Xposé IR3

Tests et TestNG

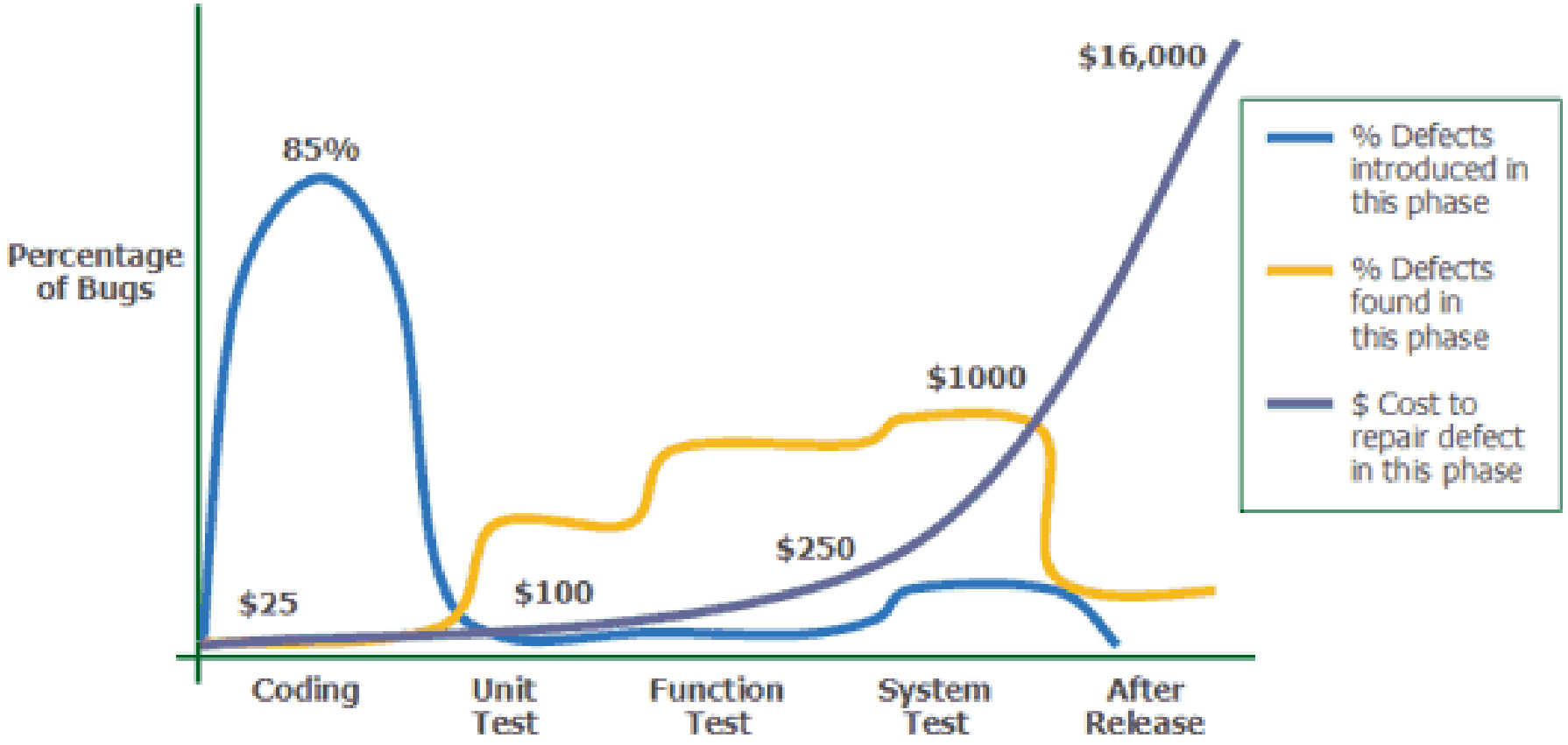
Sommaire

- Pourquoi tester un logiciel?
- Différents types de tests
- Plusieurs solutions existent
- TestNG
  - Historique
  - Principes
  - Exemples
- Conclusion

Pourquoi tester un logiciel ?

