Bases de la Programmation Orientée Objet / Java

DUT / Module M213

Organisation des tests unitaires

Cycle de développement des logiciels (1)

➔ Définition

Identification et organisation des différentes étapes de fabrication des applications logicielles (life cycle)

Description détaillée dans les modules :

- **M224** Gestion de projet informatique
- **M214** Bases de la conception orienté objet
- **M217** Description et planification de projets (PT)

Cycle de développement des logiciels (2)

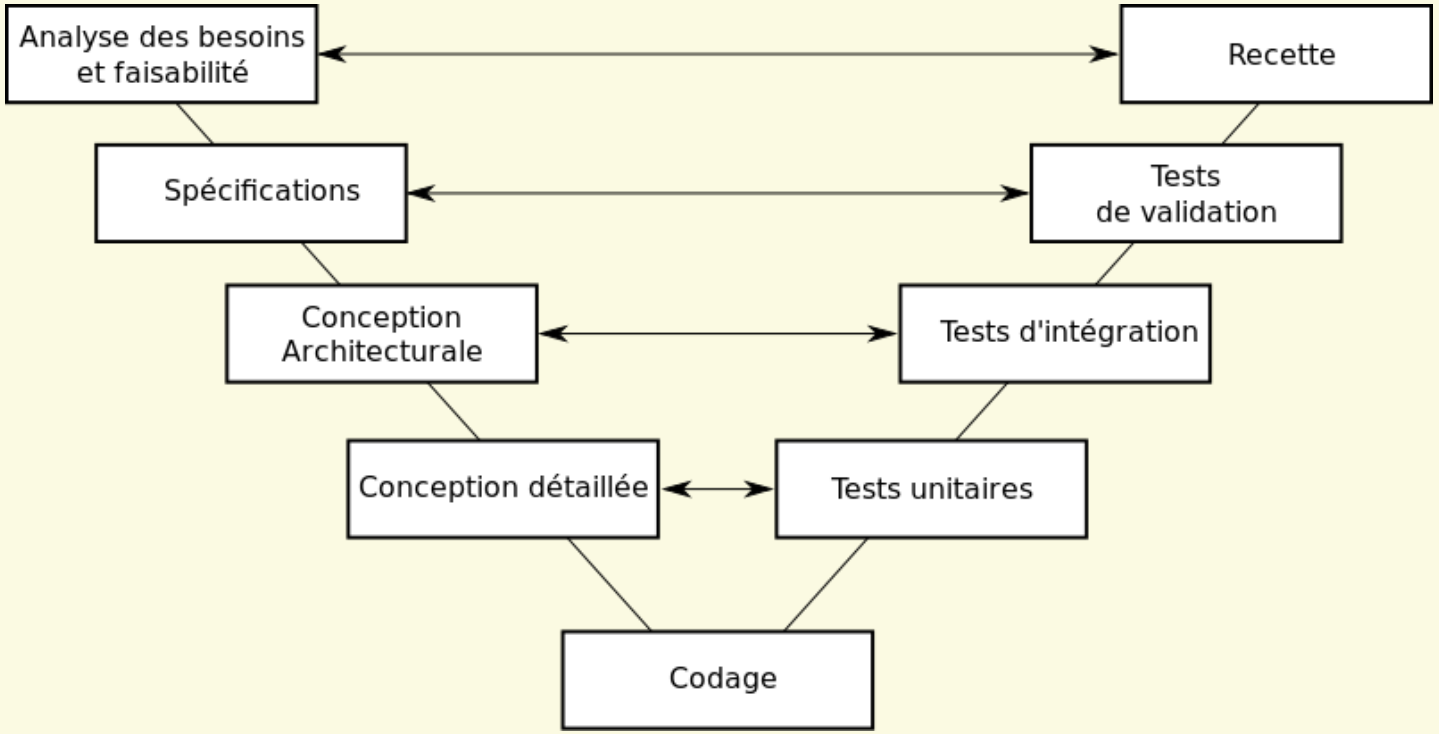
➔ **Différents modèles existants**

- **Modèle en cascade** (issu du BTP)
- **Cycle en V** (standard de fait)
- **Cycle en spirale** (analyse des risques)
- **Cycle semi itératif** (RAD, RUP, SCRUM, …)
- **Cycle itératif** (PDCA, PDSA, …)
- - - -

fr.wikipedia.org/wiki/Cycle_de_développement_(logiciel)

Cycle de développement des logiciels (3)

➔ Modèle du cycle en V



Cycle de développement des logiciels (4)

➔ **Phases de tests dans un projet**

- **Tests unitaires** (introduits en M112-M113)
- Tests d'intégration
- Tests de validation
- Tests de mise en condition opérationnelle
- Tests de déploiement

Des méthodes, des documents, des langages, des outils adaptés … pour chaque phase !

