# Introduction :
# Pourquoi de la VVT ?

VVT : Validation, Vérification & Test des logiciels

# Des bogues, des conséquences désastreuses…

- Banque de New York [21 novembre 1985] : pertes financières énormes
- Le Therac-25 [juillet 1985 ->avril 1986] : 3 morts
- Le crash d'AT&T [15 janvier 1990] : pertes financières énormes + la réputation d'AT&T entachée.
- Le Pentium [juin 1994] : pertes financières énormes + psychose
- Ariane 5-01 [4 juin 1996]

# Ariane 5-01 (4 juin 1996)

Le 23 juillet, la commission d'enquête remet son rapport : La fusée a eu un comportement nominal jusqu'à la 36ème seconde de vol. Puis les systèmes de référence inertielle (SRI) ont été simultanément déclarés défaillants. Le SRI n'a pas transmis de données correctes parce qu'il était victime d'une erreur d'opérande trop élevée du "biais horizontal" . . .

Les raisons :

1 Un bout de code d'Ariane IV (concernant le positionnement et la vitesse de la fusée) repris dans Ariane V

2 il contenait une conversion d'un flottant sur 64 bits en un entier signé sur 16 bits

3 pour Ariane V, la valeur du flottant dépassait la valeur maximale pouvant être convertie

4 ) défaillance dans le système de positionnement

5 ) la fusée a "corrigé" sa trajectoire

6 ) suite à une trop grande déviation, Ariane V s'est détruite !

# Le coût d'un Bogue ?

- Coût du bogue de l'an 2000 ?
  - Quelques chiffres avancés : 300, 1600 ou même 5 000 milliards de dollars
- Quel impact ?
  - Sécurité des personnes,
  - Retour des produits,
  - Relations contractuelles,
  - Notoriété, image,
  - …

⇒ Nécessité de « vérifier » certains logiciels/systèmes

# Nécessité de VVT

Comment effectuer de telles vérifications ?

- Méthodes formelles
  1. Test
     - nécessaire : permet de découvrir des erreurs
     - pas suffisant : non exhaustif (prouve la présence d'erreurs, pas leur absence !)
  2. Démonstration automatique
     - exhaustif
     - mise en œuvre difficile
  3. Model-checking
     - exhaustif, partiellement automatique…
     - mise en œuvre moins difficile (modèle formel+formalisation des propriétés)

⇒ VALIDATION VÉRIFICATION & TESTS

1, 2 et 3 sont des méthodes complémentaires :

- Test : non exhaustif mais facile à mettre en œuvre (bon rapport qualité/temps)
- Démonstration automatique : exhaustive mais considérée comme trop coûteux
- Model-checking : un compromis (?)

# Sans méthodes formelles :

- Coût des tests : 50 à 60% du coût total, voire 70% !
- Interprétation(s) des termes usuels (-> utilisation d'UML)
- Ambiguïté des méthodes semi-formelles (# sémantiques UML).
- Maîtrise difficile de certains types de programmations
  [événementielle / parallèle / …]
- Maintenance évolutive difficile

# Tendances actuelles ~ Méthodes formelles et certification

- Méthodes formelles :
  - Test, Démonstration automatique, Model-checking
- Politique de certification
- Certains niveaux de certification exigent des méthodes formelles
- Obligation de certification
  - Grandes entreprises
  - Application à risques
  - Sous-traitance

# Test & Validation dans les méthodes formelles

- Objectif ~ Pouvoir raisonner sur les logiciels et les systèmes afin de :
  - **Connaître** leurs comportements
  - **Contrôler** leurs comportements
  - **Tester** leurs comportements.
- Moyen ~ Les systèmes sont des objets mathématiques.
- Processus :
  1. **Obtenir un modèle formel** du logiciel ou du système. [Si la taille le permet, le modèle peut être le logiciel ou le système]
  2. **Analyser** le modèle formel par une technique formelle.
  3. **Générer des test** par une technique formelle
  4. **Transposer** les résultats obtenus sur les modèles aux logiciels et systèmes réels.
- Problèmes de l'approche :
  - Le modèle est-il fidèle ? **Validition**.
  - Peut-on tout vérifier ? **Décidabilité**.
  - Peut-on tout tester ? **Testabilité**.
  - La transposition des résultats est-elle toujours possible ? **Abstraction**.
  - Le test est-il **correct** ? Le test est-il **exhaustif** ?

