Les tests de logiciel

Tests Unitaires : Xunit

Ibrahima TOUNKARA Jonathan TWAMBA SAIBA

Université de Montpellier

20 Février 2017

Tables des Matières

Table des Matières

Introduction I

- **Qu'est-ce qu'un logiciel ?**
  - des besoins, exprimés par un client ;
  - une gestion de projet :
    - cycle de vie (classique, agile, ...) avec ses délivrables
    - les outils et l'environnement de développement
  - une spécication (fonctionalités, contraintes, facteur de qualité, interface) ;
  - une conception : globale, architecturale, détaillée. Les composants et leurs échanges sont définis ;
  - un code source ;
  - un exécutable (un prodit) ;
  - ... et des tests !

Introduction II

- **Qu'est-ce que tester un logiciel ?**

  Définition possible : "Processus d'analyse d'un programme avec l'intention de détecter des anomalies dans le but de le valider"

  Tester un logiciel : C'est valider sa conformité, par rapport à des exigences c'est-à-dire par rapport à l'ensemble de la spécification, et de la conception du logiciel.

Table des Matières

Types

- Test fonctionel

Types

- Test fonctionel
- Test non fonctionnel

Types

- Test fonctionel
- Test non fonctionnel
- Test structurel

Table des Matières

Niveaux

- Tests unitaires ou tests de composants

Niveaux

- Tests unitaires ou tests de composants
- Test d'intégration

Niveaux

- Tests unitaires ou tests de composants
- Test d'intégration
- Test fonctionnel

Niveaux

- Tests unitaires ou tests de composants
- Test d'intégration
- Test fonctionnel
- Test Système

Niveaux

- Tests unitaires ou tests de composants
- Test d'intégration
- Test fonctionnel
- Test Système
- Test d'acceptation

Niveaux

- Tests unitaires ou tests de composants
- Test d'intégration
- Test fonctionnel
- Test Système
- Test d'acceptation
- Les Beta-Tests

Niveaux

- Tests unitaires ou tests de composants
- Test d'intégration
- Test fonctionnel
- Test Système
- Test d'acceptation
- Les Beta-Tests
- Test de regression

Niveaux

Test	**Portée**	**Catégorie**	**Exécutant**
Unitaires	Petites portions du code source	Boite noire	Développeur machine
Intégration	Classes/composants	Boite blanche/ noire	Développeur
Fonctionnel	Produit	Boite noire	Testeur
Système	Produit/simulation de situation	Boite noire	Testeur
Acceptation	Produit/utilisation réelle	Boite noire	Client
Béta	Produit/ Utilisation réelle	Boite noire	client
Régression	N'importe lequel	Boite blanche/ noire	Peu importe

