

# LES TESTS

Martine GAUTIER - Université de Lorraine  
martine.gautier@univ-lorraine.fr

# Un peu de réflexion sur les tests écrits en Tp

# Un peu de réflexion sur les tests écrits en Tp

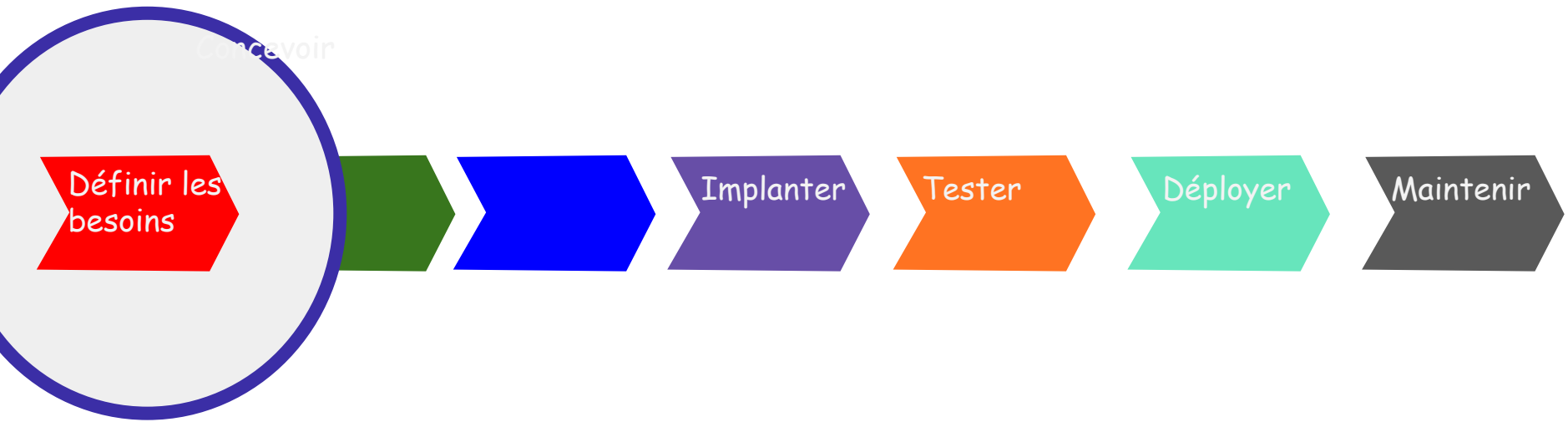
- Quelle différence entre Awaz et Iceberg2D ?
- Qu'est-ce-qu'un test ?
- L'affichage est correct, que puis-je en déduire ?
- L'affichage est incorrect, que puis-je en déduire ?
- La méthode d'écriture des tests est-elle généralisable ?
- L'effort à fournir est-il conséquent ?
- Les tests sont corrects, puis-je les jeter ?

