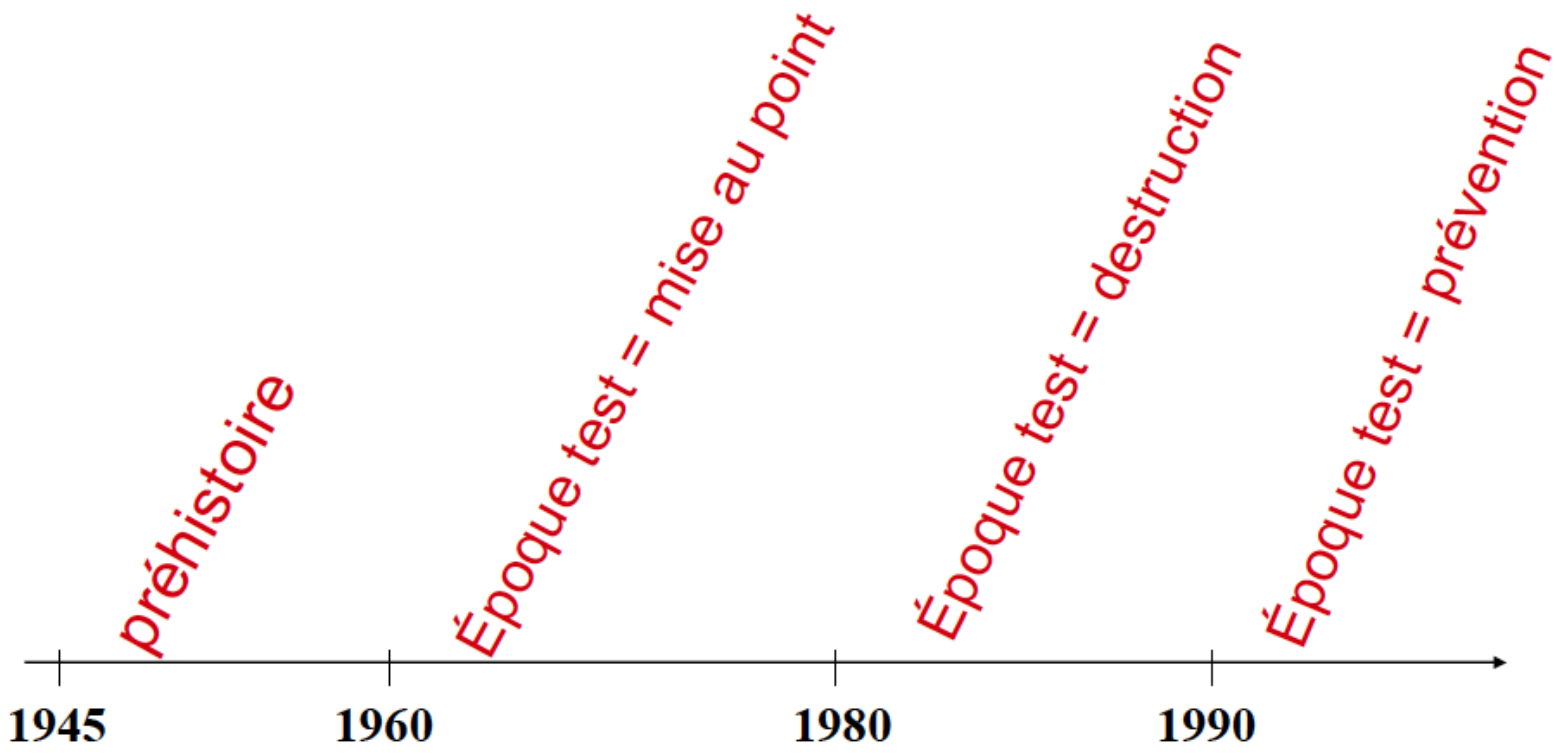
Conception Avancée

Cours 3 : Test Logiciel, Validation et Vérification

Nadjib Lazaar (lazaar@lirmm.fr) IUT de Montpellier 2018/2019

historique et problématique du test



1960-80 : Test = mise au point

Qu'avons-nous compris depuis ?

Chaîne causale : erreur ➔ faute ➔ défaillance

En fait, 3 activités distinctes :

- détection de défaillances (rôle du test)
- localisation de fautes (rôle de la mise au point)
- correction des erreurs (rôle de la mise au point)

N. Lazaar (IUT de Montpellier)
lazaar@lirmm.fr

1980-90 : Test = destruction

« Testing is the process of executing a program
with the intent of finding errors »

[G. Myers The Art of Software Testing **1979]**

Conséquence immédiate :

le testeur ne doit pas être le programmeur

Position dogmatique progressivement abandonnée !

