UMR IRISA

Eléments de **Génie Logiciel**
Cours 5
Introduction au test logiciel

Pr. Jean-Marc Jézéquel
IRISA – Université de Rennes
Campus de Beaulieu
F-35042 Rennes Cedex
e-mail : jezequel@irisa.fr
http://people.irisa.fr/Jean-Marc.Jezequel
X @jmjezequel

istic Informatique Électronique
ISTIC - L2GEN

Université de Rennes

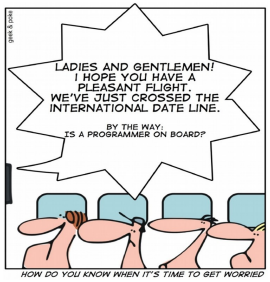
Université de Rennes

1 - Introduction

Qu'est-ce que le test et pourquoi tester?

Université de Rennes

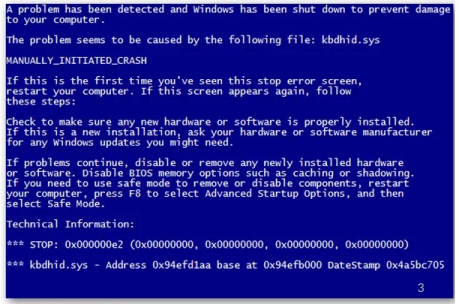
Ariane V (1996) :
Conversion d'un float 64-bit en int 16-bit



Mars Climate Orbiter (1998) :
Comparaison de valeurs dans des unités de mesures différentes

Nissan's Airbag Software Malfunction (2016)

Windows : the Blue Screen of Death



Université de Rennes

Vocabulaire

Note : Il est possible de remplacer « développeur » par « assistant automatique » (LLM/generative AI)

ChatGPT

- … une ***erreur*** du développeur introduit …
  - *Une erreur est une décision inappropriée ou erronée, faite par un développeur, qui conduit à l'introduction d'un défaut.*
- … un ***défaut*** dans le système qui provoquera …
  - *Un défaut est une imperfection dans un des aspects du système qui contribue, ou peut potentiellement contribuer, à la survenance d'une ou de plusieurs défaillances*
  - *Parfois, il faut plusieurs défauts pour causer une défaillance.*
- … sa ***défaillance*** à l'exécution.
  - *Une défaillance est un comportement inacceptable présenté par un système.*
  - *La fréquence des défaillances reflète la fiabilité.*

Vocabulaire

« Le test est un processus **manuel** ou **automatique**, qui vise à établir qu'un système **vérifie** les propriétés exigées par sa spécification, ou à **détecter** des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification »

Extrait de la norme IEEE-STD729, 1983

Le Test est le moyen le plus utilisé en pratique pour apporter de la **confiance** concernant le bon fonctionnement d'un logiciel

Le test ne certifie pas le bon fonctionnement

Améliorer la confiance
=
Améliorer la qualité des suites de tests

Tester pour apporter de la **confiance** concernant le bon fonctionnement du programme

- Fonctionnalité
- Sécurité / intégrité
- Utilisabilité
- Cohérence
- Maintenabilité
- Efficacité
- Robustesse
- Sûreté de fonctionnement
- etc.

Exemple

```
private Label label;
public Button createButton() {
  Button btn = new Button();
  EventHandler handler = event ->
label.setText("Hello World!");
  return btn;
}
```

Code correct *défectueux ?*

Spécification : un clic sur le bouton doit afficher « Hello World ! »

Défaillance : un clic sur le bouton ne produit aucun effet

Défaut : l'instruction `btn.setOnAction(handler)` est manquante

Erreur : le programmeur a oublié d'écrire `btn.setOnAction(handler)`

Test : un test (automatisé ou non) fait en sorte de cliquer sur le bouton et vérifie que « Hello World ! » est bien affiché

Université de Rennes

2 – Comment tester?

Université de Rennes

- Test statique
  - relecture / revue de code
  - analyse automatique
    - vérification de propriétés, règles de codage...