# Les différentes étapes du développement logiciel



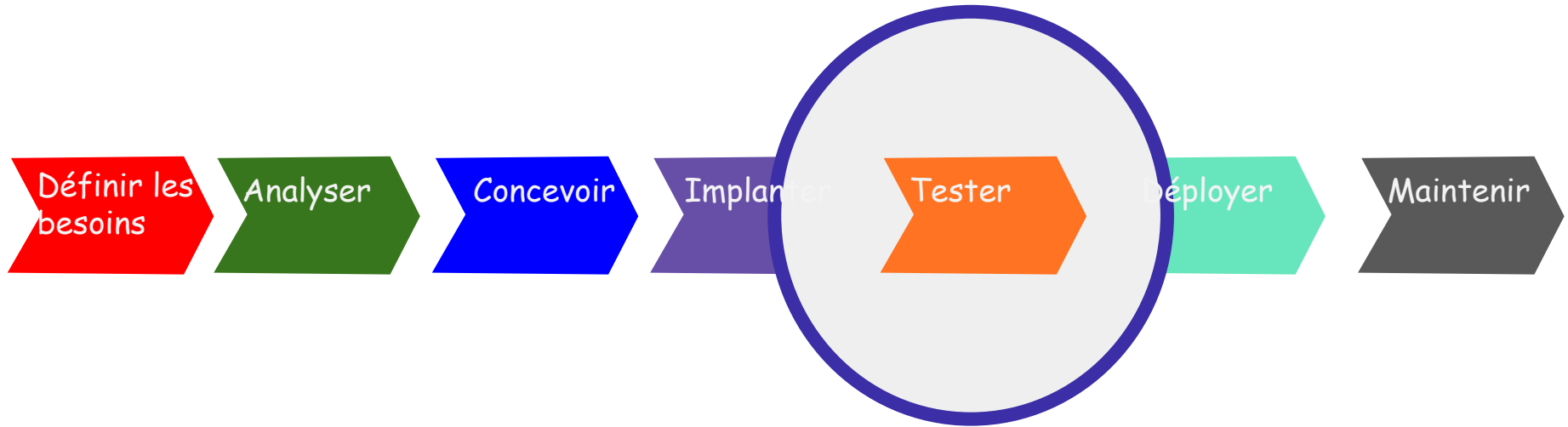
Aujourd'hui, ces étapes ne sont plus réalisées en cascade ... mais on verra ça plus tard.

# Les différentes étapes du développement logiciel



Définition d'un cahier des charges en accord avec le client, pour définir les besoins du logiciel.

# Les différentes étapes du développement logiciel



Vérifier que le logiciel obtenu est en accord avec le cahier des charges.

# Définitions

- ❑ Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus (IEEE).
- ❑ Attention, un test permet de révéler la présence d'erreurs, mais ne garantit pas l'absence d'erreurs (DIJKSTRA).

*Wikipédia*

*IEEE : Institute of Electrical and Electronics Engineers association professionnelle américaine de plus de 400 000 membres, ingénieurs électriciens, informaticiens, de professionnels du domaine des télécommunications, etc.*

*Edsger Wybe Dijkstra (prononciation : ['etsxər 'wibə 'dɛɪkstra]) est un mathématicien et informaticien néerlandais du xx<sup>e</sup> siècle.*

# Différentes sortes de tests

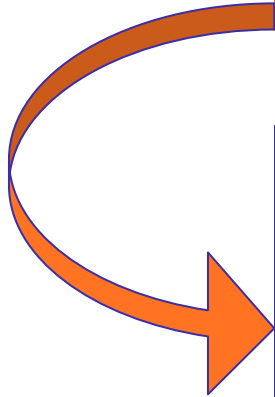
- ❑ **Tests statiques** : Revue de code, de spécifications, de documents de conception
- ❑ **Tests dynamiques** : Exécuter le code pour s'assurer d'un fonctionnement correct (60% des tests actuellement)
- ❑ **Vérifications formelles** : Preuves



# Pourquoi faire des tests dynamiques ? Comment ?

Aujourd'hui : vérifier que ce qui est implanté correspond à ce qui est attendu

Demain : s'assurer que les modifications n'ont pas cassé ce qui était correct



Savoir ce qui est attendu

Comment le vérifier ?

Conserver tous les tests

# Différentes sortes de tests dynamiques

## Tests boîte noire

On exécute le logiciel sur différentes données et on compare les résultats produits avec les résultats attendus.

## Tests boîte blanche

On teste chaque classe, chaque fonction l'une après l'autre. Les tests passent dans tous les chemins d'exécution possibles.

# Différentes sortes de tests dynamiques

Tests boîte noire

**Indépendant  
du code**

Le logiciel sur différentes données et on compare les résultats produits avec les résultats attendus.

Tests boîte  
blanche

**Lié au code**

Chaque classe, chaque fonction l'une après l'autre. On passe dans tous les chemins d'exécution possibles.

# Différentes sortes de tests dynamiques

Tests boîte noire

**Awaz**

Le logiciel sur différentes données et on compare les résultats produits avec les résultats attendus.