1990-aujourd'hui : Test = prévention

« Se convaincre, par des techniques d'analyse ou d'exécution, qu'un programme répond à ses spécifications »

- Analyse ➔ Contrôle : vérifier des propriétés

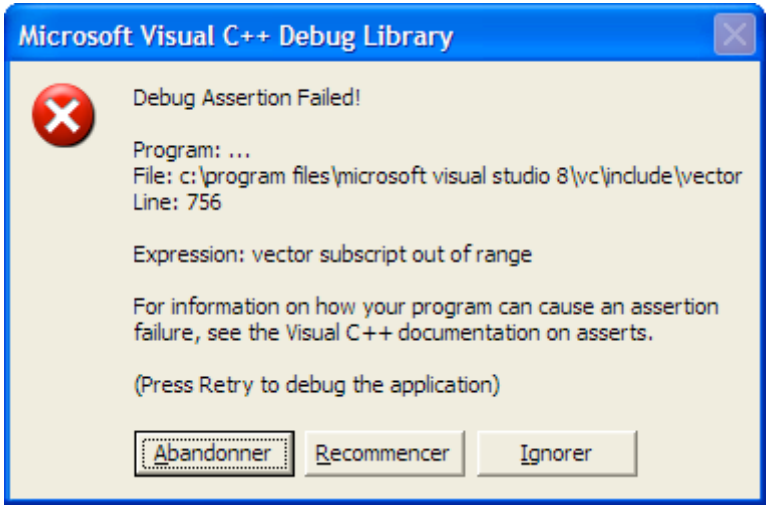
  avant exécution

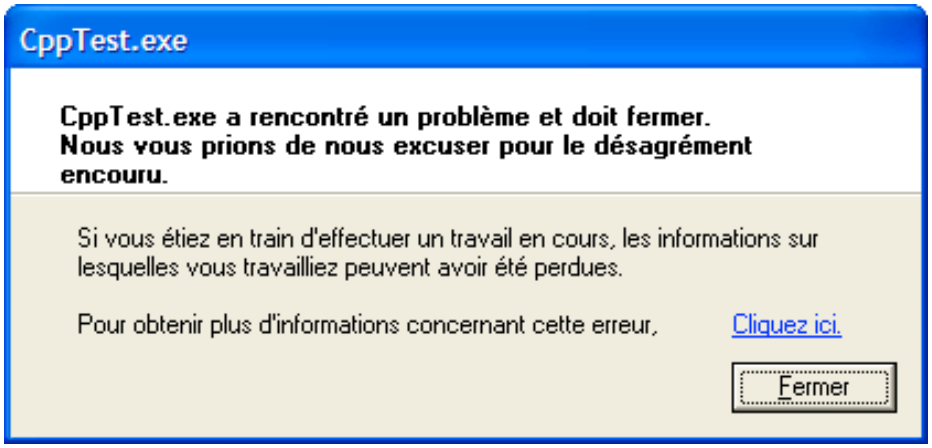
- Exécution ➔ Test : évaluer un résultat

  après exécution

N. Lazaar (IUT de Montpellier)
lazaar@lirmm.fr


Erreur Adobe Premiere Pro
Une erreur s'est produite dans Adobe Premiere Pro.
[\dev\stingray\MediaLayer\Src\Audio\MonitorMixer.cpp-147]
Continuer


Microsoft Visual C++ Debug Library
Debug Assertion Failed!
Program: ...
File: c:\program files\microsoft visual studio 8\vc\include\vector
Line: 756
Expression: vector subscript out of range
For information on how your program can cause an assertion failure, see the Visual C++ documentation on asserts.
(Press Retry to debug the application)
Abandonner
Recommencer
Ignorer


CppTest.exe
CppTest.exe a rencontré un problème et doit fermer.
Nous vous prions de nous excuser pour le désagrément encouru.
Si vous étiez en train d'effectuer un travail en cours, les informations sur lesquelles vous travailliez peuvent avoir été perdues.
Pour obtenir plus d'informations concernant cette erreur,
Cliquez ici.
Fermer

