IUT Nantes
Pôle Sciences et technologie

Nantes Université

R2.03 - Qualité de développement 1
Automatisation des tests

Jean-Marie Mottu
BUT info 1 – IUT Nantes

Organisation de CO-Test

- Volume
  - 5 cours magistraux
  - 5 TD de 2h40 (plutôt que des TP contrairement au livret)
- Enseignants
  - Jean-Marie Mottu, responsable
  - Sébastien Fauvel, Gustavo Cipriano Mota Sousa, Solen Quiniou
- Notation en contrôle continu :
  - QCM en amphi
  - Note(s) pratique(s) pendant les séances
  - Test pratique sur machine en fin de période
  - Contrôle fin de période

Plan dans le BUT

- Prolongement direct des modules
  - R2.01 - Développement orienté objets
  - R2.10 - Gestion de Projet Informatique GPO2

- Un des 4 ressources de Qualité de Développement:
  - R2.03 – QD1 Automatisation des tests
  - R3.04 – QD2 Conception avancée
  - R4.02 – QD3 Conception des tests
  - R5.A.08 – QD4 Tests dans les cycles de développement

Plan de R2.03 - Qualité de développement 1 Automatisation des tests

- Sensibilisation à la production de tests unitaires
  - Introduction au test logiciel
  - Typologie
- Automatisation de tests unitaires
  - Programmation de test unitaire
  - Programmation de suite de tests unitaires
- Première approche de la gestion des cas d'erreurs
  - Test fonctionnel
  - Gestion des exceptions
- Traces et utilisation d'outils de débogage
- Utilisation d'un outil de gestion de versions

- Problématique de la non-régression
  - tout aura été fait avec la rigueur permettant le test de non-régression

« L'erreur est humaine » et les informaticiens sont (encore) des humains

- De potentiels problèmes de qualité partout
  - On définit son besoin
  - On gère son projet
  - On conçoit son système
  - On développe son système
  - On contrôle où on s'est trompé
  - On corrige
  - On recommence

On se trompe (toujours !)

Faute, erreur, défaillance

- « le développeur se trompe (mistake) en introduisant
  une faute (fault) dans son code dont
  l'exécution sera erronée (error) et
  risque d'entraîner une défaillance (failure) »
- Cela est valable pour le code, la doc, la BDD, l'interface…

- Faute : partie du système incorrecte
- Erreur : la faute provoque un comportement erroné
- Défaillance : l'erreur a des conséquences sur le fonctionnement du système

- En testant, on cherche les fautes en provoquant les erreurs (pour éviter les défaillances)

Exemple faute/erreur/défaillance

- « A7 : des automobilistes coincés au péage, à cause du changement d'heure »

https://www.francebleu.fr/infos/insolite/a7-coinces-au-peage-en-pleine-nuit-a-cause-du-changement-d-heure-1509361121

« Plusieurs automobilistes se sont retrouvés coincés au péage de Valence-Nord, sur l'A7, dans la nuit de samedi à dimanche. A cause d'un bug inédit chez Vinci Autouroutes [sic] lié au changement d'heure, certains ont dû patienter près d'une heure avant de forcer les barrières. » (31 octobre 2017)

- Supposition :
  - Faute : le calcul de durée ne considère pas le changement d'heure
  - Erreur : probablement qu'une vérification de cohérence de durée échoue
  - Défaillance : les barrières restent fermées et les usagers coincés comme des fraudeurs

Essayons une potentielle fonction fraude

Peage.java

```java
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;

public class Peage {

    public static boolean fraude(float distance, LocalDateTime entryHour, LocalDateTime exitHour) {
        System.out.println("Less than "+distance/180*60+" minutes would be a fraud");
        long durationInMinutes = ChronoUnit.MINUTES.between(entryHour, exitHour);
        System.out.println("Duration " + durationInMinutes + " min");
        return durationInMinutes<distance/180*60;
    }

    public static void main(String[] args) {
        LocalDateTime entryHourTest = LocalDateTime.of(2017, 10, 29, 12, 30, 0);
        LocalDateTime exitHourTest = LocalDateTime.of(2017, 10, 29, 13, 37, 0);
        float distanceTest = 112; // Nantes - Niort 112km
        System.out.println("Fraud ? " + Peage.fraude(distanceTest, entryHourTest, exitHourTest));

        entryHourTest = LocalDateTime.of(2017, 10, 29, 12, 30, 0);
        // Si on s'arrête au péage pour aller récupérer un ticket à l'entrée avant de sortir
        exitHourTest = LocalDateTime.of(2017, 10, 29, 12, 37, 0);
        distanceTest = 112; // Nantes - Niort 112km
        System.out.println("Fraud ? " + Peage.fraude(distanceTest, entryHourTest, exitHourTest));
    }
}
```

Console

<terminated> Peage [Java Application] C:\Users\admin-user\.p2\pool\plugins\org.e

```
Less than 37.333336 minutes would be a fraud
Duration 67 min
Fraud ? false
Less than 37.333336 minutes would be a fraud
Duration 7 min
Fraud ? true
```

Ca semble fonctionner :

Essayons une potentielle fonction fraude
Quid du changement d'heure ?

