Humanbooster

Algorithmie

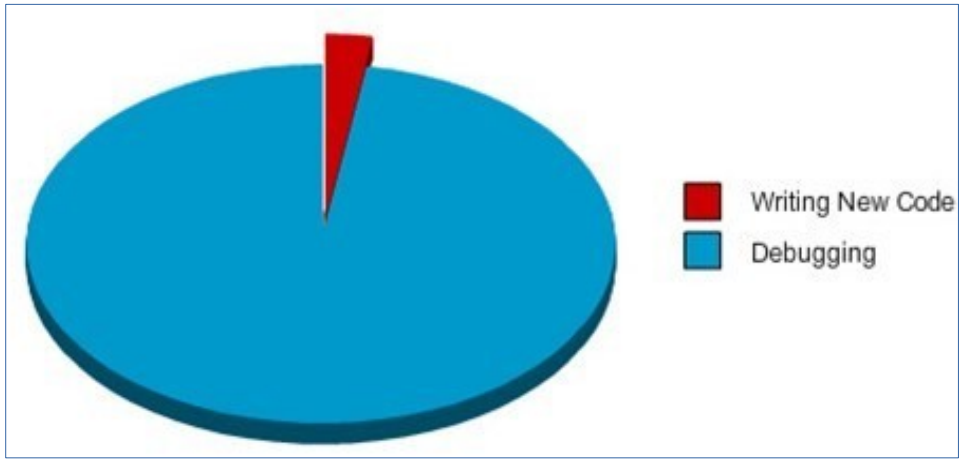
03 – Tests et TDD

DWWM PE6 LYON

Barthélémy DELUY –

Introduction

La majorité du travail de développeur consiste à trouver et résoudre les erreurs dans le code :



Comment faire pour inverser cette tendance ?

Sommaire

1. Notions générales sur les tests
2. Le TDD

1. Les tests en informatique

1. À quoi servent les tests ?
2. Les tests unitaires
3. L’importance du découpage

1.1. À quoi servent les tests ?

Les tests ont de nombreuses utilités :

- détecter les erreurs de programmation (bogues/bugs)
- vérifier qu'une modification du code n'introduit pas de régression (cassure de code existant)
- vérifier que le développement réalisé fonctionne dans son environnement cible
- vérifier que le projet répond bien aux besoins du client

1.2. Les tests unitaires

Les tests unitaires permettent de tester un fragment du code, isolé de tout contexte :

- **le code dépendant de services extérieur est « bouchonné » à l'aide d'un dummy, d'un stub, d'un fake, ou d'un mock (détails) - attention, ce concept dépend de la programmation par interfaces et l'injection de dépendance, que vous verrez avec PHP et Symfony**
- **Chaque test ne doit tester qu'une et une seule routine du SUT (System Under Test, la classe que vous êtes en train de tester)**
- **On ne teste que s'il y a quelque chose à tester ! inutile de tester une méthode « function afficherNom($nom) { echo $nom;} » Il est contre-productif de viser un taux de « Code Coverage » de 100 %**

1.3. L’importance du découpage

Autant pour faciliter l’écriture des tests que pour simplifier le code et limiter les bugs, il est souvent plus facile de découper son code :

Avant :

```
function estAvant($a1, $m1, $j1, $a2, $m2, $j2) {
    /*


     50 lignes de code



    */
}
```

Après :

```
function isLeapYear($annee) {
    /*
        5 lignes de code
    */
}
function isMonthValid($m) {
    /*
        5 lignes de code
    */
}
function isYearValid($a) {
    /*
        5 lignes de code
    */
}
```

1. Les tests en informatique 1/4

Il existe de nombreux autres types de tests que nous n'aborderons pas dans ce cours (tests de charge, tests de pénétration, fuzzing, etc).

Nous n'utiliserons que les tests unitaires et les tests d'intégration.