Bugs célèbres

Sonde Mariner 1, 1962

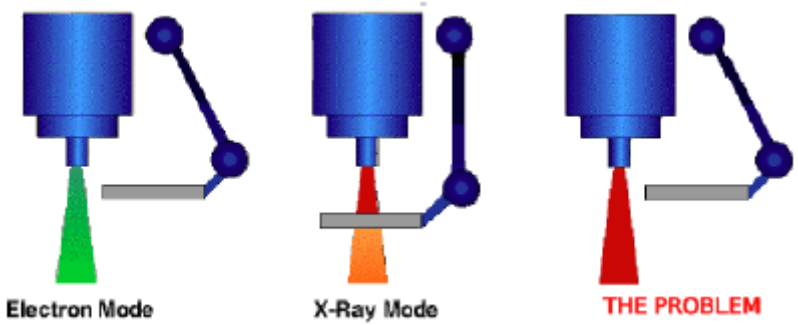
- Détruite 5 minutes après son lancement
- Coût : 18,5 millions de dollars
- Défaillance des commandes de guidage due à une erreur de spécification
- Erreur de transcription manuelle d'un symbole mathématique dans la spécification

http://fr.wikipedia.org/wiki/Mariner_1

Bugs célèbres

Therac-25, 1985-87

- Au moins 5 morts par dose massive de radiations
- Problème d'accès concurrents dans le contrôleur



http://fr.wikipedia.org/wiki/Therac-25

Processeur Pentium, 1994

- Bug dans la table de valeurs utilisée par l'algorithme de division

http://fr.wikipedia.org/wiki/Bug_de_la_division_du_Pentium

Bugs célèbres

Ariane V vol 501, 1996

- Explosion après 40 secondes de vol
- Coût : 370 millions de dollars
- Panne du système de navigation due à un dépassement de capacité (arithmetic overflow)
- Réutilisation d'un composant d'Ariane IV non re-testé

http://fr.wikipedia.org/wiki/Vol_501_d'Ariane_5

Pourquoi vérifier et valider ?

Pour éviter les bugs

- Pour l'utilisateur : coût économique, humain, environnemental
- Pour le fournisseur : coût de la correction des bugs

- en phase d'implantation → coût 1
- en phase d'intégration (bug de conception) → coût 10
- en phase de recette (bug de spécification) → coût 100
- en phase d'exploitation → coût > 1000

N. Lazaar (IUT de Montpellier)
lazaar@lirmm.fr

Pourquoi vérifier et valider ?

Pour assurer la qualité

- Capacité fonctionnelle : réponse aux besoins des utilisateurs
- Facilité d'utilisation : prise en main et robustesse
- Fiabilité : tolérance aux pannes
- Performance : temps de réponse, débit, fluidité...
- Maintenabilité : facilité à corriger ou transformer le logiciel
- Portabilité : aptitude à fonctionner dans un environnement différent de celui prévu

Pourquoi des méthodes pour vérifier et valider ?

Pour réduire le coût

- Vérification et validation :
  - environ 30% du développement d'un logiciel standard
  - plus de 50% du développement d'un logiciel critique

- Phase de test souvent plus longue que les phases de spécification, conception et implantation réunies

Test de programme : définition

Tester = exécuter un programme P pour mettre en évidence la présence de fautes, par rapport à sa spécification F

Recherche de contre-exemples :