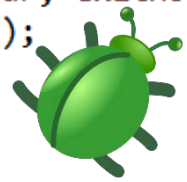
Peage.java

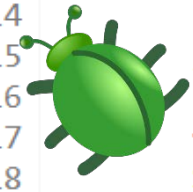
```java
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;

public class Peage {

    public static boolean fraude(float distance, LocalDateTime entryHour, LocalDateTime exitHour) {
        System.out.println("Less than "+distance/180*60+" minutes would be a fraud");
        long durationInMinutes = ChronoUnit.MINUTES.between(entryHour, exitHour);
        System.out.println("Duration " + durationInMinutes + " min");
        return durationInMinutes<distance/180*60;
    }

    public static void main(String[] args) {
        LocalDateTime entryHourTest = LocalDateTime.of(2017, 10, 29, 2, 30, 0);
        // tic toc : passage à l'heure d'hiver à 3h on revient à 2h
        LocalDateTime exitHourTest = LocalDateTime.of(2017, 10, 29, 2, 37, 0);
        float distanceTest = 112; // Nantes - Niort 112km
        System.out.println("Fraud ? " + Peage.fraude(distanceTest, entryHourTest, exitHourTest));
    }
}
```





Console

<terminated> Peage [Java Application] C:\Users\admin-user\.p2\pool\plugins\org.ec

```
Less than 37.333336 minutes would be a fraud
Duration 7 min
Fraud ? true
```

Le résultat est erroné :
une fraude annoncée alors
qu'il y a bien eu 1h07 de trajet

Qualité logicielle

- Software Engineering: A Practitioner's Approach. Roger Pressman. 1st (1982), 8th edition (2014)
  - «Conformité aux besoins explicites de fonctionnalité et de performance, aux normes de développement explicitement documentées et aux caractéristiques implicites qui sont attendues de tous les logiciels développés de façon professionnelle» (5ème édition)

- [IEEE]
  - Le degré pour lequel un système, un composant ou un processus répond aux besoins spécifiés.
  - Le degré pour lequel un système, un composant ou un processus répond aux attentes des utilisateurs.

Conséquences de défaillances

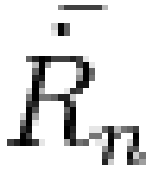
- Défaillances
  - Mauvais résultat, mauvais comportement
  - Plantage
  - Crash
- Implique des pertes
  - Image
  - Financier
  - Humain

Perte d'image

- Un bug à la mise à jour des serveurs d'une caméra de vidéosurveillance d'intérieur permet d'accéder à d'autres caméras que la sienne
  - Cela vous donne-t-il envie d'acheter ces caméras ?
  - https://www.clubic.com/domotique/video-surveillance/actualite-371913-cameras-eufy-une-mise-a-jour-a-rendu-les-flux-video-accessibles-a-d-autres-utilisateurs.html

Coût financier

- Mariner 1 (1962)
  - **Cost**: $18.5 million
  - **Disaster**: The Mariner 1 rocket with a space probe headed for Venus diverted from its intended flight path shortly after launch. Mission Control destroyed the rocket 293 seconds after liftoff.



  - **Cause**: A programmer incorrectly transcribed a handwritten formula into computer code, missing a single superscript bar. Without the smoothing function indicated by the bar, the software treated normal variations of velocity as if they were serious, causing faulty corrections that sent the rocket off course.

Coût des erreurs

- 2020, Consortium for Information and Software Quality :
  - les défauts de qualité logicielle aux Etats-Unis auraient coûté 2 080 Md$ en 2020
  - augmentation des dysfonctionnements opérationnels de 22 % sur 2 ans
  - préconise une analyse précoce et régulière du code
    - Il faut TESTER

https://www.lemondeinformatique.fr/actualites/lire-la-mauvaise-qualite-logicielle-a-coute-2-080-md$-aux-etats-unis-en-2020-81614.html

- Plus un bug est détecté tard, plus son coût est décuplé
  - Bug détecté dans un code qu'on vient d'écrire : quelques minutes perdues.
  - Rappel de 625 000 voitures Toyota en 2015 : quelques millions d'euros

Coût humain

- Système de missiles Patriot (1991)
  - Cost: 28 soldiers dead, 100 injured
  - Disaster: During the first Gulf War, an American Patriot Missile system in Saudi Arabia failed to intercept an incoming Iraqi Scud missile. The missile destroyed an American Army barracks.
  - Cause: A software rounding error incorrectly calculated the time, causing the Patriot system to ignore the incoming Scud missile.