- **Test unitaire : on teste une routine, voire un morceau d'une routine**
- **Test d'intégration : on teste que différentes routines communiquent bien entre elles**

Dans les faits, on ne les distinguera pas dans le cadre de ce module.

1. Les tests en informatique 2/4

Quelques notions :

- **Les données que l'on utilise pour tester son code s'appellent « un jeu d'essai »**
- **Un test a 2 résultats possibles : réussite ou échec ; un test en échec montre une anomalie dans le comportement du code**
- **Le cas de « comportement standard » de l'application s'appelle le cas « nominal »**
- **Les cas rares mais corrects s'appellent « cas marginaux » ou « edge cases » en anglais**
- **Les cas ne devant pas fonctionner s'appellent « cas d'erreur »**
- **Un cas d'erreur qui réussit, c'est un cas d'erreur où l'exception est bien levée par le code : on considère ici *l'échec du code* comme une *réussite au test* !**

1. Les tests en informatique 3/4

Quelques remarques :

- **Le test ne dit pas quelle est l'erreur, il dit seulement qu'il y en a une**
- **Le test ne corrige pas l'erreur**
- **Ce n'est pas parce que le test passe qu'il n'y a pas d'erreur**
- **Ce n'est pas parce que vous corrigez l'erreur qu'il n'y en a plus**

1. Les tests en informatique 4/4

Il existe de nombreux outils pour faire des tests, en fonction du langage : Jest pour JS, PHPUnit et Atoum pour PHP, ...

Dans le cadre du module Algo, on va les faire « à la main » :

```
function isLeapYear($a) {…}

function testIsLeapYear_400() {
    if (isLeapYear(1600)) {
        echo « Test isLeapYear(400) : OK » ;
    }
    else { echo « Test isLeapYear(400) : Échec » ; }
}
```

2. Le TDD

TDD signifie « Test-Driven Development », soit « Développement piloté par les Tests » en français.

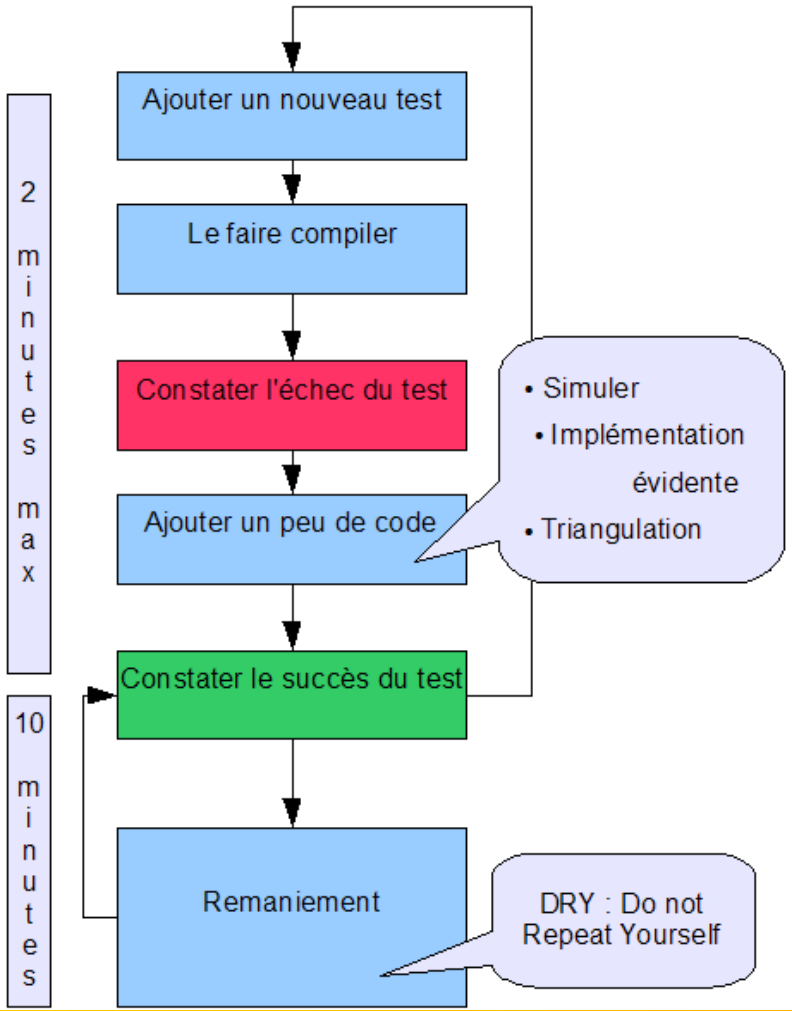
Il s’agit d’une méthode de développement, de plus en plus utilisée car elle permet de réduire drastiquement le temps consacré au débogage.