# 1: Introduction au test de logiciels

# Génie Logiciel

Toute fabrication de produit suit les étapes suivantes :

1-Conception 2-Réalisation 3-Test

**Test :** On s'assure que le produit final correspond à ce qui a été demandé selon divers critères

Exemples de critères : esthétique, performance, ergonomie, fonctionnalité, robustesse

La fabrication de logiciel : toute une panoplie de langages de programmation, méthodes de programmation, concepts, outils, méthodes, technologies, normes, etc. [+Constante évolution !]

Exemples : C, ADA, C++, Java, C#, POO, programmation événementielle Corba, .NET, architecture 3-tier/n-tier, XML, webservice, Ajax, etc.

**Génie logiciel** : domaine dont l'objectif essentiel est la maîtrise (conceptualiser, rentabiliser, etc.) de l'activité de fabrication des logiciels.

# Assurance qualité

L'**assurance qualité** permet de mettre en œuvre un ensemble de dispositions qui vont être prises tout au long des différentes phases de fabrication d'un logiciel pour accroître les chances d'obtenir un logiciel qui corresponde à ses objectifs (son cahier des charges).

La définition et la mise en place des **activités de test** ne sont qu'un sous-ensemble des activités de l'assurance qualité, et le test aura pour but de minimiser les chances d'apparition d'une anomalie lors de l'utilisation du logiciel.

L'objet de ce qui suit consiste à étudier comment cet objectif peut être atteint.

# Erreur, défaut et anomalie

Une **anomalie** (ou **défaillance**) est un comportement observé différent du comportement attendu ou spécifié.

Exemple. Le 4 juin 1996, on a constaté…

Chaîne de causalité : **erreur** => **défaut** => **anomalie**

(nature de l'erreur :spécification, conception, programmation…)

Le terme **bogue** est malheureusement utilisé pour désigner aussi bien défaut qu'une anomalie.

**défaut** ≠ **anomalie**

Exemple : Une anomalie (telle une maladie) trouve toujours son explication dans un défaut (agent pathogène) et un défaut (un microbe latent) ne provoquera pas nécessairement une anomalie.

Comme le test est en aval de l'activité de programmation, les erreurs (humaines) déjà commises, ainsi que la façon de les éviter ne nous préoccupent pas ! Nous porterons notre attention sur les défauts qui ont été malencontreusement introduits afin de minimiser les anomalies qui risquent de se produire.

Sans nuire à la suite de ce cours, nous pouvons confondre, par abus de langage, erreur et défaut (tendance humaine à confondre cause et conséquence !!!)

# Classes de défaut

L'ensemble des défauts pouvant affecter un logiciel est infini.

Mais, des **classes de défaut** peuvent être identifiées : calcul, logique, E/S, traitement des données, interface, définition des données…

Les moyens pour détecter des défauts peuvent être automatiques ou manuels et s'appliquent aussi bien sur le code source qu'à son comportement.

**Donnons maintenant une définition de l'activité test**

**dans un projet logiciel.**

# Le test : des définitions…

**Définition** (issue de 'Le test des logiciels [SX-PR-CK-2000]) : Le test d'un logiciel est une activité qui fait partie du processus de développement. Il est mené selon les règles de l'assurance de la qualité et débute une fois que l'activité de programmation est terminée. Il s'intéresse aussi bien au code source qu'au comportement du logiciel. Son objectif consiste à minimiser les chances d'apparitions d'une anomalie avec des moyens automatiques ou manuels qui visent à détecter aussi bien les diverses anomalies possibles que les éventuels défauts qui les provoqueraient.

**Définition** (issue de la norme IEEE-STD729, 1983) : Le test est un processus manuel ou automatique, qui vise à établir qu'un système vérifie les propriétés exigées par sa spécification, ou à détecter des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification.