Pas loin de chez nous

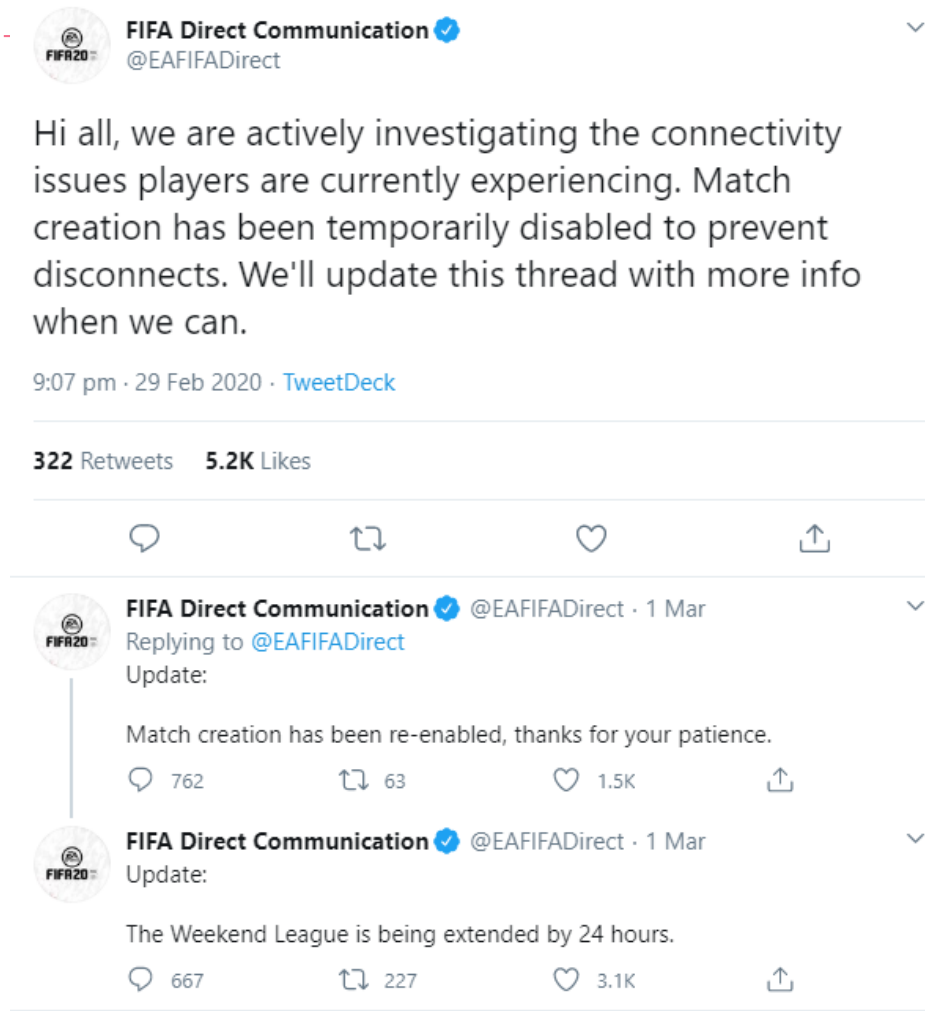
- le 29 février 2012 met le bazar dans cette semaine d'emploi du temps dans CELCAT

- le 30 juin 2012 il fallait reculer les montres d'une seconde: les comptes de l'Université de Nantes reçoivent une demande de mise à jour des mots de passe pourtant nécessaire seulement tous les 6 mois

- En 2022, l'application d'agenda zimbra de l'université à un bug qui ne permet pas d'exiger une authentification pour partager son agenda avec un tiers.

Fautes non révélées
Fautes récurrentes

- Beaucoup de fautes ne sont pas reconnues
  - Parfois masquées
    - e.g. deadline atteinte, correctifs coûteux
- D'autres temporaires n'ont pas d'explication officielle
- Iphone : passage à l'heure hiver/été:
  2018, 2014, 2010
- IWatch 2018
- Pixel : 2021
- Etc.



Vérification et Validation par le test

- Vérification
  - _____
- Validation
  - _____
- Test
  - Principe général : faire des essais
    - Combien ? De quel type ? A quel point ?
  - Différent de la preuve qui consiste à formaliser le système pour appliquer des vérifications de niveau mathématique
    - Difficulté de la formalisation
    - Preuve et test sont complémentaires

Objectifs

- Le test a pour but de

_____

dans un programme conformément à sa spécification

  - Prouver l'absence de fautes dans un programme est un problème indécidable dans le cas général
- Motivation
  - Diminuer le coût d'un logiciel
  - Augmenter la qualité
    - Augmenter la confiance

Cas de test

- Tester c'est *effectuer* un ensemble de cas de test
- Cas de test
  - Description du cas de test
  - Initialisation
  - Donnée de test
  - Oracle
    - L'Oracle contrôle l'exécution du SUT initialisé, avec la DT, retournant le verdict
- Verdict : passe, échoue

