Atelier T6.A4
Bien écrire les tests de son composant logiciel PHP

Ivan Enderlin - Jean-Marie Gauthier

6 septembre 2013

Université de Franche-Comté
Institut FEMTO-ST

DISC, Besançon, France

Qu'est-ce qu'un test ?

1. Un système sous test (SUT) : ici un robot
2. Des données de test : permet de placer le système dans un état précis
3. Un oracle : calcule le verdict du test (succès ou échec)

Pourquoi écrire des tests ?

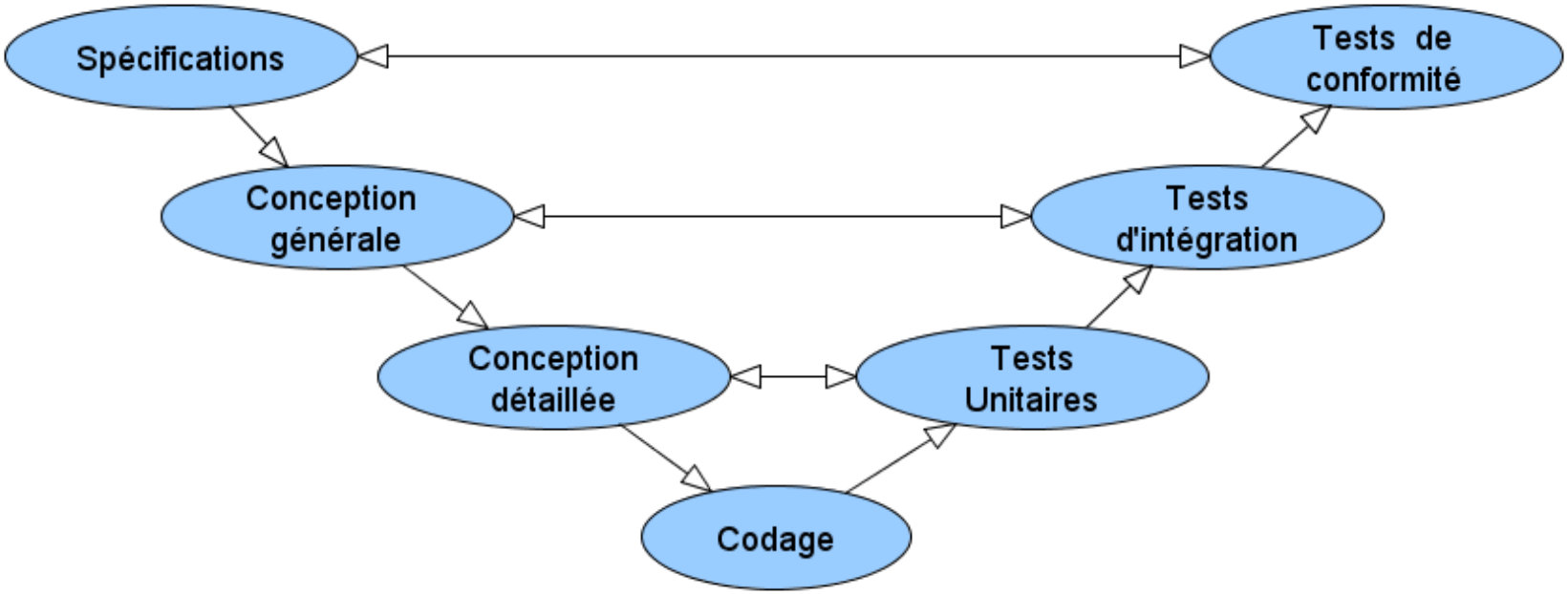
Une suite de tests (manuels) est une spécification informelle.

Objectifs lors du développement :

- Conformité avec la spécification
- Détection au plus tôt des régressions
- Maintenance réduite
- Plus de qualité et de confiance dans le code

Coût des bugs estimé à 60 milliards USD par an

Test et cycle de vie



Le test unitaire

De part sa nature et sa proximité du code source, le test unitaire est pleinement associé à l'activité de développement : **le test unitaire est à la charge des développeurs**.

Attention...

- Le test est une méthode de vérification **partielle** de logiciels (non exhaustive)
- La qualité du test dépend de la **pertinence du choix des données** de test

Présentation de Unitestor

Un robot qui se déplace sur une planète lointaine...

Composants

- une horloge interne
- une batterie
- un panneau solaire
- un GPS
- un capteur de sol

Fonctionnalités

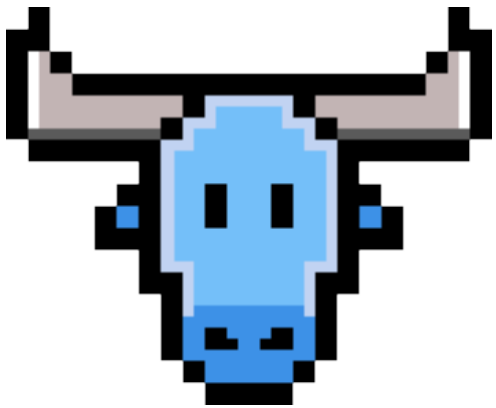
- déplacement : en fonction du sol et de la distance, plus ou moins d'énergie sera consommée
- rechargement de la batterie : le panneau solaire recharge la batterie de manière constante

Architecture

- `Source/Unitestor/` : sources du projet
  - `Clock.php`
  - `Coordinates.php`
  - `LandSendor.php`
  - `Robot.php`
  - `Vector.php`
- `Test/Unitestor/` : tests unitaires manuels

Présentation atoum

- Un framework xUnit PHP
- Simple : utilisation et déploiement facile
- Moderne : utilise PHP 5.3+ et compatibilité avec les outils industriels standards
- Intuitif : écriture et lecture des tests “naturelles”



Étape 1

Mes premiers pas avec atoum !