Tests boîte blanche

**Iceberg2D**

Chaque classe, chaque fonction l'une après l'autre. On passe dans tous les chemins d'exécution possibles.

# Différentes sortes de tests dynamiques

Tests boîte noire

**Indépendant  
du code**

Le logiciel sur différentes données et on compare les résultats produits avec les résultats attendus.

Tests boîte  
blanche

**Lié au code**

On vérifie chaque classe, chaque fonction l'une après l'autre. On passe dans tous les chemins d'exécution possibles.

En pratique, ces deux sortes de tests sont indispensables.

## Couverture de code

- ❑ Dans les deux cas (boîte blanche/noire), le test exhaustif est impossible.
- ❑ La couverture de code est une mesure utilisée en génie logiciel pour décrire le taux de code source testé d'un programme.
  - Mesurer la qualité des tests effectués.
  - Un test sur une seule donnée ne suffit donc pas.
  - Se rapprocher d'une couverture à 100%

# Tests boîte noire

- ❑ S'appuyer sur le cahier des charges établi en accord avec le client  
= contrat qui engage le client et l'équipe de développement
- ❑ Le cahier des charges impose des scénarios d'exécution qu'il convient de reproduire.

Cahier des charges de Adobe Reader

Quand l'utilisateur clique sur le menu Fichier/Ouvrir, il peut choisir un fichier dans son arborescence de fichiers ; celui est alors ouvert et affiché



# Tests boîte blanche

- ❑ Tester classe par classe, fonction par fonction
- ❑ S'appuyer sur
  - la documentation de la classe et de la fonction à tester
  - les instructions qui constituent le corps de la fonction
- ❑ Pour tester une fonction, on a besoin d'un receveur et peut-être de paramètres
  - comment les choisir ?



# Choisir les données des tests

- ❑ La donnée est dans un domaine infini (ex : un point)
  - Identifier des classes d'équivalence d'objets supposés se comporter de la même façon
  - Souvent, on identifie un élément particulier, tout seul dans une classe
  
- ❑ La donnée est dans un domaine borné (ex : un intervalle)
  - Tester une valeur quelconque mais aussi les valeurs limites
  
- ❑ La donnée est une collection
  - S'inquiéter du taux de remplissage de la collection

# Organisation des tests

- ❑ En pratique, si on combine toutes les possibilités des données, le nombre de cas de tests est important.
- ❑ Le nombre de lignes à écrire est conséquent.
- ❑ Sans méthode, le test est laborieux, ce qui explique les réticences des programmeurs.

Objectif : faciliter l'écriture et la relecture des tests, l'adjonction de nouveaux tests pour augmenter la couverture

# Organisation des tests .... un peu de méthode

- ❑ Bannir l'affichage de résultats sur la sortie standard.
- ❑ Bannir la saisie des données.
- ❑ Ecrire un programme de test qui vérifie lui-même les résultats des tests en ne signalant que les bugs.
- ❑ Créer de nouveaux objets à chaque test.
- ❑ Structurer les tests
  - ❑ = une fonction main par classe
  - ❑ = ou un package de test avec une classe de test par classe
  - ❑ une fonction par fonction testée
  - ❑

# Organisation des tests ..... un exemple

- ❑ Test de la classe Iceberg2D

→ classe de test `glaces.tests.TestIceberg2D`

- ❑ Test d'une fonction, par exemple

`public boolean estPlusGrosQue(Iceberg2D i)`

- ❑ Que doit faire la fonction ?