$$\exists X \text{ tq } P(X) \neq F(X) \ ?$$

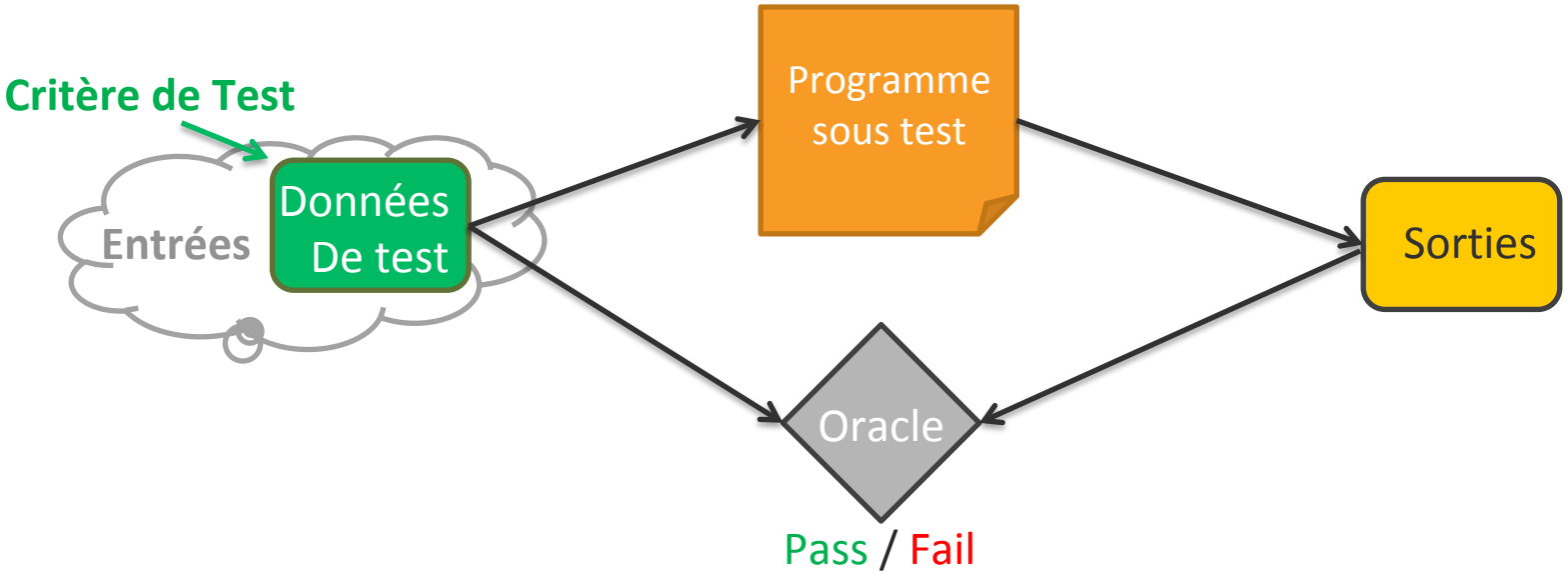
développeur expérimenté ➔ 1 faute / 10 lignes de code

163 fautes / 1000 instructions

[B. Beizer Software Testing Techniques **1990]**

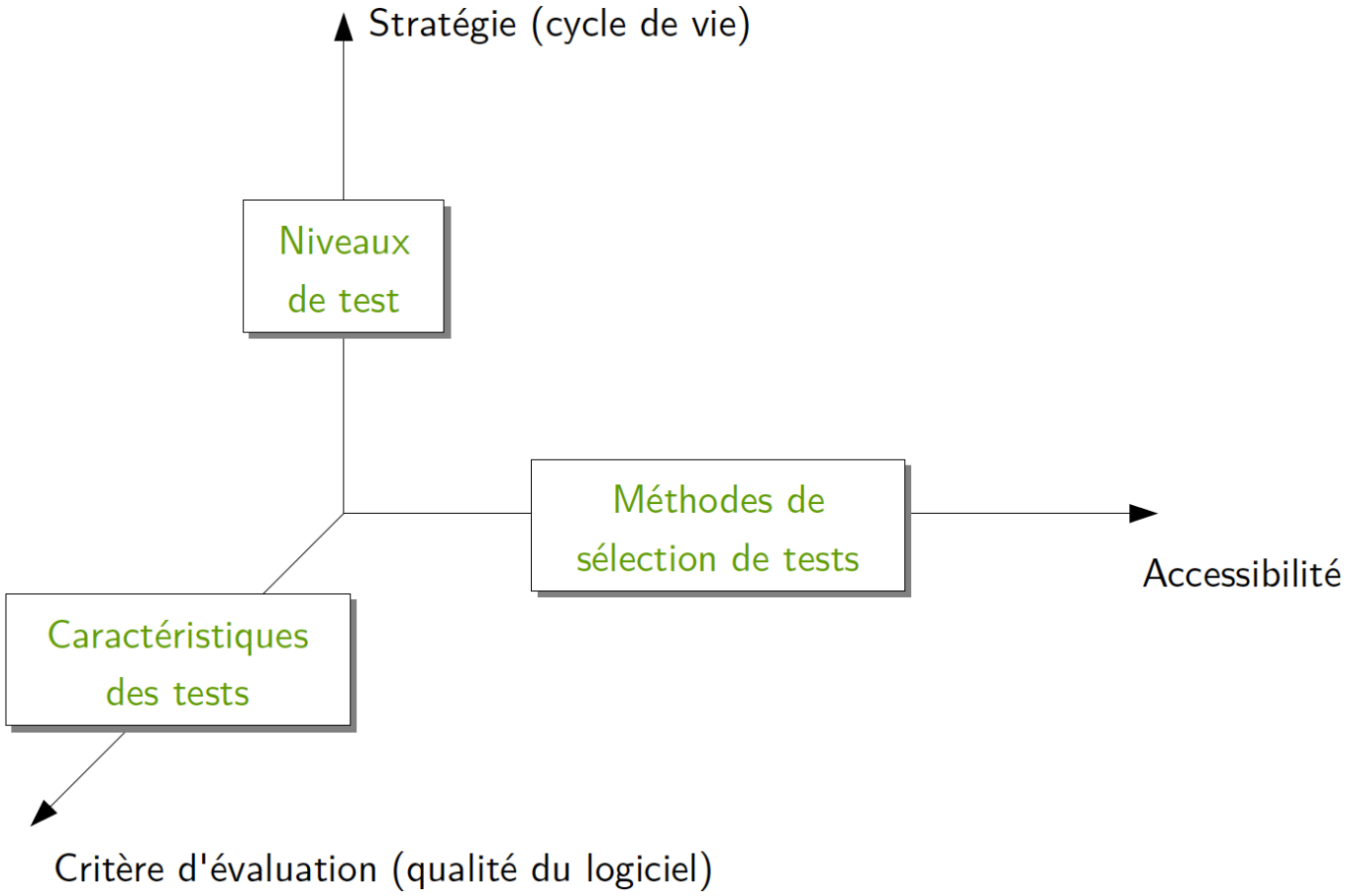
N. Lazaar (IUT de Montpellier)
lazaar@lirmm.fr

