné. Prix achat + dysfonctionnement $\simeq$ 470M€.

Autres bugs ayant coûté cher

- 1985 : la machine **Therac-25** (traitement cancer) envoie une dose mortelle de radiations ⇒ cause : overflow quand le technicien tapait trop vite, 5 décès
- 1991 : des missiles **Patriot** ratent l'interception d'un Scud ⇒ cause : bug dans l'arrondi de temps, 28 morts
- 1999 : la sonde **Mars Climate Orbiter** entre trop bas dans l'atmosphère de Mars ⇒ cause : problème d'unités entre système métrique et système anglo-saxon, coût : $125M
- 1996-2013 : LOgiciel Unique à VOcation Interarmées de la Solde (**Louvois**). Logiciel mal conçu, nombreux bugs ⇒ calcul de solde fortement erroné. Prix achat + dysfonctionnement $\simeq$ 470M€.
- 2017 : problème lors de la mise à jour du logiciel d'aiguillage de la SNCF. Gare Montparnasse bloquée.

Problématique

- On ne peut pas tout tester ni tout prouver formellement

$\Rightarrow$ choisir les cas intéressants

- **Dijkstra** : Program testing can be used to prove the presence of bugs, but never their absence.
- Prouver l'absence de bug : indécidable...
- On se contente d'augmenter la confiance dans le logiciel
- Test : 80% du coût total d'un projet

Différents tests

Échelles de test :

- Test unitaire
- Test d'intégration
- Test système

Types de tests :

- Test de non régression
- Test de montée en charge
- Test de robustesse

Différents tests

Échelles de test :

- Test unitaire
- Test d'intégration
- Test système

Types de tests :

- Test de non régression
- Test de montée en charge
- Test de robustesse

Sommaire

Test manuel

- Principe : on fait tout à la main...

Test manuel

- Principe : on fait tout à la main...
- Souvent pratiqué par les débutants (utilisation de `print`, ...)

Test manuel

- Principe : on fait tout à la main...
- Souvent pratiqué par les débutants (utilisation de `print`, ...)
- Peu fiable : obligation de lire le résultat

Test manuel

- Principe : on fait tout à la main...
- Souvent pratiqué par les débutants (utilisation de `print`, ...)
- Peu fiable : obligation de lire le résultat
- Difficile de relancer les tests

Test manuel

- Principe : on fait tout à la main...
- Souvent pratiqué par les débutants (utilisation de `print`, ...)
- Peu fiable : obligation de lire le résultat
- Difficile de relancer les tests
- Pas d'automatisation

Test manuel

- Principe : on fait tout à la main...
- Souvent pratiqué par les débutants (utilisation de `print`, ...)
- Peu fiable : obligation de lire le résultat
- Difficile de relancer les tests
- Pas d'automatisation
- Ne laisse pas de trace

Test automatisé

Objectifs :

- Lisibilité des résultats de test
- Possibilité de relancer les tests simplement
- Possibilité de lancer les tests régulièrement
- Compte-rendu de test

Test unitaire en pratique

- Utiliser un framework de test (unittest)
- Cas de test = une méthode
  - configuration initiale
  - données de test
  - **oracle**
- Une classe de test par classe testée

Remarque

L'oracle a souvent besoin d'accéder à des données internes $\Rightarrow$ prévoir la « testabilité » du logiciel.

Tests de non régression

Objectif

Vérifier que l'ajout de nouvelles fonctionnalités n'altère pas les fonctionnalités existantes.

- Test à appliquer :
  - après refactoring ;
  - après ajout ou suppression de fonctionnalité ;
  - après une correction de bug.
- Caractéristiques :
  - test fastidieux ;
  - test à exécuter très souvent.

Couverture de code

Objectif

Vérifier que le code a été « suffisamment » testé

Plusieurs types de couverture :

- Couverture fonctionnelle : chaque fonction est exécutée

Couverture de code

Objectif

Vérifier que le code a été « suffisamment » testé

Plusieurs types de couverture :

- Couverture fonctionnelle : chaque fonction est exécutée
- Couverture structurelle : chaque instruction est exécutée

Couverture de code

Objectif

Vérifier que le code a été « suffisamment » testé

Plusieurs types de couverture :

- Couverture fonctionnelle : chaque fonction est exécutée
- Couverture structurelle : chaque instruction est exécutée
- Couverture niveau décision : chaque structure de contrôle est évaluée à VRAI et FAUX