Exemples de 2 cas de test

- Description : test d'un passage péage normal
- Initialisation : lancement du système
- Donnée de test : 112km, dans la journée du 29/10/2017
- Oracle : ne doit pas annoncer une fraude

=> l'exécution de ce test est un succès

- Description : test d'un passage péage au moment du changement d'heure
- Initialisation : lancement du système
- Donnée de test : 112km, le 29/10/2017, entre 2h (été) et 2h07 (hiver)
- Oracle : ne doit pas annoncer une fraude

=> l'exécution de ce test est un échec (révélant un bug)

Problématique du test

- On ne peut pas tester tout le temps ni tous les cas de test possibles
  - Le test exhaustif est impossible : e.g. le nombre de date est infini)
  - Il faut des critères pour choisir les cas intéressants en quantité appropriée
  - Il faut des heuristiques réalistes
    - Formalisation de critères pour guider la sélection des cas de tests qui ont le plus de propension à détecter des fautes
- Sans technique ni outil, le test serait extrêmement laborieux
  - Ne pas savoir où chercher les bugs potentiels
  - Passer du temps sans savoir quand s'arrêter
  - Gaspiller du temps à faire de nouveaux tests sans intérêt
  - etc.

Le test – Définition Générale

_____ pour _____ si ça _____ ____.

Apprendre
- pourquoi c'est fait
- ce que ça doit faire
- comment c'est fait
- comment ça marche

Modéliser

S'en faire une idée

Exécuter

Analyser

Qu'y a-t-il à observer?

Que faut-il regarder?

Qu'est-ce qui est visible?

Qu'est ce qu'on cherche?

Comment le regarder?

Qu'est ce qui devrait fonctionner ?

Identifier une erreur

Diagnostiquer une erreur

Catégoriser ces erreurs

Ca peut fonctionner, mais assez vite ?

Qu'est-ce qu'on teste ?
(quelles propriétés?)

- Les propriétés du système sous test (SUT – System Under Test) :
  - fonctionnalités
  - sécurité / intégrité
  - utilisabilité
  - cohérence
  - maintenabilité
  - efficacité
  - robustesse
  - sûreté de fonctionnement

- de manière générale en vérification :

la conformité à la spécification

Sur quoi baser le test ?

- Une spécification : exprime ce qu'on attend du système
  - un cahier des charges (en langue naturelle)
  - une documentation
  - des échanges avec le client
  - commentaires dans le code
  - contrats sur les opérations (à la Eiffel)
  - un modèle UML
  - une spécification formelle (automate, modèle B...)