Processus de Test

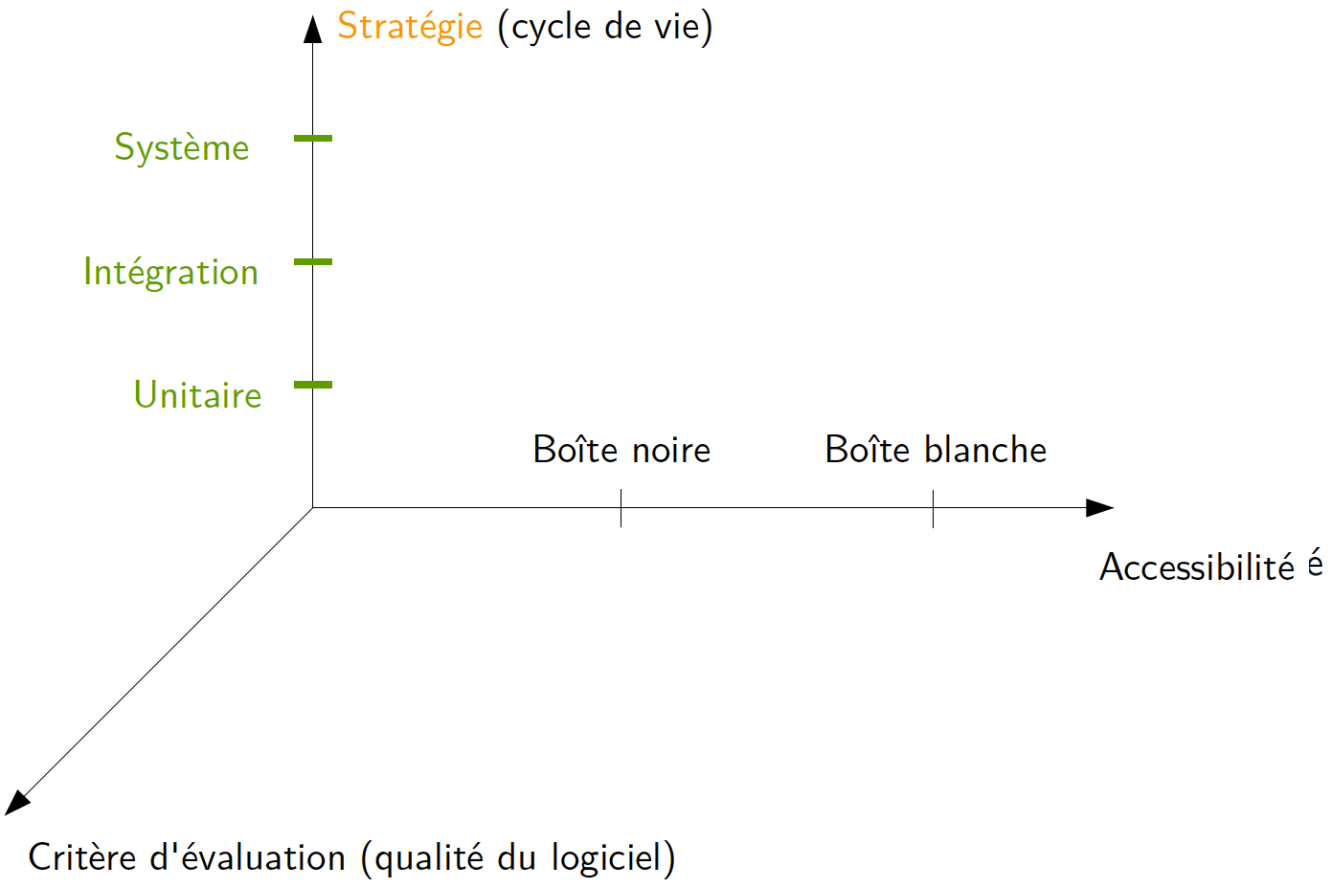


1. Choisir les comportements à tester (programme sous test)
2. Choisir des données de test permettant de déclencher ces comportements + décrire le résultat attendu pour ces données
3. Exécuter les cas de test sur le système + collecter les résultats
4. Comparer les résultats obtenus aux résultats attendus pour établir un verdict