- Un logiciel est un problème complexe
- Les tests permettent de réaliser la cohérence et la fiabilité du logiciel

Pourquoi tester un logiciel ?



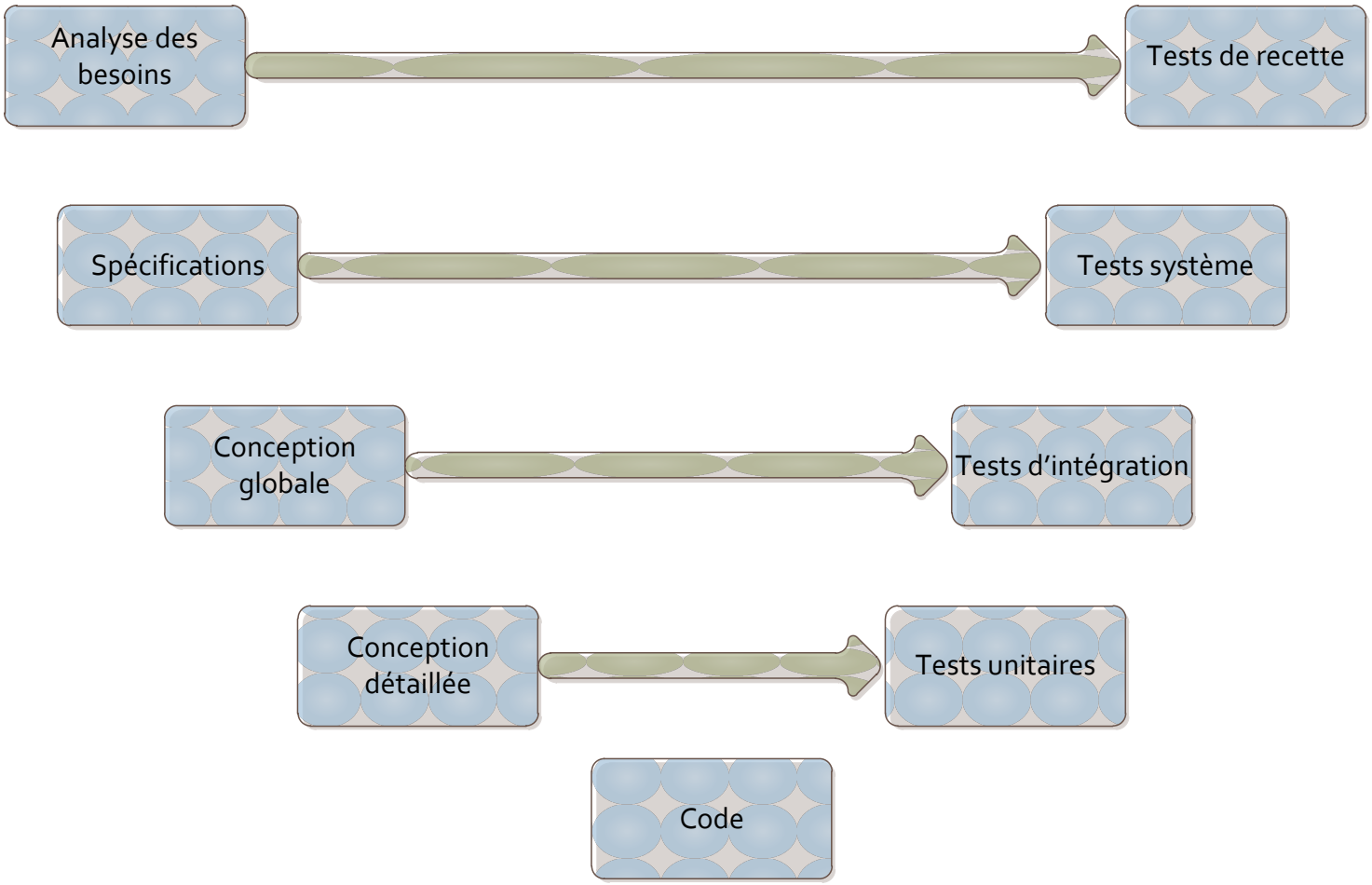
Catégories de tests

- Tests de conformité>> cohérence avec les algorithmes
- Tests de non régression
- Tests de performance
- Vérification / Validation
  - Vérification: Avons-nous créé le système correctement?
  - Validation: Le système créé correspond-il au besoin du client?

Tests et étapes de dev logiciel

- Tests unitaires
- Tests d'intégration
- Tests système
- Tests de recette
- Tests de non-régression

Cycle en V


Analyse des besoins
Tests de recette
Spécifications
Tests système
Conception globale
Tests d'intégration
Conception détaillée
Tests unitaires
Code

Framework de tests

- Plusieurs sur le marché:
  - JUnit
  - TestNG
  - Sélénium

Présentation de TestNG

- Inspiré de JUnit

- Quelques caractéristiques:
  - Supporté par beaucoup d'IDE grâce à des plugins
  - Vérification que le code testé est threadsafe
  - Choix de comment utiliser les threads pour exécuter les tests
  - Tout type de tests (d'unitaire à tests d'intégration)
  - Reporting des résultats des tests

Présentation de TestNG : principes

- 3 étapes de mise en oeuvre:
  - Écrire la logique des tests et ajouter les annotations de testNG sur les tests.
  - Définir quelles classes tester et dans quel ordre grâce au fichier testng.xml ou build.xml
  - Lancer TestNG

Présentation de TestNG: les annotations

- Tout comme dans Junit, TestNG possède un ensemble d'annotations dont les grandes catégories sont:
  - Informations de configuration pour les classes de tests
  - Déclaration de méthodes fournissant des données pour les tests
  - Marquage des classes de tests