# Le test : des définitions…(suite et fin)

**Définition** (issue de l'A.F.C.I.Q) : "Le test est une technique de contrôle consistant à s'assurer, au moyen de son exécution, que le comportement d'un programme est conforme à des données préétablies".

AFCIQ : Association Française pour le Contrôle Industriel et la Qualité

**Définition** (issue de 'The art of software Testing' [GJM]) : « Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts".

# Qq commentaires sur les définitions du test

Le test d'un logiciel :

- a pour objectif de réduire les risques d'apparition d'anomalies avec des moyens manuels et informatiques.
- fait partie du processus de développement.
- n'a pas pour objectif de :
  - de corriger le défaut détecté (débogage ou déverminage)
  - de prouver la bonne exécution d'un programme.

**Procédure de test** : On applique sur tout ou une partie du système informatique un échantillon de données d'entrées et d'environnement, et on vérifie si le résultat obtenu est conforme à celui attendu. S'il ne l'est pas, cela veut dire que le système informatique testé présente une anomalie de fonctionnement. (Le test du logiciel est également appelé vérification dynamique.)

# Difficultés du test

1. Processus d'introduction des défauts très complexe
2. Mal perçu par les informaticiens et délaissé par les théoriciens

# Difficultés du test : Testabilité

**Testabilité** : Facilité avec laquelle les tests peuvent être développés à partir des documents de conception

**Facteurs de bonne testabilité** :

- Précision, complétude, traçabilité des documents
- Architecture simple et modulaire
- Politique de traitements des erreurs clairement définie

**Facteurs de mauvaise testabilité :**

- Fortes contraintes d'efficacité (espace mémoire, temps)
- Architecture mal définie

# Difficultés du test : Limites théoriques

**1-Indécidabilité** : une propriété indécidable est une propriété qu'on ne pourra jamais prouver dans le cas général (pas de procédé systématique)

Exemples de propriétés indécidables :

- L'exécution d'un programme termine
- Deux programmes calculent la même chose
- Un programme n'a pas d'erreurs

**2-Explosion combinatoire** : un programme a un nombre infini (ou extrêmement grand !) d'exécutions possibles

Le test n'examine qu'un nombre fini (ou très petit) d'exécutions

**Heuristiques** : approcher l'infini (ou l'extrêmement grand) avec le fini (très petit).

=> Choisir les exécutions à tester !

# Difficultés du test : conclusion.

**Conclusion :** Impossibilité d'une automatisation complète satisfaisante !

# Évolution du test

Aujourd'hui, le test de logiciel :

- est la technique de validation la plus utilisée pour s'assurer de la correction du logiciel.
- fait l'objet d'une pratique trop souvent artisanale.

Demain, le test de logiciel devrait être :

- une activité rigoureuse,
- fondée sur des modèles et des théories
- De plus en plus « automatique »

# Approches du test

L'activité de test se décline selon 2 approches :

- rechercher statiquement des défaut simples et fréquents (contrôle)
- définir les entrées (appelées 'données de test') qui seront fournies au logiciel pendant une exécution

Exemple de données de test (DT)