Types de Test

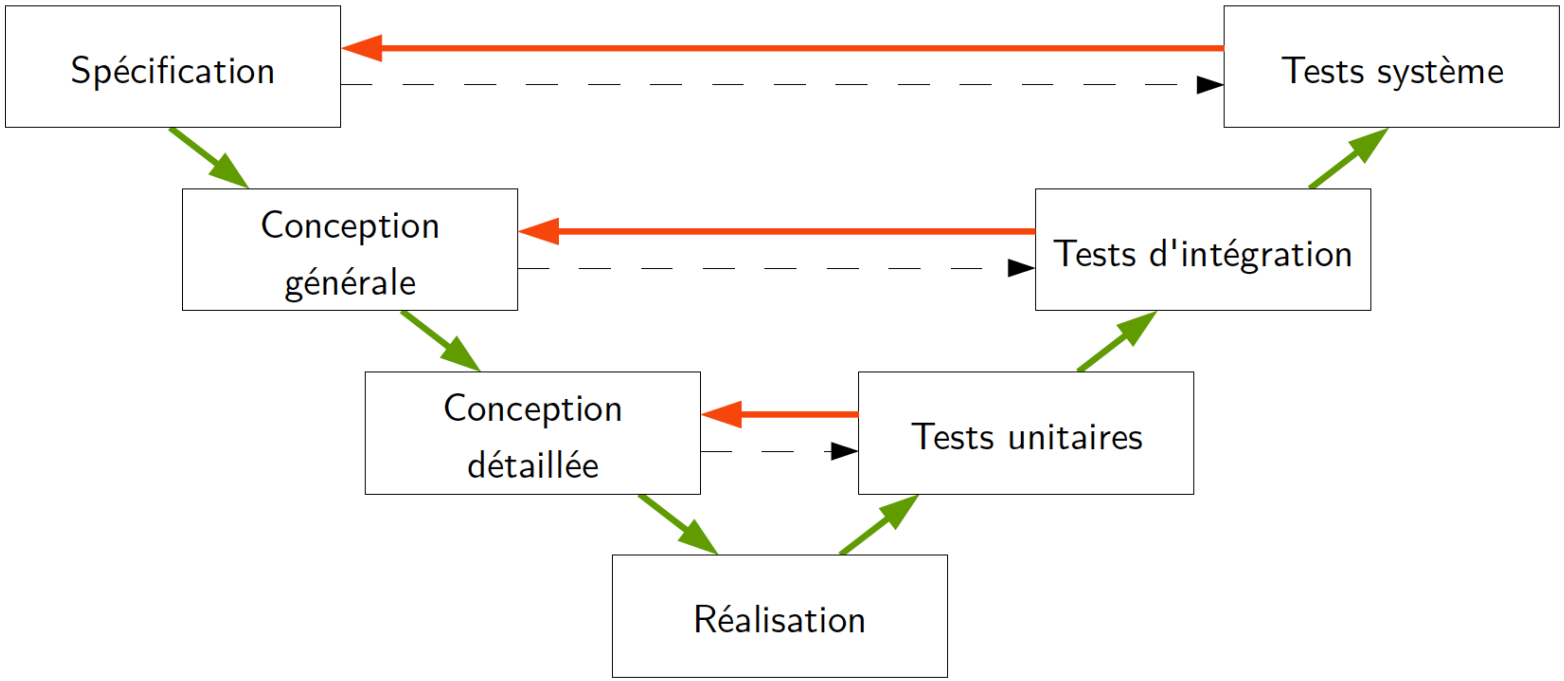


Types de Test



N. Lazaar (IUT de Montpellier)
lazaar@lirmm.fr

Cycle de vie du logiciel



Test unitaire

Test des unités de programme de façon isolée,
indépendamment les unes des autres, c'est-à-dire sans appel à une fonction d'un autre module, à une base de données...