Cycle de développement des logiciels (5)

➔ **Documentation technique associée aux tests**

- **Plans et rapports de tests unitaires**
- **Plans et rapports de tests d'intégration**
- **Plan et rapport de validation**
- **- - -**

Documents produits suivant des plans types fournis par le **Plan Qualité du projet**

Réalisation des tests unitaires

➔ **Principes (**appliqués à la programmation objet)

Le développement d'une classe doit être accompagné en parallèle et avec le même soin de la réalisation de ses modules de tests unitaires.

En P.O.O., les modules de tests unitaires sont des classes spécialisées dotées de la méthode *main*

Indépendance structurelle des classes de tests unitaires et de la classe opérationnelle (sous tests).

Rôle des modules de tests unitaires

Les modules de tests unitaires doivent permettre de vérifier la conformité de la classe vis à vis de ses spécifications techniques détaillées.

Ils doivent permettre d'exécuter au moins une fois et **dans chacun des contextes opérationnels** :

- **chaque constructeur**
- **chaque méthode publique**

Cas particuliers et **cas généraux**

Limites des tests unitaires (1)

Un module de tests unitaires ne prouve pas que la classe cible fonctionne parfaitement.

Seuls les langages disposant d'un modèle formel permettent d'atteindre l'exhaustivité :

- Prolog (prog. déclarative / I.A.)
- Esterel (prog. synchrone / Systèmes réactifs)
- Lustre (prog. par flots / Systèmes critiques)
- Lustre ++ (Dassault Aviation)
- ...

Limites des tests unitaires (2)

Un module de tests unitaires permet seulement de vérifier que les résultats produits par les méthodes exécutées sont conformes à des résultats attendus et prédéfinis.

Le programmeur a la charge de définir de manière la plus complète possible (ratio coûts/délais/efficacité) :

- **tous les contextes d'exécution**
- **tous les résultats attendus**

*www.jmdoudoux.fr/java/dej/chap-frameworks-**test**.htm*

Conditions d'exécution et de contrôle

La maîtrise des contextes opérationnels des méthodes de la classe cible doit être l'unique pré-requis pour exécuter des modules de tests unitaires et en contrôler les résultats.

Aucune explication ou connaissance particulière ne doit être nécessaire à l'opérateur, **notamment celle des codes sources.**

Organisation des tests unitaires (1)

Les modules de tests unitaires d'une classe seront séparés en deux catégories :

- **contrôles des cas nominaux** (absence d'anomalies)

  Fichiers sources T_XXXX_N_k (k est un indice)

- **contrôles des cas d'anomalies**

  Fichiers sources T_XXXX_A_k

  XXXX désigne le nom de la classe cible

  Un ou plusieurs fichiers par méthode pour les cas nominaux

Organisation des tests unitaires (2)

Rôles distincts des modules suivant la catégorie :

- **contrôles des cas nominaux**

  Contrôler **chaque cas particulier** d'appel de chaque méthode (jeu de paramètres effectifs ad hoc)

  Contrôler le ou les **cas généraux** d'appel

- **contrôles des cas d'anomalies**

  Contrôler que chaque cause d'anomalie provoque **l'exception spécifiée** (ou le retour singulier)

Organisation des tests unitaires (3)

Le fichier source d'un module de test est constitué :

- d'une entête indiquant la cible des tests : classe, méthode, contexte opérationnel éventuel
- du corps de la **classe dédiée** qui supporte le module

Structuration d'un module (1)

Chaque classe dédiée (une par module) est constituée :

- de la méthode publique ***main***
- de blocs ({---}) internes à cette méthode
- de méthodes privées éventuelles pour gérer des contextes complexes

Structuration d'un module (2)

Un module de tests unitaires est une succession de "***test design***" parfaitement identifiés au sein du ***main***.

Chaque "***test design***" permet de vérifier un comportement majeur de la méthode cible.

Chaque "***test design***" produit un compte-rendu global de contrôle de la cible.

Les "***test design***" sont structurellement indépendants.

Chaque "***test design***" est un sous bloc interne Java ({---}).

Structuration d'un module (3)

Un "***test design***" est une succession de "***test case***" parfaitement identifiés.

Chaque "***test case***" permet de vérifier un sous-ensemble cohérent de comportements attachés à la méthode cible.

Chaque "***test case***" produit un compte-rendu local de contrôle des comportements cibles.

Les "***test case***" sont structurellement indépendants.

Chaque "***test case***" est un sous bloc interne Java ({---}).

Structuration d'un module (4)

Un "***test case***" est une succession de création de contextes d'exécution de la méthode cible et de "***test unit***".

Chaque "***test unit***" permet de vérifier :

- un résultat obtenu (par l'exécution de la méthode cible dans un des contextes préalablement créés)
- par rapport à un résultat attendu prédéfini (en fonction du contexte et fourni par le programmeur du test)

Niveaux préalables "***test design***" et "***test case***" obligatoires.