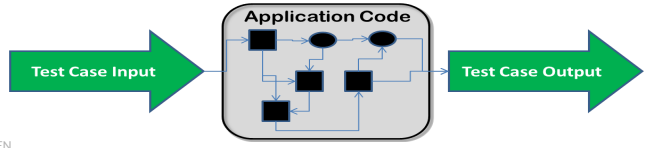
- Test dynamique
  - on exécute le programme avec des valeurs en entrée et on observe le comportement

Université de Rennes

Principes

- Test fonctionnel (**test boîte noire**)
  - Utilise la description des fonctionnalités du programme

- Test structurel (**test boîte blanche**)
  - Utilise la structure interne du programme



Université de Rennes

Test Dynamique

- **Statique** : le code n'est pas exécuté
- **Dynamique** : le code est exécuté et on analyse le résultat de son exécution
- Il existe de nombreuses techniques de test dynamique
  - Test Unitaire (TU)
  - Test d'intégration
  - Test système ou test E2E (end-to-end, de bout en bout)
  - Test d'acceptation
  - Test de montée en charge
  - ...

Test Dynamique

Cas de test

- Un cas de test = une méthode
- Corps de la méthode
  - Configuration initiale
    - Des données de test
  - Exécution du scénario
  - Un **oracle**
    - il faut construire le résultat attendu
    - ou vérifier des propriétés sur le résultat obtenu
- Une classe de test pour une classe testée
  - Regroupe les cas de test
  - Il peut y avoir plusieurs classes de test pour une classe testée

Test dynamique

Cas de test

- Un cas de test = une méthode
- Corps de la méthode
  - Configuration initiale
  - Exécution du scénario
  - Un oracle de test
    - Qui compare le résultat observé au résultat attendu

```
@Test
public void testRemoveRoue() {
    Roue roue = new Roue();
    Velo velo = new Velo();

    velo.addRoue(roue);
    velo.removeRoue(roue);

    assertTrue(velo.getRoues().isEmpty());
}
```


Test dynamique

- Concepts partagés dans la plupart des langages de programmation
- Java : **Junit** 4 ou 5
  - http://junit.org/
- Python :
  - **unittest** intégré dans la bibliothèque standard de Python
- C#
  - **MSTest** intégré dans l'écosystème de développement avec Visual Studio
- C++
  - GoogleTest (https://github.com/google/googletest)
  - CppUnit http://freedesktop.org/wiki/Software/cppunit/
  - Cpputest http://cpputest.github.io/

VS Code intègre ces outils de test

3 – Zoom sur JUnit

Test dynamique avec Junit 4

```
package test;

import static org.junit.Assert.*;
import org.junit.Test;

public class TestNombre {

  @Test
  void test() {
    fail("Not yet implemented");
  }
}
```

Package Java : **doit être le même que celui de la classe testée**

Importation de packages nécessaires

Définition de la classe de test : *TestNomClasse* Ou *NomClasseTest*

Annotations pour marquer la méthode comme une méthode de test

Code du test

Test dynamique avec JUnit

Code Java à tester dans dossier « src/fr/istic/monProjet »

```
package fr.istic.monProjet;

public class Nombre {
  int valeur;

  public Nombre(int val) {
    valeur = val;
  }

  public int multiplier(int val) {
    return valeur*val;
  }
}
```

Code de test dans le dossier « test/fr/istic/monProjet »

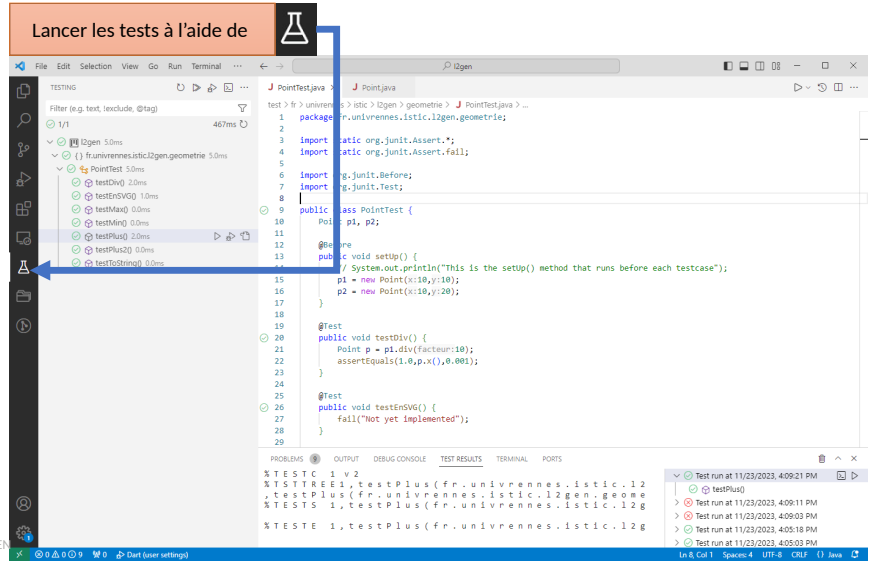
```
package fr.istic.monProjet;

import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class TestNombre {
  @Test
  void testMultiplier() {
    Nombre nb = new Nombre(2);
    assertEquals(6, nb.multiplier(3));
  }
}
```