➔ méthodes, classes, modules, composants

Exemple : Commerce électronique

- Supprimer un article d'un panier vide

Test d'intégration

Test de la composition des modules via leur interface

➔ communications entre modules, appels de procédures...

Exemple : Commerce électronique

- Chainage des vues de l'IHM
- Saisie sécurisée des données

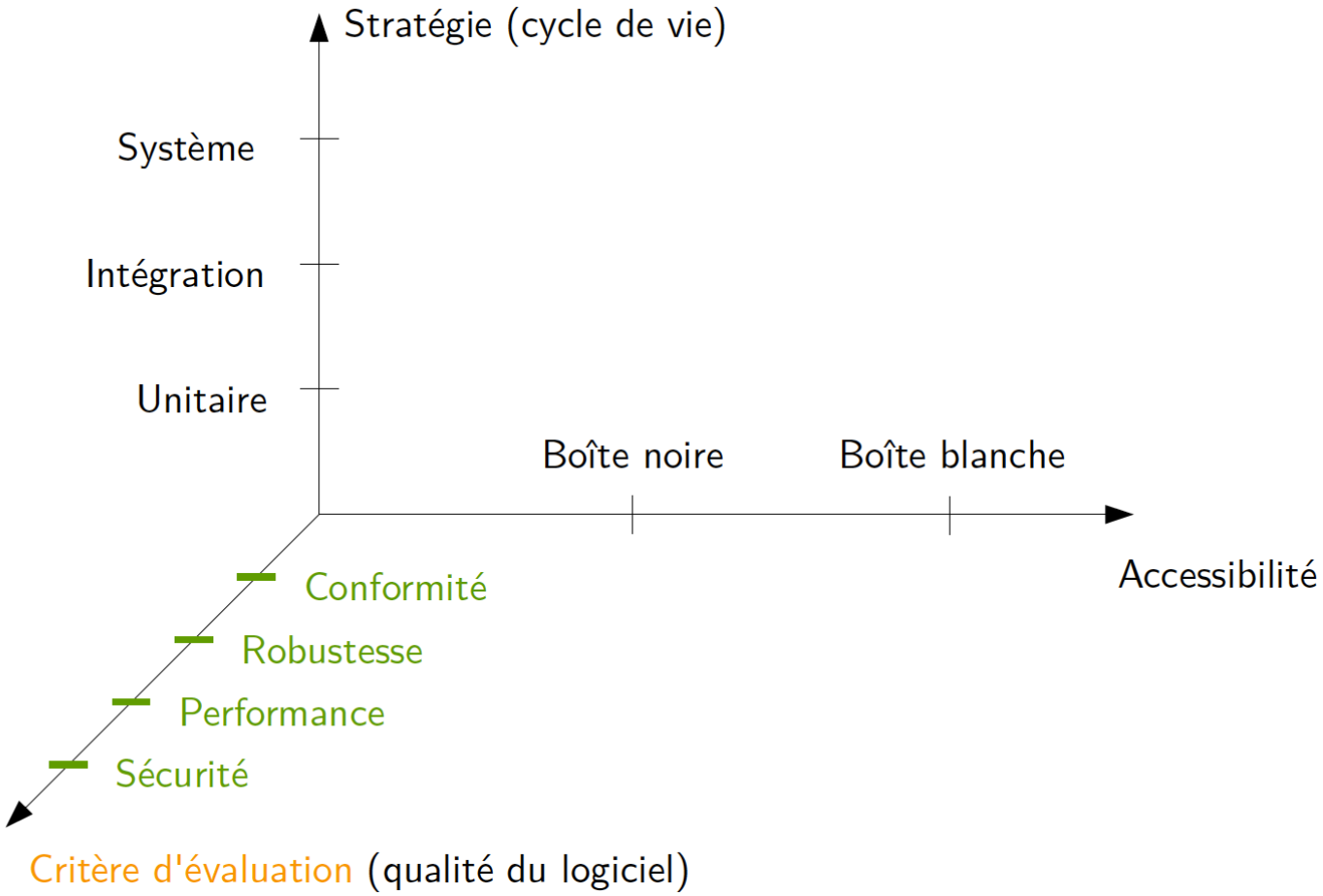
Test système

Test de la conformité du produit fini par rapport au cahier des charges, effectué en boîte noire au travers de son interface

Exemple : Commerce électronique

- Utilisation du système sur des scénarios réalistes et complets

Types de Test



N. Lazaar (IUT de Montpellier)
lazaar@lirmm.fr

Test de conformité

But : Assurer que le système présente les fonctionnalités attendues par l'utilisateur

Méthode : Sélection des tests à partir de la spécification, de façon à contrôler que toutes les fonctionnalités spécifiées sont implantées selon leurs spécifications

Exemple : Commerce électronique

- Scénarios avec transaction acceptée/refusée, couverture des différents cas et cas d'erreur prévus

Test de robustesse

But : Assurer que le système supporte les utilisations imprévues

Méthode : Sélection des tests en dehors des comportements spécifiés (entrées hors domaine, utilisation incorrecte de l'interface, environnement dégradé...)