- La référence c'est la spécification
  - Et pas le code potentiellement bogué

IUT Nantes
Pôle Sciences et technologie

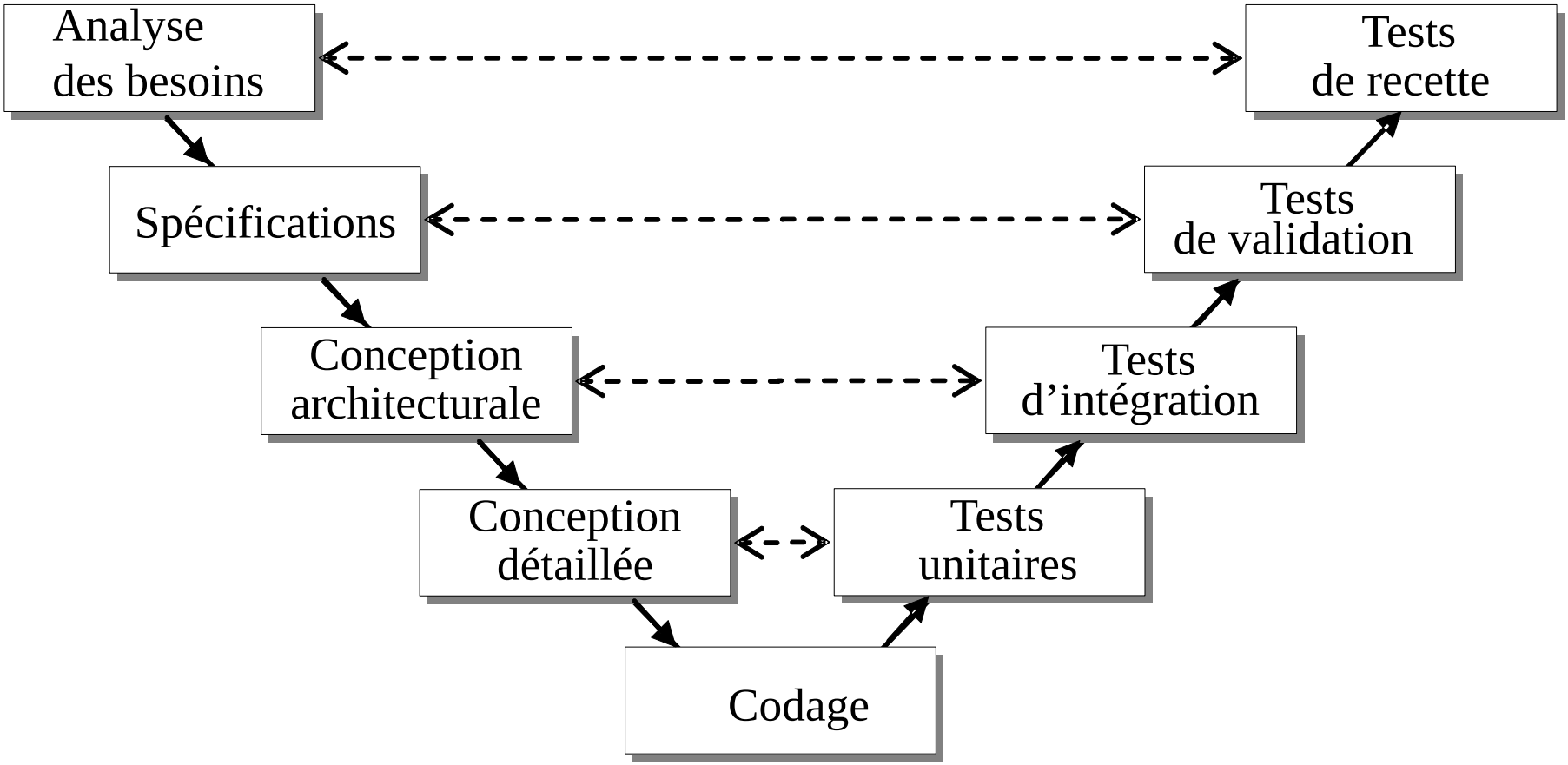
Nantes Université

Typologie

Test de logiciel : terminologie

- Etapes de test
  - Unitaire, intégration, système, recette
- Phases transversales
  - non régression, performance
- Différente dynamicité, les tests peuvent être
  - Dynamique / statique
- Création de tests avec une approche
  - Fonctionnelle / structurelle

Le test dans un cycle de développement en V



Etapes de test : Test unitaire

- Vérification d'une unité indépendamment des autres
- Vérifier intensivement les unités

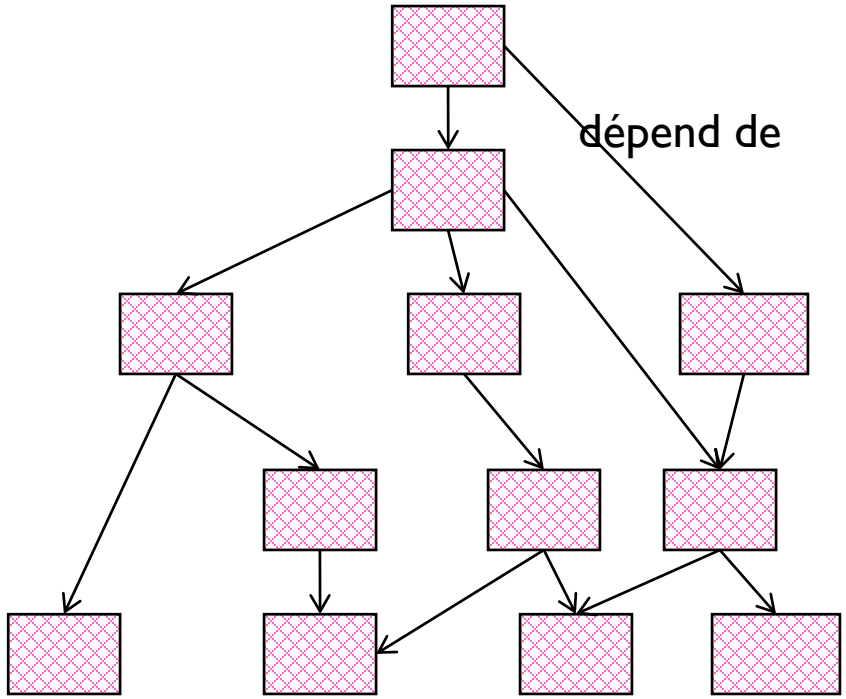
- Pour un langage procédural
  - unité de test = procédure
- Dans un contexte orienté objet
  - unité de test = classe