- DT1={a=2, z=4.3}

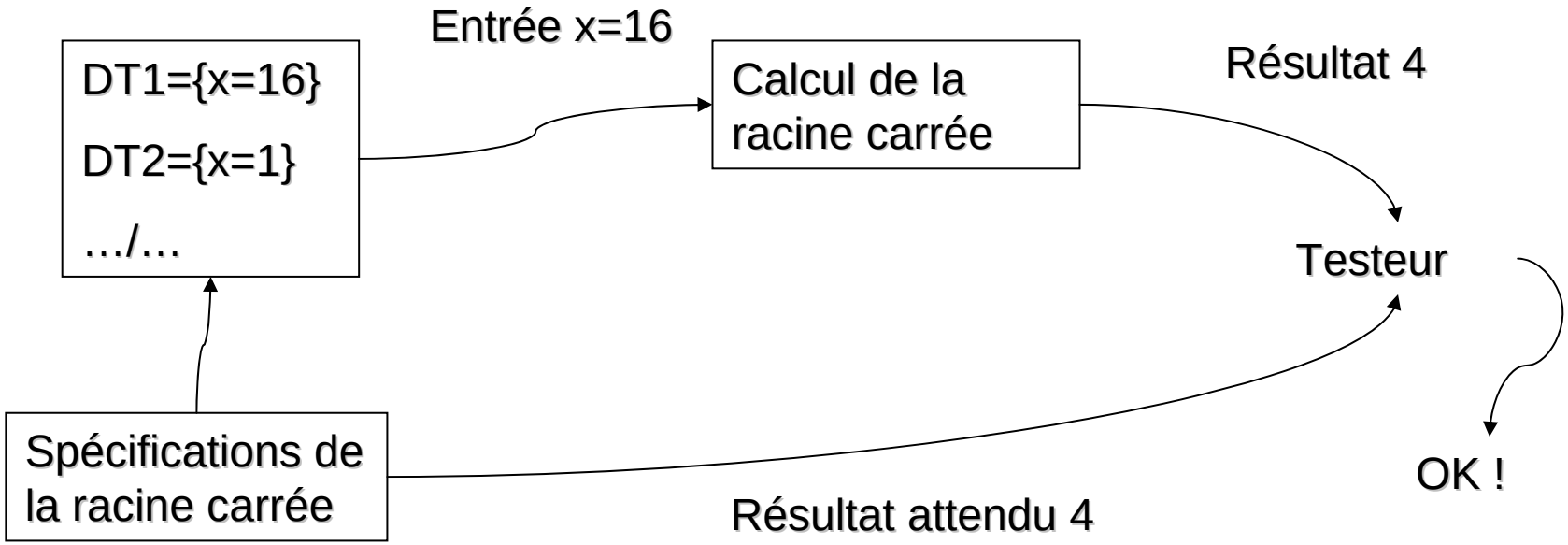
**Jeu de test** : est un ensemble de données de test.

**Scénario de test** : actions à effectuer avant de soumettre le jeu de test

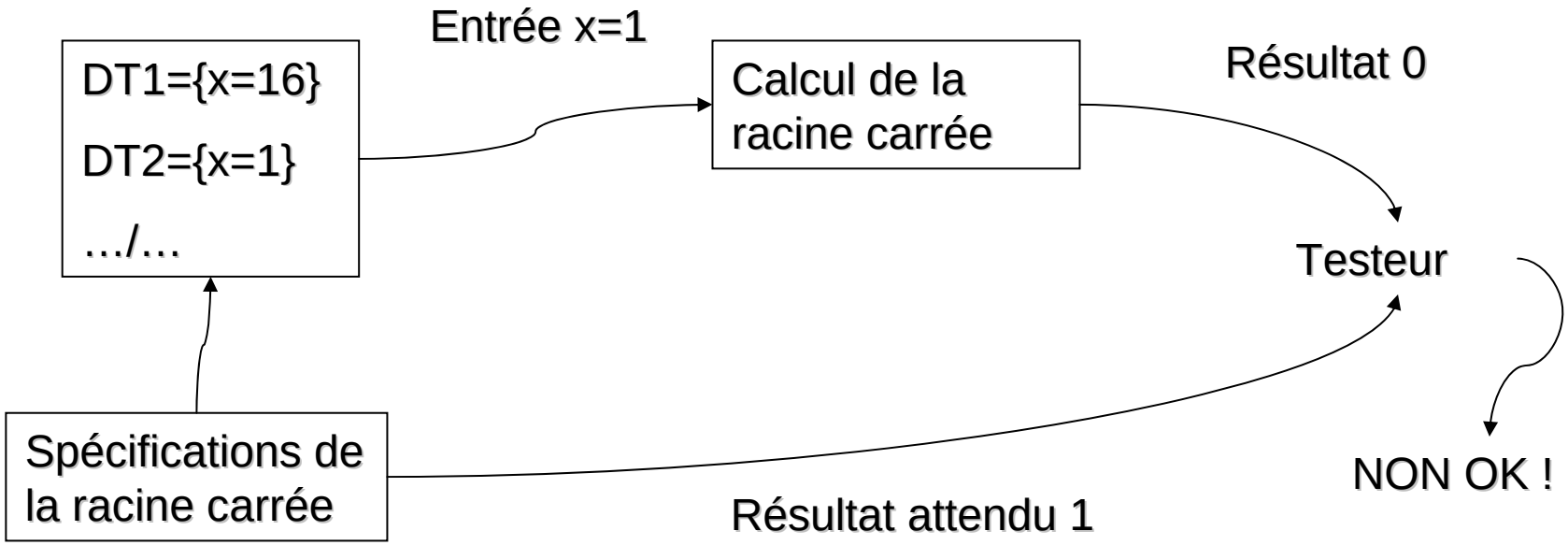
Le scénario de test produit un **résultat**