Création d'un objet à tester

Vérification d'une propriété

Test dynamique avec Junit dans VS Code

Lancer les tests à l'aide de



Test dynamique avec JUnit

Les assertions

Permettre de vérifier des propriétés sur l'objet à tester

assertEquals(valAttendue, valObservée);	Vérifie que *valObservée* et *ValAttendue* sont égales
assertNull(obj);	Vérifie que *obj* est nul
assertNotNull(obj);	Vérifie que *obj* n'est pas nul
assertFalse(bool);	Vérifie que *bool* est faux
assertTrue(bool);	Vérifie que *bool* est vrai

...

Université de Rennes

Test dynamique avec JUnit

- L'annotation *@Test* identifie un test
- Pas de *main* : chaque opération annotée par *@Test* est exécutée comme un programme
- Un test contient des **assertions**
- Un test possède un **oracle**
  - Oracle = objet (personne, instruction, etc.) connaissant le résultat attendu du test
  - Les assertions définissent l'oracle d'un test, exemple :
    - *assertEquals*(6, nb.multiplier(3));

Université de Rennes

4 – Test Nominal vs Test de Robustesse

Test aux Limites

Université de Rennes

Test dynamique avec JUnit

On doit tester différents aspects :

- Le bon fonctionnement d'une opération en mode nominal
  - *On peut avoir plusieurs opérations de tests visant à couvrir l'ensemble des fonctionnalités*
- La bonne gestion des cas non standards :
  - *Valeurs non valides (erreur correctement gérée)*
  - *Très élevées*
  - *Bornes des valeurs*
  - *Etc.*

Université de Rennes

Question : Résultat du test?

```
double x = 999999999999999.;
double y = x + 1.0;
assertTrue(y - x == 1.0);
```

```
double x = 9999999999999999.;
double y = x + 1.0;
assertTrue(y - x == 1.0);
```

Pass

Fail

- Attention avec l'égalité entre nombres flottants
  - Pas que en Java!
  - Car c'est un problème de la norme IEEE 754 implémentée par les processeurs
- Java et Junit
  - v1==v2 => Double.*compare*(v1, v2) == 0
  - *assertEquals*(x, y, 0.0001);

Test dynamique avec JUnit

```
public class Nombre {
  double valeur;

  public Nombre(int val) {
    valeur = val;
  }

  public double multiplier(double val) {
    return valeur*val;
  }

  public double diviser(double v) throws IllegalArgumentException{
    if(Double.compare(v, 0.0) == 0)
      throw new IllegalArgumentException();
    return valeur/v;
  }
}
```

Exception quand divise par 0

Rappel exception Java

Exception = Erreur gérée dans le code
Le '*throws*' déclare les exceptions lancées par l'opération

```
public double diviser(double v) throws IllegalArgumentException {
  if(Double.compare(val, 0.0)==0)
    throw new IllegalArgumentException();
  return valeur/val;
}
```

Le '*throw*' lance une exception

'*try*' permet d'encadrer du code pouvant lever une exception

'*catch*' définit le code à exécuter en cas d'exception levée

```
try {
  nb.diviser(0);
}catch(IllegalArgumentException e) {
  e.printStackTrace();
}
```


Test dynamique avec JUnit

```
public class TestNombre {
  @Test
  void testMultiplier() {
    Nombre nb = new Nombre(2);
    assertEquals(6.0, nb.multiplier(3.0), 0.0001);
  }

  @Test
  void testDiviser() {
    Nombre nb = new Nombre(6);
    assertEquals(2.0, nb.diviser(3.0), 0.0001);
  }

  @Test
  void testDiviserPar0() throws IllegalArgumentException {
    Nombre nb = new Nombre(2);
    assertThrows(IllegalArgumentException.class, () -> nb.diviser(0));
  }
}
```

Plusieurs tests par classe

1 test = 1 scénario

Teste que division ok

Teste que division par 0 lève une exception

Test dynamique avec JUnit

JUnit permet de tester que des erreurs sont bien gérées

'assertThrows' permet de définir l'exception qui doit être levée

puis on donne l'exception

```
@Test
void testDiviserPar0() throws IllegalArgumentException {
  Nombre nb = new Nombre(2);
  assertThrows(IllegalArgumentException.class, () -> nb.diviser(0));
}
}
```