  - Description d'utilisation des classes de tests

TestNG: Annotations

- Les plus utilisées:
  - @Test
  - @DataProvider
  - @BeforeTest
  - @AfterTest
  - @BeforeMethod
  - @AfterMethod
  - …

TestNG: invocation

- Plusieurs méthodes pour l'invoquer:
  - Avec le fichier testng.xml
  - Avec Ant
  - En ligne de commande

TestNG: testng.xml

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

 <suite name="testGestionBiblio" >
  <test name="testAjout" verbose="5">
    <packages>
        <package name="univ.mlv.fr.xpose.tests">
        </package>
     </packages>
 </test>
</suite>
```

TestNG: regroupement de tests

- Possibilité de définir des groupes de tests (les groupes pouvant inclure d'autres groupes)
- Spécifiés dans le fichier testng.xml

TestNG: génération dynamique

- Tests générés dynamiquement
- Factories

TestNG: parallélisation des tests

- Utilisation de l'attribut "parallel"
- Parallélisation:
  - Des tests méthodes dans des threads différents

  ⇒parallel="methods"

  - des tags <test>

  ⇒parallel="tests"

  - des classes

  ⇒Parallel="classes"

TestNG: Résultats des tests

- 3 états:
  - Succès
  - Échecs
  - “sauté”
- Résultats dans fichier index.html
- Possibilités de gérer son propre fichier de sortie des résultats (par ex: pdf)
- Ou encore visualisation par le biais de notre IDE

TestNG: Démo

Références

- http://testng.org/doc/index.html
- Cours de Génie Logiciel de Dominique Revuz
- « Object-Oriented Software Engineering » Ivar Jacobson

Conclusion

- Une bonne pratique !
- Exhaustivité des tests impossible
- Tester le code dès le début, pour corriger au plus vite et éviter des pertes de temps et d'argent importantes
- A ne pas négliger dans les projets, pensez-y pour les last project ;-)

Merci de votre attention

Avez-vous des questions?