Ce résultat doit être évalué de manière manuelle ou automatique pour produire un **oracle**

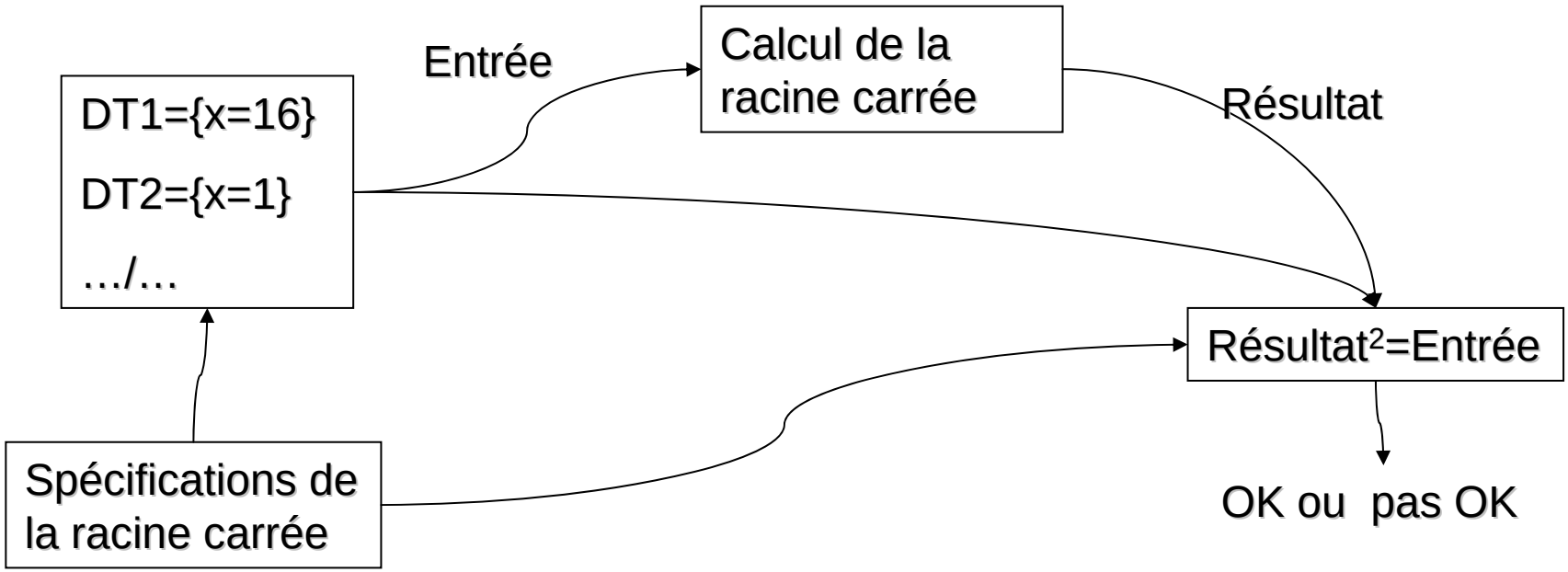
# Exemple 1 de test avec oracle manuel



# Exemple 2 de test avec oracle manuel



# Exemple 3 de test avec oracle automatique



# Choix des jeux de test

Les données de test sont **toutes** les entrées possibles :

- test **exhaustif**
- Idéal, mais non concevable !!!