Etapes de test : Test d'intégration

- Choisir un ordre pour intégrer et tester les différents modules du système

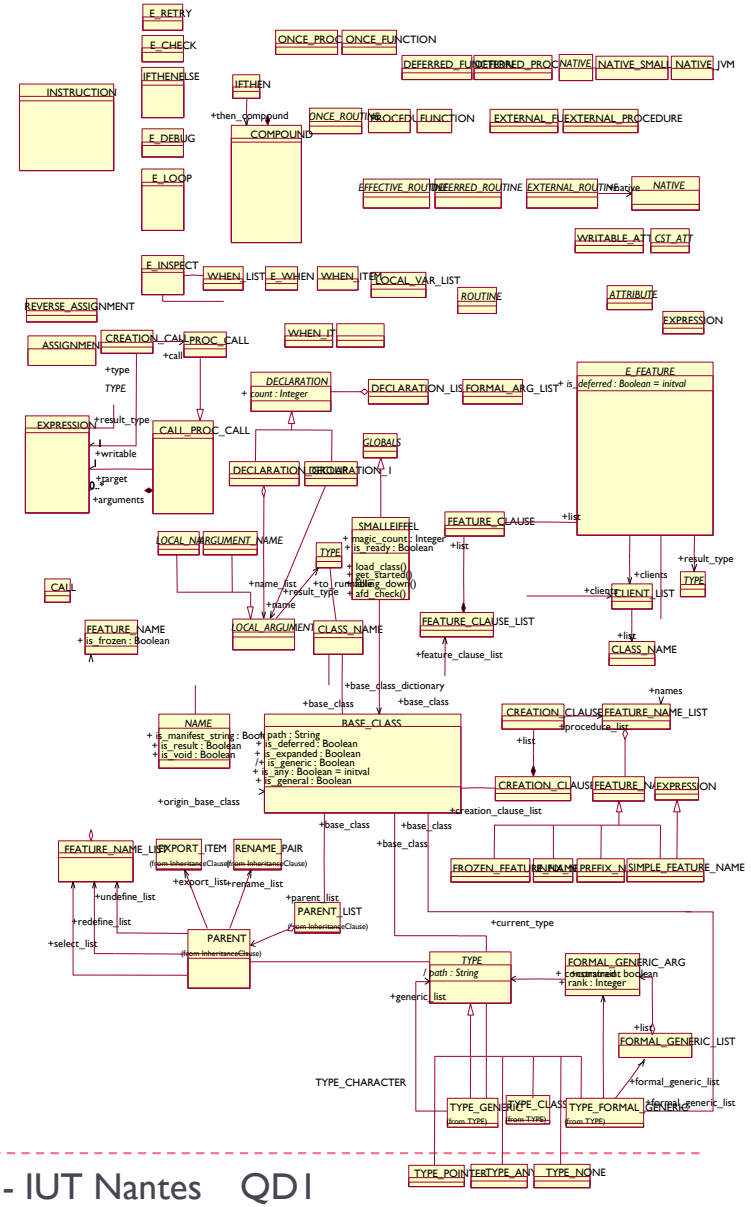
- Cas simple: il n'y a pas de cycle dans les dépendances entre modules

- Les dépendances forment un arbre et on peut intégrer simplement de bas en haut



Etapes de test : Test d'intégration

- Cas plus complexe: il y a des cycles dans les dépendances entre modules
- Cas très fréquent dans les systèmes à objets
- Il faut des heuristiques pour trouver un ordre d'intégration



Etapes de test :

- Test de validation ou système
  - Valider la globalité du système
    - Les fonctions offertes
    - A partir de l'interface
  - Remet en cause les exigences, la spécification

- Test recette
  - Effectué avec la MOA ou par la MOA
  - Test avec utilisateur final

Phases transversales de test :

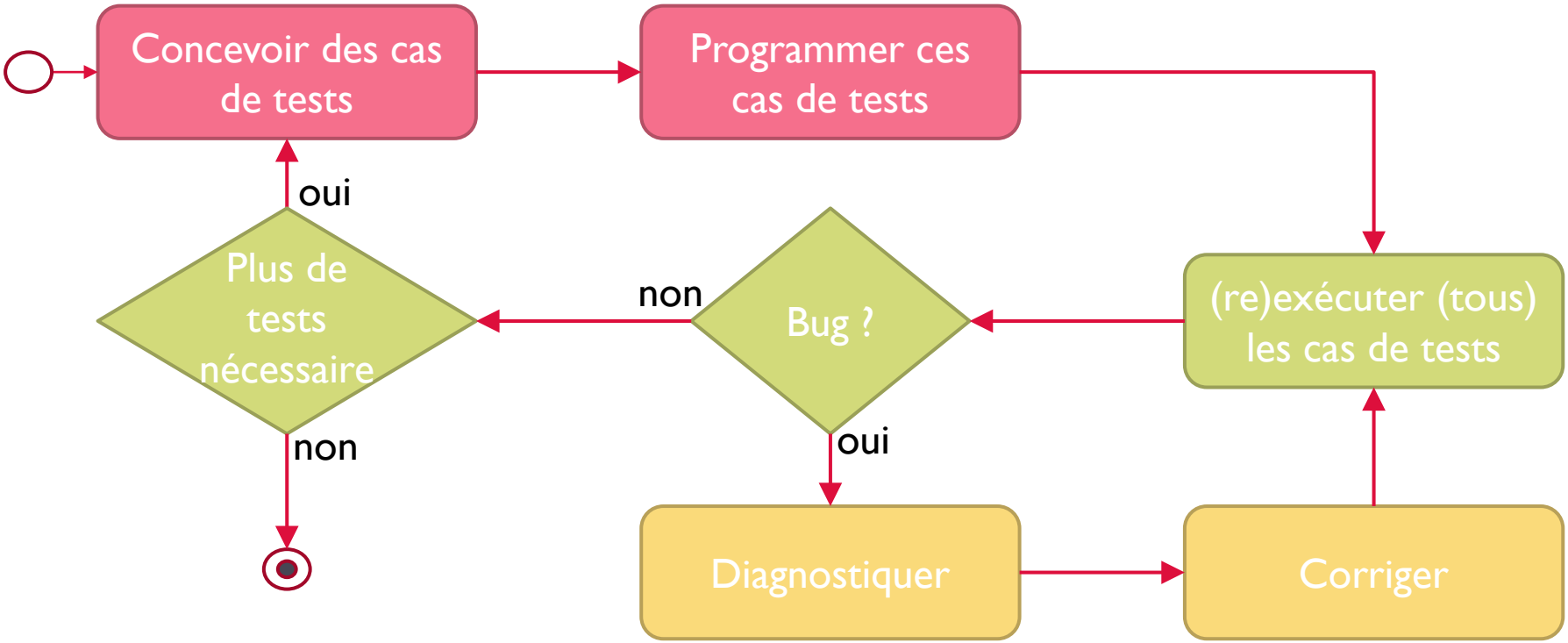
- Test de non régression
  - Consiste à vérifier que des modifications apportées au logiciel n'ont pas introduit de nouvelle erreur
    - vérifier que ce qui marchait marche encore
  - Dans la phase de maintenance du logiciel
    - Après refactoring, ajout/suppression de fonctionnalités
  - Après la correction d'une faute
- Test de performance
- Etc.