Niveaux

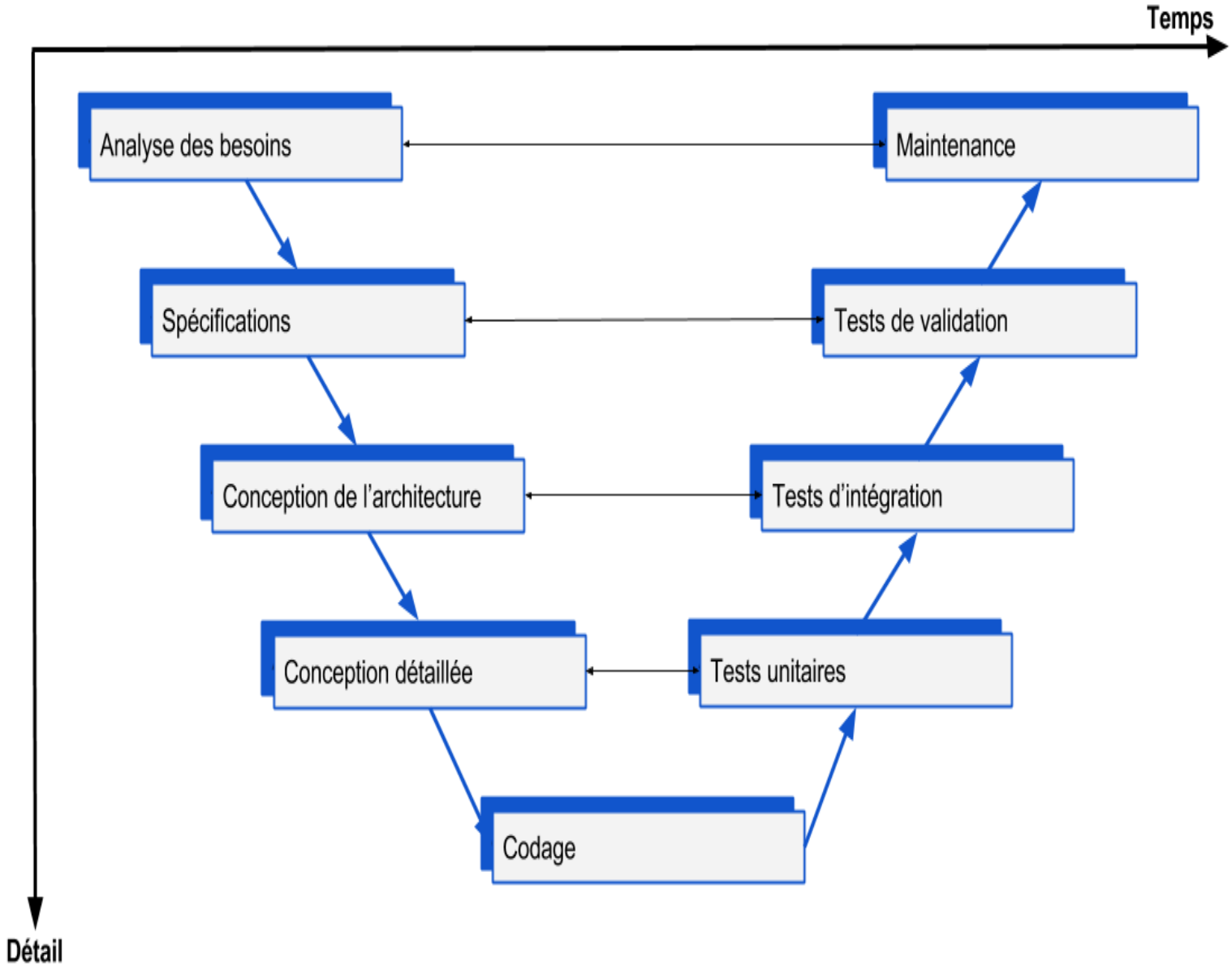
Temps
Analyse des besoins
Maintenance
Spécifications
Tests de validation
Conception de l'architecture
Tests d'intégration
Conception détaillée
Tests unitaires
Codage
Détail

Table des Matières

Objectifs

- Son objectif principal est d'identifier un nombre maximum de comportements problématiques du logiciel afin d'en augmenter la qualité.

Objectifs

- Son objectif principal est d'identifier un nombre maximum de comportements problématiques du logiciel afin d'en augmenter la qualité.
- Le test peut aussi avoir pour objectif d'apporter des informations quant à cette qualité afin de permettre la prise de décisions.

Objectifs

- Son objectif principal est d'identifier un nombre maximum de comportements problématiques du logiciel afin d'en augmenter la qualité.
- Le test peut aussi avoir pour objectif d'apporter des informations quant à cette qualité afin de permettre la prise de décisions.
- Les tests de vérification ou de validation visent ainsi à vérifier que ce système réagit de la façon prévue par ses développeurs ou est conforme aux besoins du client.

Objectifs

- Un test ressemble à une expérience scientifique. Il examine une hypothèse exprimée en fonction de trois éléments : les données en entrée, l'objet à tester et les observations attendues.

Objectifs

- Un test ressemble à une expérience scientifique. Il examine une hypothèse exprimée en fonction de trois éléments : les données en entrée, l'objet à tester et les observations attendues.
- Eliminer les défauts qui conduisent à des défaillances avant que le logiciel entre en service
- S'assurer que le logiciel satisfait un certain nombre légal ou contractuel nécessaire pour une certification du logiciel par exemple.

Table des Matières

Quelques principes de Test

- **Montrer la présence de défauts**
  Il est important de savoir que les tests ne peuvent pas prouver qu'il n'y a pas d'erreurs mais plutôt de montrer la présence de défauts pour pouvoir les corriger. Il est important de faire des tests pertinents avant de conclure qu'il n'a pas de défauts.

Quelques principes de Test

- **Montrer la présence de défauts**
  Il est important de savoir que les tests ne peuvent pas prouver qu'il n'y a pas d'erreurs mais plutôt de montrer la présence de défauts pour pouvoir les corriger. Il est important de faire des tests pertinents avant de conclure qu'il n'a pas de défauts.
- **Tester raisonnablement**
  Tester un logiciel ne signifie pas tester tous les états de celui-ci. Il est donc important de se concentrer sur l'essentiel ; car pour certain programme c'est théoriquement impossible vu le temps que ça prendrait.