Les données de test constituent un **échantillon représentatif** de toutes les entrées possibles :

- Exemple ‘Racine carrée’

  16, 1, 0, 2, 100, 65234, 826001234, -1, - 3

⇒ **Critère de test** (ou de sélection) : Un critère permet de spécifier formellement un objectif (informel) de test. Un critère de test peut, par exemple, indiquer le parcours de toutes les branches d'un programme, ou l'examen de certains sous-domaines d'une opération.

- Validité
- Fiabilité

# Validité, fiabilité, complétude d'un critère de test

**Validité** : Un critère de test est dit valide si pour tout programme incorrect, il existe un jeu de test non réussi satisfaisant le critère.

- P:programme, F:spécification,

  $T \subset D$ est fiable $\Leftrightarrow$

  [pour tout $t \in T$ $F(t)=P(t) \Rightarrow$ pour tout $t \in D$ $F(t)=P(t)$]

**Fiabilité** : Un critère est dit fiable s'il produit uniquement des jeux de test réussis ou des jeux de test non réussis.

**Complétude** : Un critère est dit complet pour un programme s'il produit uniquement des jeux de test qui suffisent à déterminer la correction du programme (pour lequel tout programme passant le jeu de test avec succès est correct)

Remarque : Tout critère valide et fiable est complet.

# La complétude : un rêve…

**Hypothèse de test** : La complétude étant hors d'atteinte en général, on peut qualifier un jeu de test par des hypothèses de test qui caractérisent les propriétés qu'un programme doit satisfaire pour que la réussite du test entraîne sa correction

# Classification des tests

Différentes classes de tests selon :

- les critères de test utilisées
- Les entités utilisées (spécification, code source, exécutable…)

Exemples de classes :

1. Les modalités de test : **Statique** / **Dynamique**
2. Les méthodes de test : **Structurelle** / **Fonctionnelle**
3. **Manuel** / **Automatique**
4. Les niveaux de tests : **Unitaire** / **Intégration** / **Système** / **Non-régression**
5. Les caractéristiques de test : **Robustesse** / **Conformité** / **Performance** /…

# Les modalités de test

1. Test **statique** : Test «par l'humain», sans machine, par lecture du code
   - inspection ou revue de code;
   - réunions (le programmeur, le concepteur, un programmeur expérimenté, un testeur expérimenté, un modérateur)
   - le but : trouver des erreurs dans une ambiance de coopération

2. Test **dynamique** : Test par l'exécution du système
   - **Implantation** du système (IUT = Implementation Under Test)
   - Une propriété / caractéristique à tester
   - un test réussit (Passes) si les résultats obtenus sont les résultats attendus, sinon il échoue (Fails);

# Les niveaux de tests

Tests **unitaires** (ou test de composant): s'assurer que les composants logiciels pris individuellement sont conformes à leurs spécifications et prêts à être regroupés.

Tests **d'intégration** :s'assurer que les interfaces des composants sont cohérentes entre elles et que le résultat de leur intégration permet de réaliser les fonctionnalités prévues.

Tests **système** : s'assurer que le système complet, matériel et logiciel, correspond bien à la définition des besoins tels qu'ils avaient été exprimés. [validation]

Tests de **non-régression** : vérifier que la correction des erreurs n'a pas affecté les parties déjà testées. [Cela consiste à systématiquement repasser les tests déjà exécutés]

# Les méthodes de test

1. Les méthodes **structurelles** : repose sur des analyses du code source
   - Examen de la structure du programme (flot de contrôle ou de données)
   - Aussi appelées test en boîte blanche, ou test basé sur l'implantation.
   - possibilité de fixer finement la valeur des entrées pour sensibiliser des chemins particuliers du code;
   - conception des tests uniquement pour le code déjà écrit.

2. Les méthodes **fonctionnelles** : repose sur une spécification (formelle ou informelle) du programme, le code source du programme n'est pas utilisé.
   - Aucune connaissance de l'implantation;
   - Aussi appelées test en boîte noire, ou test basé sur la spécification.
   - permet d'écrire les tests avant le codage;

Parfois : Combinaison des deux méthodes fonctionnelles et structurelles.