Quelle exécution des tests ?

- Test statique
  - relecture / revue de code
  - analyse automatique
    - vérification de propriétés, règles de codage...

- Test dynamique
  - on exécute le programme avec des données de test en entrée et on contrôle le comportement

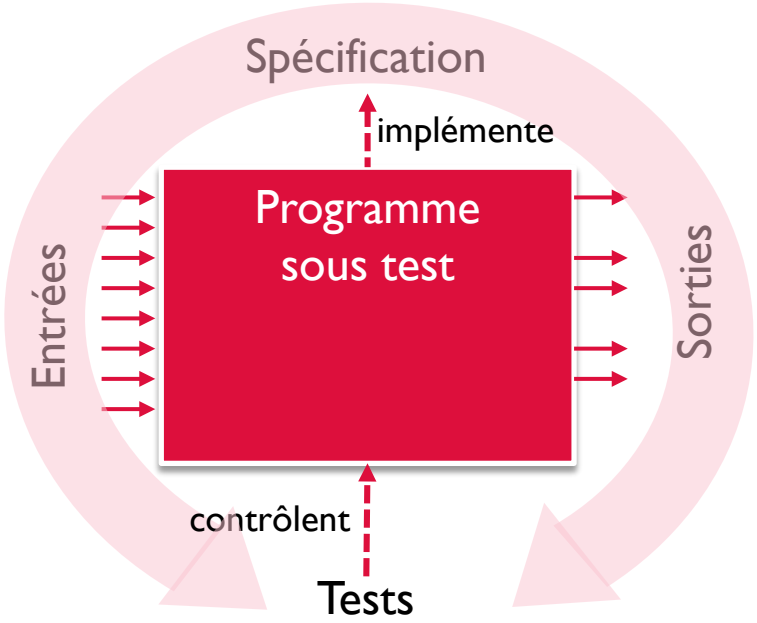
Le cycle de test dynamique



Création de tests avec une approche :

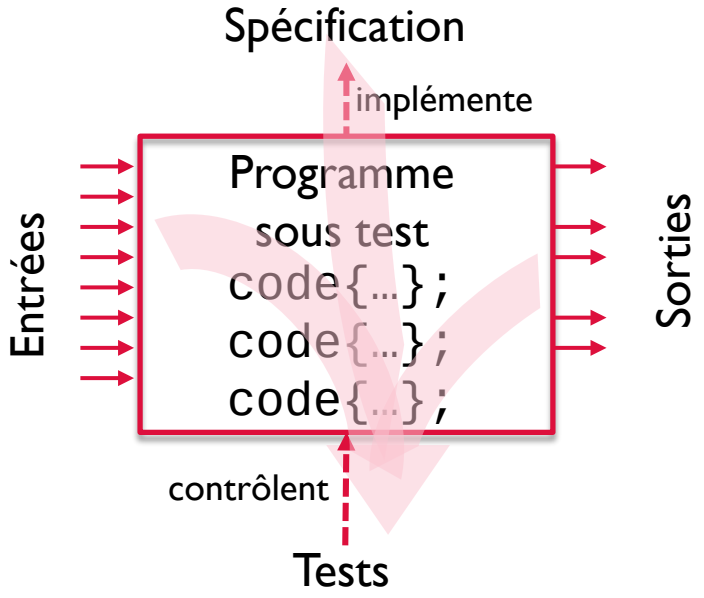
Test fonctionnel (test boîte noire)