On provoque l'erreur

Université de Rennes

4 – Préludes et Postludes de Tests

Setup & Tear Down

Université de Rennes

Test dynamique avec JUnit

```
public class TestNombre {
 @Test
 void testMultiplier() {
   Nombre nb = new Nombre(6);
   assertEquals(2.0, nb.multiplier(12.0), 0.01);
 }

 @Test
 void testDiviser() {
   Nombre nb = new Nombre(6);
   assertEquals(2.0, nb.diviser(3.0), 0.01);
 }

 @Test
 void testDiviserPar0() throws IllegalArgumentException {
   Nombre nb = new Nombre(6);
   assertThrows(IllegalArgumentException.class, () -> nb.diviser(0));
 }
}
```

Souvent, initialisation identique dans différents tests : **Redondant**

Université de Rennes

Test dynamique avec JUnit

Test Fixture

```
public class TestNombre {
 Nombre nb;

 @Before
 void setUp() {
   nb = new Nombre(6);
 }

 @Test
 void testMultiplier() {
   assertEquals(18., nb.multiplier(3.), 0.);
 }

 @Test
 void testDiviser() {
   assertEquals(2., nb.diviser(3.), 0.);
 }

 @Test
 void testDiviserPar0() throws IllegalArgumentException {
   assertThrows(IllegalArgumentException.class, () -> nb.diviser(0));
 }
}
```

Déclaration d'1 attribut *nb*

L'annotation *@Before* permet de définir 1 opération appelée avant chaque test: **test fixture**
Utilisée pour initialiser des valeurs.

Initialisation de *nb*

Utilisation de *nb*

Université de Rennes

Tear Down

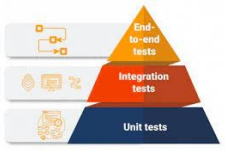
- Action à faire après chaque test
  - Par exemple relâcher une ressource, fermer un fichier…

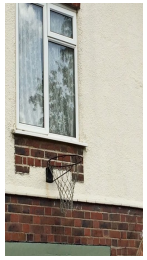
```
@After
 public void tearDown() {
      System.out.println("This is the tearDown() method that runs after each testcase");
}
```

Université de Rennes

Test dynamique

- JUnit permet de d'écrire différents types de tests dynamiques
  - Les slides précédents faisaient référence à du test unitaire
  - Test d'intégration
  - Test système (E2E)
  - Test d'acceptation
  - Test de robustesse
  - Test de performance
  - Test de non régression
    - *Archiver et automatiser les tests*
  - ...
- Mais ce sera pour plus tard…





Université de Rennes

5 – TDD : Test Driven Design

Développement dirigé par les tests

Université de Rennes

Principe du TDD

- Pour ajouter une fonctionnalité X à un logiciel:
  1. Ecrire les tests T montrant que X fonctionne
  2. Vérifier que le test échoue (X n'est pas encore développée)
     => Permet de vérifier que le test est valide!
  3. Ecrire le code nécessaire pour passer T (et pas plus)
  4. Vérifier que les tests T passent, sinon retourner en 3
  5. Si nécessaire, remanier le code pour le simplifier

Université de Rennes

Exercice TDD

- Ecrire les tests pour une fonction **`add(s:String):Int`** qui additionne les nombres contenus dans une chaîne de caractères.
  - pour une chaîne vide, la fonction rend 0
  - pour une chaîne contenant un seul nombre, la fonction rend ce nombre
  - pour une chaîne contenant deux nombres séparés par une virgule, la fonction rend la somme
  - etc.

```
@test
 public void testAdd0() {
      //…
}
@test
 public void testAdd1() {
      //…
}
```

Université de Rennes

TDD : Conclusion

- **Tests Simples et Rapides**
  - L'accent est mis sur la création de tests simples et rapides pour maintenir une suite de tests efficace.
- **Communication Améliorée**
  - Les tests servent également de documentation vivante, améliorant la communication au sein de l'équipe de développement.
- **Inconvénients :**
  - Peut être couteux si fait sans discernement
  - Peut entraîner une confiance excessive dans la stabilité du logiciel
    - certains problèmes peuvent subsister au niveau des interactions entre composants.

Vous le pratiquerez au prochain TP!