3. Les tests **orientés-erreurs** :
   les méthodes statistiques, le "semage" d'erreurs et les tests de mutation.

# Test manuel / test automatisé

1. Test **manuel**
   - le testeur entre les données de test par ex via une interface;
   - lance les tests;
   - observe les résultats et les compare avec les résultats attendus;
   - fastidieux, possibilité d'erreur humaine;
   - ingérable pour les grosses applications;

2. Test **automatisé**
   - Avec le support d'outils qui déchargent le testeur :
     - du lancement des tests;
     - de l'enregistrement des résultats;
     - parfois de la génération de l'oracle;
     - test unitaire pour Java: JUnit
     - génération automatique de cas de test : de plus en plus courant (cf Objecteering).

3. **Built-in** tests
   - Code ajouté à une application pour effectuer des vérifications à l'exécution:
     - À l'aide d'assertions !
     - ne dispense pas de tester ! test embarqué différent de code auto-testé !
     - permet un test unitaire "permanent", même en phase de test système;
     - test au plus tôt;
     - assertions: permettent de générer automatiquement l'oracle;

# Test de caractéristiques

Quelques exemples :

- test de **robustesse** : permet d'analyser le système dans le cas où ses ressources sont saturées ou bien d'analyser les réponses du système aux sollicitations proche ou hors des limites des domaines de définition des entrées. La première tâche à accomplir est de déterminer quelles ressources ou quelles données doivent être testées. Cela permet de définir les différents cas de tests à exercer. Souvent ces tests ne sont effectués que pour des logiciels critiques, c'est-à-dire ceux qui nécessitent une grande fiabilité.

- test de **performance** : permet d'évaluer la capacité du programme à fonctionner correctement vis-à-vis des critères de flux de données et de temps d'exécution. Ces tests doivent être précédés tout au long du cycle de développement du logiciel d'une analyse de performance, ce qui signifie que les problèmes de performances doivent être pris en compte dès les spécifications.

# Classification des tests

Classement des techniques de tests de logiciels selon :

- Critères adoptés pour choisir des DT représentatives
- Entités utilisées (spécification, code source, ou code exécutable)

- Techniques fonctionnelles / structurelles
- Techniques statiques / dynamiques
- Techniques combinant fonctionnelles, structurelles, dynamiques et statiques (c'est le cas du test boîte grise)

Un exemple de classement selon trois axes :

- le **niveau de détail** (étape dans le cycle de vie)
- le **niveau d'accessibilité**
- la **caractéristique**

# Classification des tests (suite)

**Niveau de détail**

- tests unitaires : vérification des fonctions une par une,
- tests d'intégration : vérification du bon enchaînement des fonctions et des programmes,
- tests de non-régression : vérification qu'il n'y a pas eu de dégradation des fonctions par rapport à la version précédente,