Frameworks pour les tests unitaires

➔ **Liés au langage de dév.** (plugins IDE)

- **Java** : **JUnit**, TestNG, Cactus (JEE), **Tests**
- **C/C++** : CUnit, Boost, Google Testing Fk, …
- **Python** : Unittest, PyUnit, …
- **PHP** : PHPUnit, Drupal, Atoum, Lime, …
- - - -

fr.wikipedia.org/wiki/Test_unitaire

La classe *Tests*

Développée au département informatique de l'IUT de Nice / UNS et exploitée dans d'autres établissements

Contribution de plusieurs promotions de DUT

Implémente les outils facilitant le respect de la charte décrite ci-avant :

- Codes sources fournis et étudiés en TD
- Usage fortement recommandé en TD&TP
- Non instanciable (classe **abstraite**)
- Mise en oeuvre très intuitive

Les méthodes de la classe ***Tests*** (1)

➔ **Begin**

- Usage obligatoire avant la description du premier test design
- Nom de la classe cible et version en paramètres

➔ **End**

- Usage obligatoire après la description du dernier test case du dernier test design
- Aucun paramètre

Les méthodes de la classe ***Tests*** (2)

➔ **Design**

- Appel obligatoire avant description premier *test case*
- Identification du *test design* et niveau de détail de la trace en visualisation en paramètres

➔ **Case**

- Appel obligatoire avant description premier *test unit*
- Identification du *test case* en paramètre

Les méthodes de la classe ***Tests*** (3)

➔ **Unit**

- **Valeur attendue** (classe dérivée de **Object**) et
- **Valeur constatée** (classe dérivée de **Object**)

Création du contexte d'exécution (1)

Le contexte d'exécution d'un test (***test_unit***) est constitué de l'ensemble des objets supports exploités par le test.

Le contexte d'exécution comprend :

- le ou les objets qui définissent **le résultat attendu**
- le ou les objets nécessaires à l'élaboration (par la méthode cible) du **résultat constaté**

Création du contexte d'exécution (2)

La création du contexte d'exécution d'un test est entièrement à la charge du programmeur

Les méthodes prédéfinies de la classe **Tests** automatisent :

- la comparaison entre chaque résultat attendu et le résultat constaté correspondant (méthode ***Unit***)
- la propagation du compte rendu vers les niveaux de tests supérieurs (méthodes ***Design*** et ***Case***)
- visualisation de tous les comptes rendus
- la production du rapport de test (méthode ***End***)

Création du contexte d'exécution (3)

➔ Les blocs d'instruction qui créent le contexte d'exécution de chaque ***test_unit*** ne doivent avoir aucune dépendance structurelle ou fonctionnelle entre eux.

➔ Des blocs d'instruction de niveau supérieur (test_case ou test_design) peuvent être utiles pour créer une partie commune (factorisation) à des contextes de niveau inférieur.

Création du contexte d'exécution (4)

➔ Dans l'hypothèse de blocs d'instructions de niveaux supérieurs, les instructions de niveaux inférieurs ne doivent pas modifier les objets communs (principe d'indépendance des *test_unit*)

Problème :

Comment concilier la dernière règle avec le test des méthodes qui modifient l'objet cible (accesseurs de modification par exemple) ?

Exemples de modules de tests unitaires

Cf fichiers sources de tous les tests unitaires des cas nominaux de la classe **Vecteur** (en annexe).

Bibliographie sur les tests

- ***Pratique des tests logiciels***

  JF Pradat-Peyre – Dunod Edition

- **Méthodologie ECSS - ESA**

Bibliographie sur les tests unitaires

- *fr.wikipedia.org/wiki/Test_unitaire*
- *morphm.ensmp.fr/attachments/35/UnitTest.pdf*
- *www.jmdoudoux.fr/java/dej/chap-frameworks-test.htm*
- *www.normalesup.org/~labatut/ED6/cours-6.pdf*
- *blog.xebia.fr/2008/04/11/les-10-commandements-des-tests-unitaires/*

Bibliographie pour la classe **Tests**

- Méthodologie ECSS - ESA
- Note technique sur la charte des tests unitaires
- Codes sources de la classe **Tests** fournis (en annexe)

Bibliographie pour **JUnit**

- *www.jmdoudoux.fr/**java**/dejae/chap011.htm*
- *www.liafa.jussieu.fr/~sighirea/cours/methtest/c_JUnit.pdf*
- *gfx.developpez.com/tutoriel/java/junit*
- *java.cnam.fr/iagl/glg203/cours/JUnit.pdf*
- *humbert-florent.developpez.com/java/EDI/junitsjcreator/*
- *http://en.wikipedia.org/wiki/JUnit*
- *http://tutorials.jenkov.com/java-unit-testing/index.html*
- *http://howtodoinjava.com/junit/*
- *www.cs.le.ac.uk/people/hqy1/JUnit%20Tutorial.ppt*