Couverture de code

Objectif

Vérifier que le code a été « suffisamment » testé

Plusieurs types de couverture :

- Couverture fonctionnelle : chaque fonction est exécutée
- Couverture structurelle : chaque instruction est exécutée
- Couverture niveau décision : chaque structure de contrôle est évaluée à VRAI et FAUX
- Couverture niveau condition : chaque sous-expression booléenne est évaluée à VRAI et FAUX

Couverture de code

Objectif

Vérifier que le code a été « suffisamment » testé

Plusieurs types de couverture :

- Couverture fonctionnelle : chaque fonction est exécutée
- Couverture structurelle : chaque instruction est exécutée
- Couverture niveau décision : chaque structure de contrôle est évaluée à VRAI et FAUX
- Couverture niveau condition : chaque sous-expression booléenne est évaluée à VRAI et FAUX
- Couverture de chemin : tous les chemins d'exécution possibles sont couverts

Exemple en avionique : DO-178C

Logiciel critique : 5 niveaux de criticité

- E Problème sans effet sur la sécurité du vol

Contraintes sur le logiciel

Aucune contrainte.

Exemple en avionique : DO-178C

Logiciel critique : 5 niveaux de criticité

- E Problème sans effet sur la sécurité du vol
- D Problème pouvant perturber la sécurité du vol

Contraintes sur le logiciel

Couverture fonctionnelle du code, justification de toute modification, documentation du code, assurance qualité.

Exemple en avionique : DO-178C

Logiciel critique : 5 niveaux de criticité

- E Problème sans effet sur la sécurité du vol
- D Problème pouvant perturber la sécurité du vol
- C Problème sérieux : dysfonctionnement des équipements vitaux de l'appareil

Contraintes sur le logiciel

Couverture structurelle du code, vérification formelle des exigences bas niveau.

Exemple en avionique : DO-178C

Logiciel critique : 5 niveaux de criticité

- E Problème sans effet sur la sécurité du vol
- D Problème pouvant perturber la sécurité du vol
- C Problème sérieux : dysfonctionnement des équipements vitaux de l'appareil
- B Problème majeur : dégâts sérieux/mort de quelques occupants

Contraintes sur le logiciel

Couverture niveau décision du code, développement et vérification par des équipes indépendantes.

Exemple en avionique : DO-178C

Logiciel critique : 5 niveaux de criticité

- E Problème sans effet sur la sécurité du vol
- D Problème pouvant perturber la sécurité du vol
- C Problème sérieux : dysfonctionnement des équipements vitaux de l'appareil
- B Problème majeur : dégâts sérieux/mort de quelques occupants
- A Problème catastrophique : crash de l'avion

Contraintes sur le logiciel

Couverture de code niveau condition.

Analyse de mutation

Objectif

Mesurer l'efficacité des tests

Principe :

- Introduire des erreurs intentionnellement (injection de faute)

Analyse de mutation

Objectif

Mesurer l'efficacité des tests

Principe :

- Introduire des erreurs intentionnellement (injection de faute)
- Évalue la proportion de fautes détectées

Analyse de mutation

Objectif

Mesurer l'efficacité des tests

Principe :

- Introduire des erreurs intentionnellement (injection de faute)
- Évalue la proportion de fautes détectées
- Passer les tests, et examiner les mutants vivants

Analyse de mutation

Objectif

Mesurer l'efficacité des tests

Principe :

- Introduire des erreurs intentionnellement (injection de faute)
- Évalue la proportion de fautes détectées
- Passer les tests, et examiner les mutants vivants
  - Mutant équivalent à l'original : supprimer manuellement

Analyse de mutation

Objectif

Mesurer l'efficacité des tests

Principe :

- Introduire des erreurs intentionnellement (injection de faute)
- Évalue la proportion de fautes détectées
- Passer les tests, et examiner les mutants vivants
  - Mutant équivalent à l'original : supprimer manuellement
  - Mutant non équivalent : cas de test insuffisants

Analyse de mutation

Objectif

Mesurer l'efficacité des tests

Principe :

- Introduire des erreurs intentionnellement (injection de faute)
- Évalue la proportion de fautes détectées
- Passer les tests, et examiner les mutants vivants
  - Mutant équivalent à l'original : supprimer manuellement
  - Mutant non équivalent : cas de test insuffisants
- Aide : génération automatique de mutants