2. Le cycle du TDD

Le TDD suit quelques étapes strictes :

1. **Création du squelette de la routine à tester (routine vide)**
2. **Écriture d'un test simple, qui doit échouer**
3. **Écriture du code le plus simple possible pour valider le test**
4. **[optionnel] Refactoring du code**

2. Le cycle du TDD



2. Le TDD – Exemple

Code	Test

function isLeapYear($a) {
  return false ;
}
	public void testIsLeapYear_400() {
	  if (isLeapYear(400)) {
	  echo « test isLeapYear(400) réussi ! »;
	  } else {
	  echo « test isLeapYear(400) échoué ! » ;
	  }
	}
	=> Le test échoue, false != true

2. Le TDD – Exemple

Code	Test

function isLeapYear($a) {
  return **true** ;
}
	public void testIsLeapYear_400() {
	  if (isLeapYear(400)) {
	  echo « test isLeapYear(400) réussi ! »;
	  } else {
	  echo « test isLeapYear(400) échoué ! » ;
	  }
	}
	=> Le test réussie, true == true

2. Le TDD – Exemple

Code	Test

function isLeapYear($a) {
  return true ;
}
	public void testIsLeapYear_400() {
	  if (isLeapYear(400)) {
	  echo « test isLeapYear(400) réussi ! »;
	  } else {
	  echo « test isLeapYear(400) échoué ! » ;
	  }
	}
	=> Le test réussie, true == true
	public void testIsLeapYear_401() {
	  if (!isLeapYear(401)) {
	  echo « test isLeapYear(401) réussi ! »;
	  } else {
	  echo « test isLeapYear(401) échoué ! » ;
	  }
	}
	=> Le test échoue, false != true

2. Le TDD – Exemple 4/5

Code	Test

function isLeapYear($a) {
  return **false** ;
}
	public void testIsLeapYear_400() {
	  if (isLeapYear(400)) {
	  echo « test isLeapYear(400) réussi ! »;
	  } else {
	  echo « test isLeapYear(400) échoué ! » ;
	  }
	}
	=> Le test échoue, false != true
	public void testIsLeapYear_401() {
	  if (**!**isLeapYear(401)) {
	  echo « test isLeapYear(401) réussi ! »;
	  } else {
	  echo « test isLeapYear(401) échoué ! » ;
	  }
	}
	=> Le test réussie, true == true

2. Le TDD – Exemple 5/5

Code	Test

function isLeapYear($a) {
if (($a % 400) == 0) { return true ; }
return false ;
}
	public void testIsLeapYear_400() {
	if (isLeapYear(400)) {
	echo « test isLeapYear(400) réussi ! »;
	} else {
	echo « test isLeapYear(400) échoué ! » ;
	}
	}
	=> Le test réussie, true == true
	public void testIsLeapYear_401() {
	if (**!**isLeapYear(401)) {
	echo « test isLeapYear(401) réussi ! »;
	} else {
	echo « test isLeapYear(401) échoué ! » ;
	}
	}
	=> Le test réussie, true == true

Conclusion

Ce support vous a présenté l'importance des tests en développement informatique, ainsi que le cycle du TDD.

Vous allez désormais mettre en pratique ces connaissances dans le TP 02 :-)