Exploite la description des fonctionnalités du programme

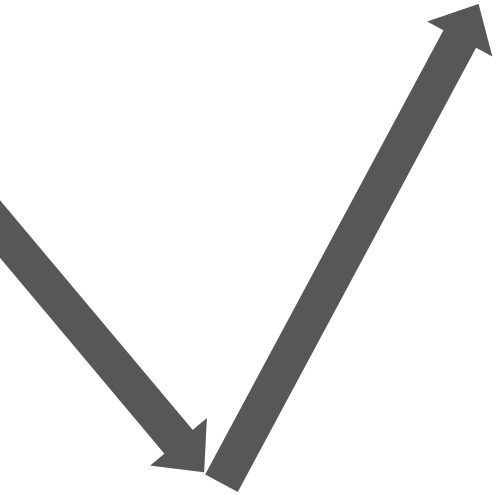


Test structurel (test boîte blanche)

Exploite la structure interne du programme

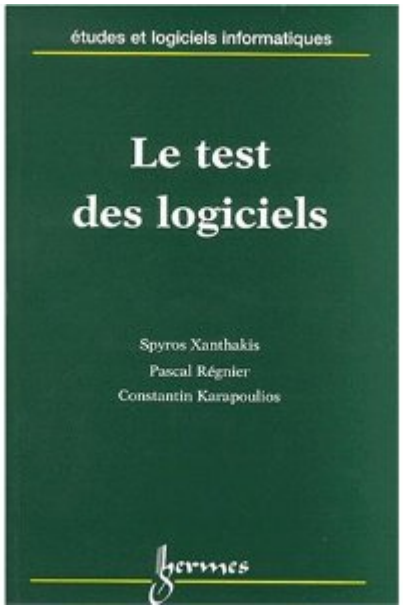


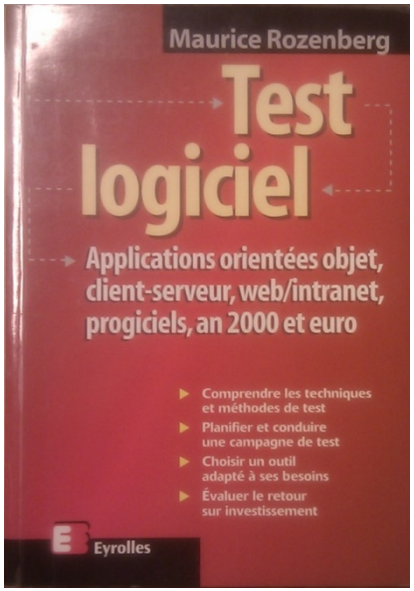
Etapes et Approches des tests

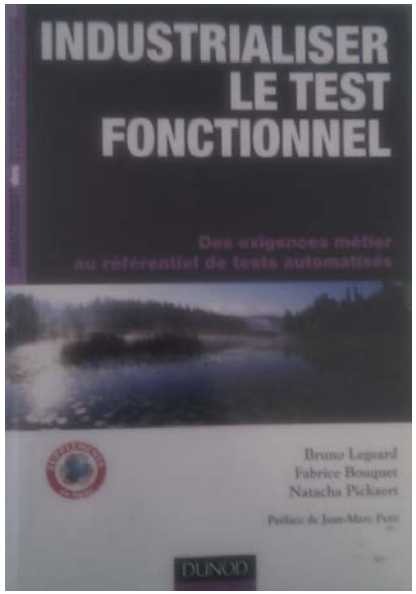


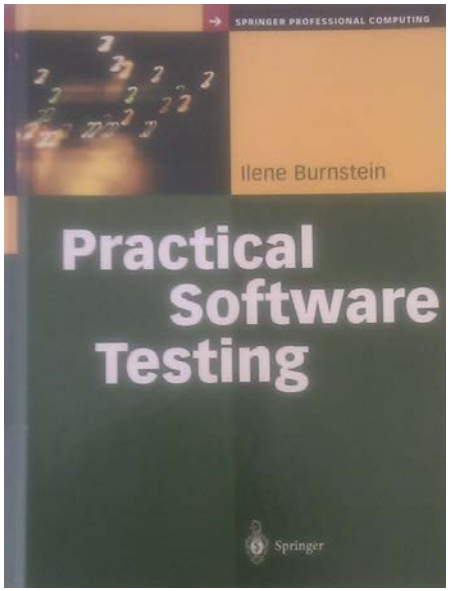
Approche fonctionnelle	***Tests de validation***
Approche structurelle	***Tests d'intégration***
	Tests unitaires

Quelques livres









- Autres sources de cours, des collègues :
  - Yves Le Traon, Benoit Baudry, Gerson Sunye