Analyse de mutation

Objectif

Mesurer l'efficacité des tests

Principe :

- Introduire des erreurs intentionnellement (injection de faute)
- Évalue la proportion de fautes détectées
- Passer les tests, et examiner les mutants vivants
  - Mutant équivalent à l'original : supprimer manuellement
  - Mutant non équivalent : cas de test insuffisants
- Aide : génération automatique de mutants
  - Opérateurs de mutation (remplacer + par -, modifier un opérateur logique, modifier les opérateurs relationnels, supprimer des instructions, perturber les constantes)

Sommaire

Python et unittest

- Framework de test

$\Rightarrow$ cadre simplifiant le test

- Objectifs :
  - Test d'applications
  - Facilite la création de tests
  - Facilite l'exécution de tests
  - Permet les tests de non régression
  - Test de couverture de code

Écriture d'un cas de test

Principe

- Cas de test : classe héritant de `unittest.TestCase`
- Une méthode de test par méthode à tester
- Le nom de la méthode de test doit commencer par `test`
- Méthodes appelées automatiquement par `unittest.main()`
- Utilisation d'assertions

Assertions

- `assertEqual(a, b)` — `assertNotEqual(a, b)`
- `assertTrue(x)` — `assertFalse(x)`
- `assertIs(a, b)` — `assertIsNot(a, b)`
- `assertIsNone(x)` — `assertIsNotNone(x)`
- `assertIn(a, b)` — `assertNotIn(a, b)`
- `assertIsInstance(a, b)` — `assertNotIsInstance(a, b)`
- `assertAlmostEqual(a, b)` — `assertNotAlmostEqual(a, b)`
- `assertGreater(a, b)` — `assertLess(a, b)`
- `assertRaises()`

Exemple de test unitaire

```
class Rationnel:
    def __init__(self, num, den=1):
        if den == 0:
            raise Exception('Invalid number')
        else:
            self.num = num
            self.den = den

    def __eq__(self, other):
        return self.num * other.den == other.num *
            self.den

    def __add__(self, other):
        n = self.num * other.den + self.den * other.
            num
        d = self.den * other.den
        return Rationnel(n, d)
```

Exemple de test unitaire

```
import unittest
from rationnel import Rationnel

class TestRationnel(unittest.TestCase):
    def testInit(self):
        r = Rationnel(1, -1)
        self.assertEqual(r.num, -1)
        self.assertEqual(r.den, 1)

    def testInitUndef(self):
        with self.assertRaises(Exception):
            r = Rationnel(1, 0)

    def testEq(self):
        r1 = Rationnel(-1, 3)
        r2 = Rationnel(2, -6)
        self.assertEqual(r1, r2)
```

Exemple de test unitaire

```
class TestRationnel(unittest.TestCase):
    def setUp(self):
        self.__two = Rationnel(2)

    def testAdd(self):
        r1 = Rationnel(1, 2)
        r2 = Rationnel(-1, 3)
        self.assertEqual(r1+r2, Rationnel(1, 6))

    def testAddMult(self):
        for num in range(-3, 4):
            for den in range(1, 3):
                r1 = Rationnel(num, den)
                r2 = r1 * self.__two
                self.assertEqual(r1 + r1, r2)

if __name__ == '__main__':
    unittest.main()
```