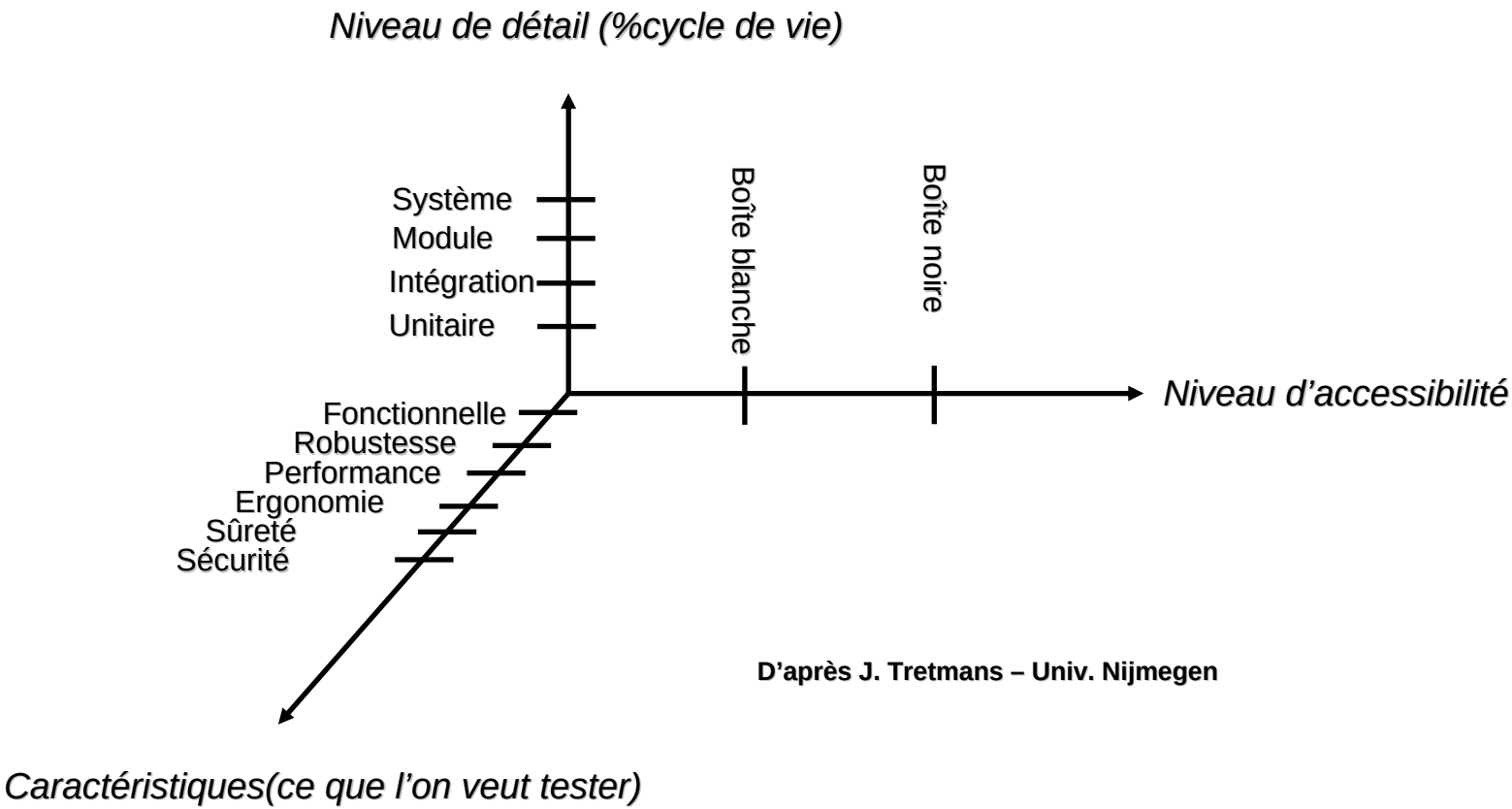
**Niveau d'accessibilité**

- Boîte noire : à partir d'entrée définie on vérifie que le résultat final convient.
- Boîte blanche : on a accès à l'état complet du système que l'on teste.
- Boîte grise : on a accès à certaines information de l'état du système que l'on teste.

**Caractéristique** :

- test fonctionnel
- test de robustesse
- test de performance

# Classification des tests (suite)



# Quelques exemples d'application

Test de programmes impératifs

- modèles disponibles : ceux issus de l'analyse de leur code source
- Donc : méthodes de test structurelles pour couvrir le modèle
- Couverture suivant des critères liés au contrôle ou aux données.

Test de conformité des systèmes réactifs

- Modèle disponible : la spécification
- Donc : méthodes de test fonctionnelles
- génération automatique de tests de conformité,

Test de systèmes

- Techniques de test d'intégration lors de la phase d'assemblage
- Aspects méthodologiques
- Test système.

# Stratégie de test

Une technique de test doit faire partie d'une stratégie de test

- adéquation avec le plan qualité
- Intégration dans le processus de développement des logiciels
- Une technique de test puissante restera sans effet si elle ne fait pas partie d'une stratégie de test…

La stratégie dépend :

- de la criticité du logiciel
- du coût de développement

Une stratégie définit :

- Des ressources mises en œuvre (équipes, testeurs, outils, etc.)
- Les mécanismes du processus de test (gestion de configuration, évaluation du processus de test, etc.)

Une stratégie tient compte :

- Des méthodes de spécif, conception
- Langages de programmation utilisés
- Du types d'application (temps réel, protocole, base de données…)
- L'expérience des programmeurs
- Etc.