Écriture des tests pour :

- `Coordinates.php` : tester les *getters*
- `Vector.php` : tester les *getters* et le calcul de la longueur d'un vecteur

Bouchonnage avec les mocks

Le code à des dépendances : système de fichiers, base de données, connexion réseau, Web service, etc.

Un mock permet de simuler un composant :

- Améliore la portabilité des tests
- Permet de simuler des erreurs, des pannes, des montées en charges, etc
- Accélère l'exécution des tests

En pratique, un mock M_C est un "clone" qui étend une classe C. Il est utilisé à la place de C dans les tests.

Étape 2

Mes premiers pas avec les mocks !

- `Clock.php` : tester la méthode `getDifference()`
- `Landsensor.php` : tester la méthode `getNeededEnergy()` (mock orphelin et mock de fonction)

Quelle est la qualité de mes tests ?

Le moyen principal est la couverture du graphe de flot de contrôle de la méthode à tester.

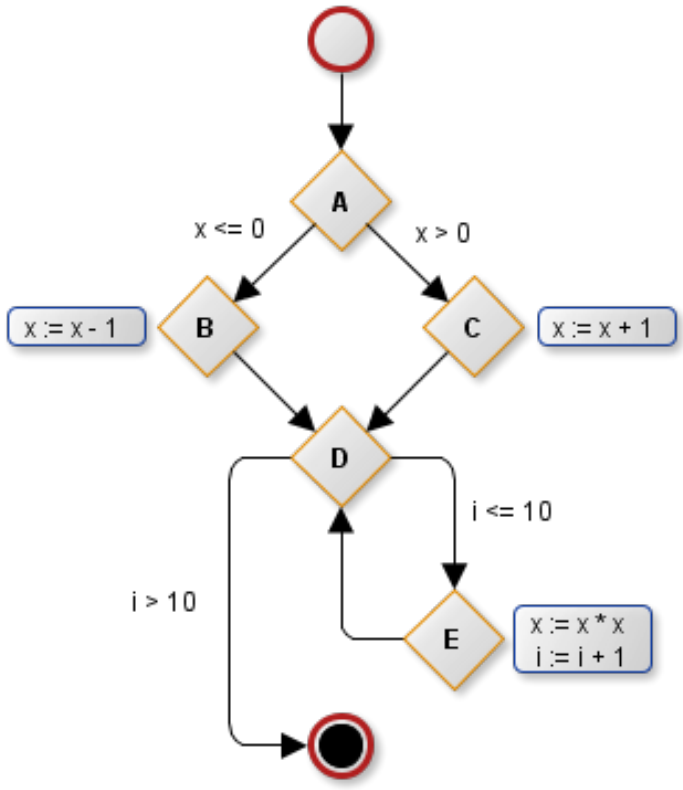
- Un graphe permet de représenter n'importe quel algorithme sous la forme de **noeuds** et **d'arcs**.
- Les noeuds du graphe représentent des **blocs d'instructions**.
- Les arcs représentent la possibilité de **transfert de l'exécution** d'un noeud à un autre.
- Il possède une seule **entrée** (noeud à partir duquel on peut visiter tous les autres) et une seule **sortie**.

Graphe de flot de contrôle : exemple

Soit le programme P_1 suivant :

```
1 if x <= 0 then
2   x := x - 1
3 else
4   x := x + 1
5 end if
6 for i : 0..10 do
7   x := x * x
8   i := i + 1
9 end for
```

Et son graphe G_1 :



Étape 3

Tricoter ma première couverture...

- `Robot.php` : couvrir la méthode `move()` à 100%

Praspel

- Praspel est un langage de spécification pour PHP
- Écriture de contrats en annotation directement dans le code source sur les données
- Est utilisé pour du *Contract-Based Testing*

Principe

Exploite le contrat pour générer des cas de tests :

- Utilise les préconditions pour générer des données de tests
- Utilise les postconditions et les invariants pour établir le verdict du test

Premier pas avec Praspel

1. Un système sous test (SUT) : ici un robot
2. Des données de test : Praspel partie 1
3. Un oracle : Praspel partie 2

Etape 4

Génération automatique de données de tests.

- Utilisation du shell Praspel
- Données dynamiques dans les tests

Merci beaucoup

Merci pour votre attention !

Quelques liens

- ivan.enderlin@femto-st.fr
- jean-marie.gauthier@femto-st.fr
- `https://github.com/cnrs-jdev/unitestor.php`
- `http://atoum.org/`
- `http://hoa-project.net/`