Exemple : Service de paiement en ligne

- Login dépassant la taille du buffer
- Coupure réseau pendant la transaction

Test de sécurité

But : Assurer que le système ne possède pas de vulnérabilités permettant une attaque de l'extérieur

Méthode : Simulation d'attaques pour découvrir les faiblesses du système qui permettraient de porter atteinte à son intégrité

Exemple : Commerce électronique

- Essayer d'utiliser les données d'un autre utilisateur
- Faire passer la transaction pour terminée sans avoir payé

Test de sécurité

But : Assurer que le système ne possède pas de vulnérabilités permettant une attaque de l'extérieur

Méthode : Simulation d'attaques pour découvrir les faiblesses du système qui permettraient de porter atteinte à son intégrité

Exemple : Commerce électronique

- Essayer d'utiliser les données d'un autre utilisateur
- Faire passer la transaction pour terminée sans avoir payé

Test de performance

But : Assurer que le système garde des temps de réponse satisfaisants à différents niveaux de charge

Méthode : Simulation à différents niveaux de charge d'utilisateurs pour mesurer les temps de réponse du système, l'utilisation des ressources...

Exemple : Commerce électronique

- Lancer plusieurs centaines puis milliers de transactions en même temps

Test de performance

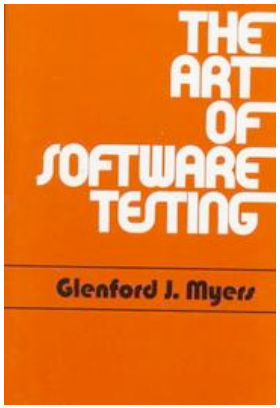
But : Assurer que le système garde des temps de réponse satisfaisants à différents niveaux de charge

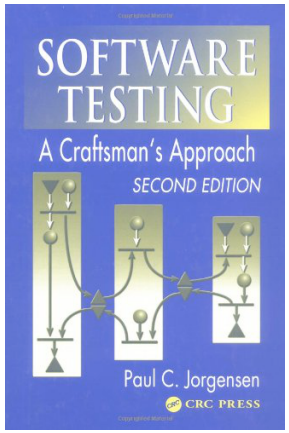
Méthode : Simulation à différents niveaux de charge d'utilisateurs pour mesurer les temps de réponse du système, l'utilisation des ressources...

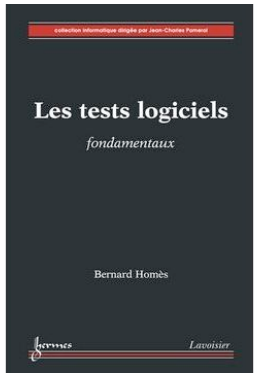
Exemple : Commerce électronique

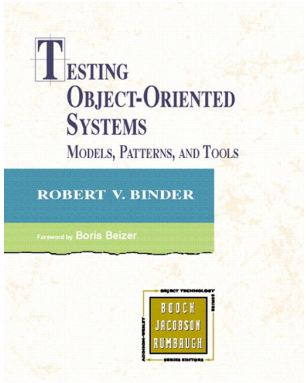
- Lancer plusieurs centaines puis milliers de transactions en même temps

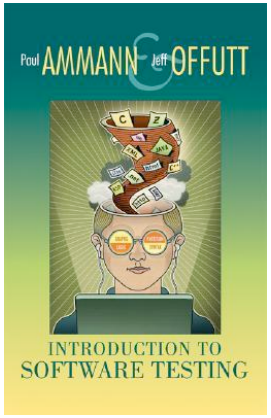
Bibliographie : quelques livres











N. Lazaar (IUT de Montpellier)
lazaar@lirmm.fr

Et quelques sites intéressants...

- Software Testing Online Resources

  `www.mtsu.edu/~storm/`

- Software Testing Stuff

  `www.testdriven.com`

- Model-based Software Testing

  `www.geocities.com/model_based_testing/`

- Opensource ST

  `www.opensourcetesting.org/unit_java.php`