Exemple : exécution de test

Test OK

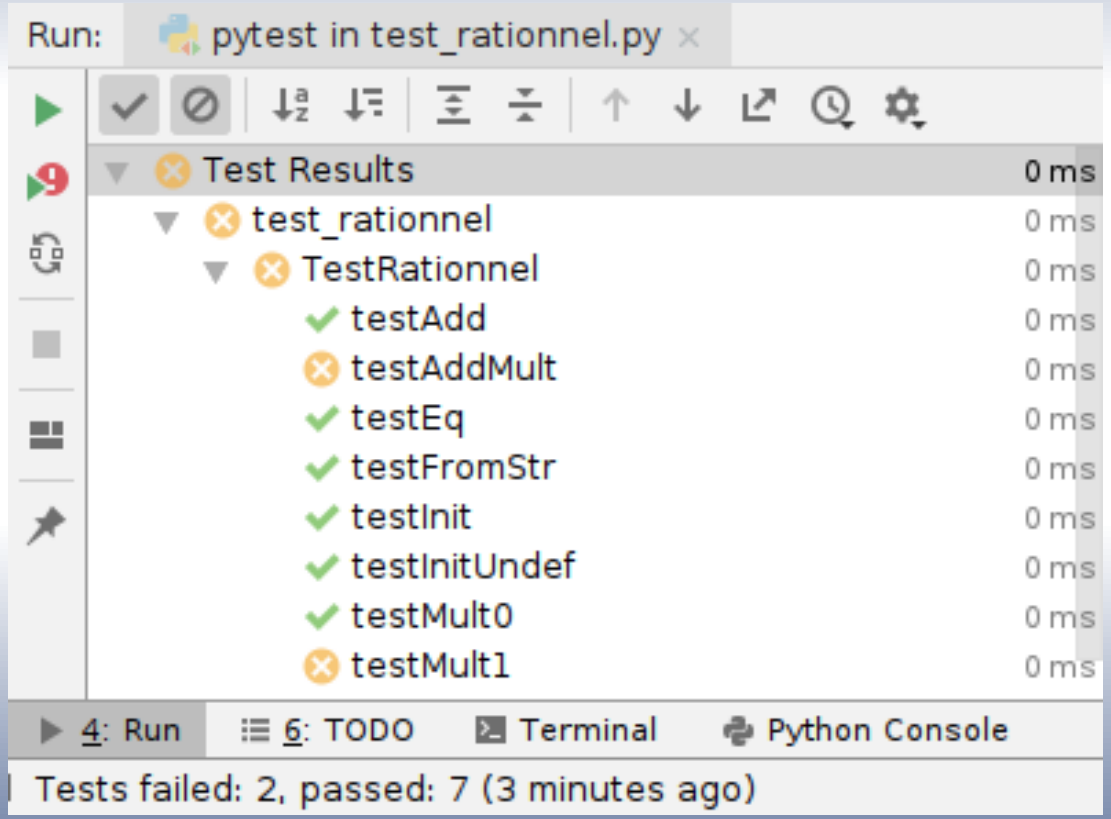
```
.......
----------------------------------------------
Ran 7 tests in 0.001s
OK
```

Test erreur

```
FF.....
FAIL: testAddMult (__main__.TestRationnel)
----------------------------------------------
Traceback (most recent call last):
  File "test_rationnel.py", line 55, in testAddMult
...
FAILED (failures=2)
```

Exemple : test dans Pycharm

Si le programme contient des fonctions dont le nom commence par `test` : exécute avec **pytest**



Couverture de code

```
$ coverage run test_rationnel.py
.......
----------------------------------------------------------------------
Ran 7 tests in 0.001s

OK
$ coverage report -m
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
rationnel               30      5    83%   53, 57-62
test_rationnel          46      0   100%
--------------------------------------------------
TOTAL                   76      5    93%
```

Sommaire

Objectif

Commentaire de programme

- Commentaires de documentation : """
- Commenter proprement le code
  - En-têtes
  - Classes
  - Fonctions
- Générer automatiquement la documentation
  - Création de pages HTML

Outils de génération de documentation

Sphinx

- Voir le poly annexe B
- `http://www.sphinx-doc.org/en/1.5.1/tutorial.html`

Pydoc

- Lire la page de manuel
- `pydoc3.5 -w fichier.py`
- `https://docs.python.org/3/library/pydoc.html`

Exemple : documentation de module

```
"""Example NumPy style docstrings.

This module demonstrates documentation as specified by
the `NumPy Documentation HOWTO`_. Docstrings may
extend over multiple lines. Sections are created with
a section header followed by an underline of equal
length.

Example
-------
Examples can be given using either the ``Example``
or ``Examples`` sections.

.. Source:
    http://sphinxcontrib-napoleon.readthedocs.io/en/
        latest/example_numpy.html
"""
```

Exemple : documentation de fonction

```
def function_with_types_in_docstring(param1, param2):
    """Example function with types documented in the
       docstring.

    Parameters
    ----------
    param1 : int
        The first parameter.
    param2 : str
        The second parameter.

    Returns
    -------
    bool
        True if successful, False otherwise.
    """
```

Exemple : documentation de classe

```
class ExampleClass(object):
    """The summary line for a class docstring should
    fit on one line.

    If the class has public attributes, they may be
    documented here in an ``Attributes`` section.
    Alternatively, attributes may be documented
    inline with the attribute's declaration.

    Attributes
    ----------
    attr1 : str
        Description of `attr1`.
    attr2 : :obj:`int`, optional
        Description of `attr2`.
    """
```