→ sous l'hypothèse que le paramètre est non null, elle retourne true si le receveur est plus volumineux que i, false sinon

→ deux sortes de résultats possibles, donc au moins 2 cas de tests


```
package glaces.tests ;  
    public class TestIceberg2D {  
        public static void main (String[] a) {  
            testEstPlusGrosQueCasFaux() ;  
            testEstPlusGrosQueCasVrai() ;  
  
            ...  
        }  
    }
```

```
        private static void testEstPlusGrosQueFaux() {  
            ...  
        }
```

```
        private static void testEstPlusGrosQueVrai() {  
            ...  
        }
```


```
    ..
```

```
}
```



Une fonction  
privée par cas  
de test

```
package glaces.tests ;  
public class TestIceberg2D {  
    public static void main (String[] a) {  
        testEstPlusGrosQueFaux() ;  
        testEstPlusGrosQueVrai() ;  
        ...  
    }  
    private static void testEstPlusGrosQueFaux() {  
        // this est plus petit que le paramètre  
        Point p1, p2, p3, p4 ; Iceberg i1, i2 ; boolean b ;
```

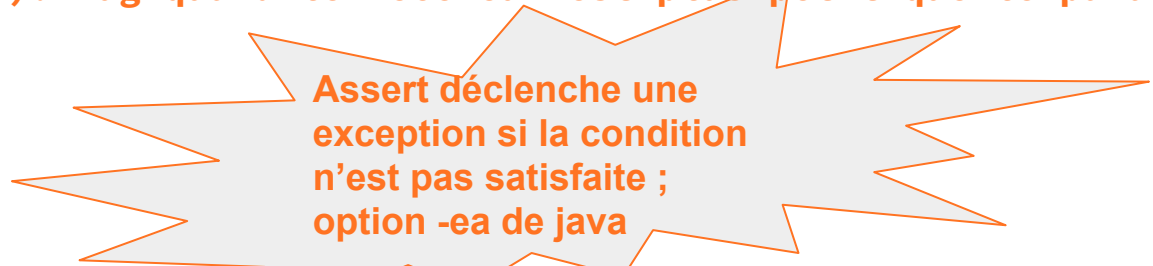


Déclarer les  
variables au  
début de la  
fonction

```
        p1 = new Point(1., 2.) ; p2 = new Point(3., 4.) ;  
        i1 = new Iceberg(p1, p2);  
        p3 = new Point(10., 2.) ; p4 = new Point(355., 444.) ;  
        i2 = new Iceberg(p3, p4);  
        b = i1.estPlusGrosQue(i2) ;
```

```
        assert (!b):"Bug quand le receveur est plus petit que le paramètre";
```

```
    } .. }
```



Assert déclenche une  
exception si la condition  
n'est pas satisfaite ;  
option -ea de java

```

package glaces.tests ;
    public class TestIceberg2D {
        public static void main (String[] a) {
            testEstPlusGrosQueFaux() ;
            testEstPlusGrosQueVrai() ;
            ...
        }
        private static void testEstPlusGrosQueVrai() {
            // this est plus gros que le paramètre
            Point p1, p2, p3, p4 ; Iceberg i1, i2 ; boolean b ;

            p1 = new Point(100., 2.) ; p2 = new Point(3000., 4.) ;
            i1 = new Iceberg(p1, p2);
            p3 = new Point(10., 2.) ; p4 = new Point(44., 44.) ;
            i2 = new Iceberg(p3, p4);
            b = i1.estPlusGrosQue(i2) ;
            assert (b):"Bug quand le receveur est plus gros que le paramètre";

        }
        ..
    }
}

```

# Les tests dynamiques ... en résumé

Savoir ce qui est attendu


Comment le vérifier ?

Conserver tous les tests



# Les tests dynamiques ... en résumé

Savoir ce qui est attendu



Spécification/cahier des  
charges/documentation/  
programme

Comment le vérifier ?

Conserver tous les tests

# Les tests dynamiques ... en résumé

Savoir ce qui est attendu

Spécification/cahier des charges/documentation/programme

Comment le vérifier ?

Tests boîte noire/boîte blanche. Couverture importante. Tests automatiques

Conserver tous les tests

# Les tests dynamiques ... en résumé

Savoir ce qui est attendu

Spécification/cahier des charges/documentation/programme

Comment le vérifier ?

Tests boîte noire/boîte blanche. Couverture importante. Tests automatiques

Conserver tous les tests

Une fonction main par classe. Un package de classes de tests.