Quelques principes de Test

- **Tester tôt**
  Les tests doivent commencer plus tôt pour pouvoir retrouver les erreurs plus vites et ainsi les corriger car les difficultés de correction augmentent avec la progression.

Quelques principes de Test

- **Tester tôt**
  Les tests doivent commencer plus tôt pour pouvoir retrouver les erreurs plus vites et ainsi les corriger car les difficultés de correction augmentent avec la progression.
- **Regrouper les défauts**
  Il faut répartir uniformémént les tests pour ne pas oublier certaines parties du logiciel. Si l'on divise le logiciel en des parties, on remarquera que 80% des erreurs de celui-ci sont sont concentrées dans 20% des parties, c'est la règle du 80/20. Il faudra ainsi agir en conséquence sur les 80%.

Quelques principes de Test

- **Tester des parties bien définies du Logiciel**
  Après avoir testé longtemps, le nombre d'erreurs va en décroisssant. Il est important de choisir donc la méthode de test appropriée pour chaque partie de logiciels et constamment renouveler les tests en changeant de point de vue.

Quelques principes de Test

- **Tester des parties bien définies du Logiciel**
  Après avoir testé longtemps, le nombre d'erreurs va en décroisssant. Il est important de choisir donc la méthode de test appropriée pour chaque partie de logiciels et constamment renouveler les tests en changeant de point de vue.
- **Tester selon le contexte**
  Certains logiciels peuvent être utilisés dans des contexte différents.Par exemple, un système de gestion de bases de données peut être utilisé pour mémoriser des transactions bancaires. En fonction de l'utilisation du composant, les objectifs et les types de test ne seront pas forcément les mêmes.Les paramètres temps et environnement peuvent beaucoup jouer.

Quelques principes de Test

- **Respectez les normes**
  Il faut le rappeler, les tests sont là pour découvrir des défauts par rapport à ce que l'on attend du logiciel. Mais il ne suffit pas de répérer et de corriger toutes les erreurs du logiciel, il faut le rendre utilisable.L'absence de défauts sous un angle particulier ne garantira pas une bonne qualité sous un autre. Les différents attendus du logiciels doivent correspondre à la norme ISO 9126 dont les critères principaux sont : Fonctionnality, Usability, Reliability, Portability, Efficiency, Maintainability.

Table des Matières

Introduction

- Les tests unitaires sont des tests en boîte blanche
- Les tests unitaires sont composés d'un ensemble de classes appelées "classes de test"
- Ces classes valident que des portions de code répondent à un certain besoin
- Les tests unitaires sont importants car ils permettent de détecter le maximum de défaillance avant les tests en boîte noire et qu'ils peuvent s'exécuter d'une manière automatique

Table des Matières

Objectifs

- Les tests unitaires ont deux objectifs : "couveture de code" et "couverture de données"
- La couverture du code stipule de tester chaque ligne de code écrite (appel de fonction, boucles, décisions, décisions multiples, ...
- La couverture des données oriente les tests vers les données (donnéess valides, données invalides, accès aux éléments en dehors de la capacité d'un tableau, peu de données, trop de données,... etc)

Table des Matières

Les frameworks de type xUnit

Pour écrire des tests unitaires, vous avez à votre disposition des frameworks qui vont vous faciliter l'écriture des tests. Vous n'aurez plus qu'à écrire les classes de tests et c'est le framework qui se chargera de les trouver, de les lancer et de vous donner les résultats ou les erreurs qui ont été détectées.

- **En Java** : JUnitest le framework de type xUnit le plus utilisé pour Java. Il est tellement utilisé qu'il est livré avec la plupart des IDE.

- **En C++** : Cutte, Google propose Google C++ Testing Framework, la fameuse bibliothèque Boost comprend la Boost Test Library.
- **En Python** : la distribution de base de Python intègre unittest mais il existe aussi PyUnit.
- **En PHP** : les développeurs PHP utilisent PHPUnit (ex : Zend Framework, Drupal V8[2]) ; SimpleTest (ex : Drupal V7[3]) qui possède une documentation en français ; Atoum.
- **En Ruby** : Ruby intègre Test : :Unit

- **En D** : le langage intègre nativement le mot-clé unittest.
- **En Groovy** : voir Unit Testing
- **En JavaScript** : le framework jQuery utilise qunit, Jarvis, jfUnit, google-js-test
- **Dans d'autres langages** : il existe des frameworks équivalents dans la plupart des langages : vous pouvez les découvrir dans l'article « List of unit testing frameworks » sur Wikipedia anglophone.

Table des Matières

JUnit

- **Écrire un test** : Avec JUnit, on va créer une nouvelle classe pour chaque classe testée. On crée autant de méthodes que de tests indépendants : imaginez que les tests peuvent être passés dans n'importe quel ordre (i.e. les méthodes peuvent être appelées dans un ordre différent de celui dans lequel elles apparaissent dans le code source). Il n'y a pas de limite au nombre de tests que vous pouvez écrire. Néanmoins, on essaye généralement d'écrire au moins un test par méthode de la classe testée. Pour désigner une méthode comme un test, il suffit de poser l'annotation `@Test`.

JUnit

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

public class StringTest {

    @Test
    public void testConcatenation() {
        String foo = "abc";
        String bar = "def";
        assertEquals("abcdef", foo + bar); }

    @Test
    public void testStartsWith() {
        String foo = "abc";
        assertTrue(foo.startsWith("ab")); }
}
```

JUnit

- **Les assertions**
  Via import static org.junit.Assert.*;, vous devez faire appel dans les tests aux méthodes statiques assertTrue, assertFalse, assertEquals, assertNull, fail, etc. en fournissant un message. Votre environnement de développement devrait vous permettre de découvrir leurs signatures grâce à l'auto-complétion. À défaut, vous pouvez toutes les retrouver dans la documentation de l'API JUnit. Attention à ne pas confondre les assertions JUnit avec «assert». Ce dernier est un élément de base du langage Java.

JUnit

- **Lancer les tests** :
  Pour lancer les tests, vous avez plusieurs possibilités selon vos préférences :
  - La plus courante : lancer les tests depuis votre IDE
  - Utiliser l'outil graphique
  - Lancer les tests en ligne de commande
  - Utiliser un système de construction logiciel (comme Ant ou Maven pour Java)

JUnit

- **Factoriser les éléments communs entre tests** :
  On peut déjà chercher à factoriser les éléments communs à tous les tests d'une seule classe. Un test commence toujours par l'initialisation de quelques instances de formes différentes pour pouvoir tester les différents cas. C'est souvent un élément redondant des tests d'une classe, aussi, on peut factoriser tout le code d'initialisation commun à tous les tests dans une méthode spéciale, qui sera appelée avant chaque test pour préparer les données. Si on considère l'exemple ci-dessus, cela donne :

```
import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.*;
public class StringTest {
    private String foo;
    private String bar;
    @Before
//cette methode sera appelee avant chaque test
    public void setup() {
        foo = "abc";
        bar = "def"; }
    @After
    public void tearDown() {
//dans cet exemple, il n'y a rien a faire mais on peut,
//dans d'autres cas, avoir besoin de fermer une connexio
//a une base de donnees ou de fermer des fichiers
    }
    @Test
    public void testConcatenation() {
```

```
        assertEquals("abcdef", foo + bar); }
    @Test
    public void testStartsWith() {
        assertTrue(foo.startsWith("ab")); }
}
```

JUnit

- **Écrire de bon tests** : Idéalement, les tests unitaires testent une classe et une seule et sont indépendants les uns des autres. En effet, si une classe A est mal codée et qu'elle échoue au test unitaire de A, alors B qui dépend de A (parce qu'elle manipule des instances de A ou hérite de A) va probablement échouer à son test alors qu'elle est bien codée. Selon les langages, on a souvent des conventions pour nommer les classes de tests. En Java, on choisit souvent d'appeler MaClassTest le test de la classe MaClass. Les tests peuvent être commentés si le déroulement d'un test est complexe. Évidemment, un bon test assure la qualité de l'intégralité d'une classe, et pas seulement d'une partie de celle-ci.

Table des Matières

Conclusion

Différentes stratégies de tests peuvent être mise en oeuvre pour tester un logiciel. Comme nous l'avons vu, les tests doivent être un guide lors des différentes phases du projet et ceci quelque soit le cycle de développement retenu pour rendre le logiciel plus simple à concevoir et à exécuter durant les phases de réalisation, de déploiement ou de maintenace du logiciel.

Conclusion

Merci de votre attention !

Bibliographie I


Jean-François Pradat-Peyre.
Pratique des tests logiciels.
DUNOD, 2014.


John Dooley.
Software Development and Professionnel Pratice.
APress, 2010.


F.X. Fornari.
Introduction aux tests du logiciel.
https ://www.irif.fr/ eleph/Enseignement/2010-11/CoursTests.pdf,
2011.

Bibliographie II

Merlyn Albery-Speyer.
Ten commandments of unit testing.
https ://monkeyis-land.pl/2008/01/31/10rules/http ://www.curiousattemptbunny.com/commandments-of-unit-testing.html,
2011.