Modélisation & Vérification

Introduction

Asma BERRIRI

asma.berriri@universite-paris-saclay.fr

Page web du cours :
https://sites.google.com/view/asmaberriri/home

Outline

Organisation du cours

- Modalités de contrôle des connaissances :
  - TDs/TPs notés
  - Examen écrit

  Note du contrôle continu (CC) = Moyenne des TDs/TPs notés
  Note finale = 40% CC + 60% Examen écrit
- Page web du cours :
  https://sites.google.com/view/asmaberriri/home

POLYTECH PARIS-SACLAY

Organisation du cours

Dans ce cours (36h):

- Cours Magistraux (+Exercices Intégrés)
  - Introduction / Motivation
  - Modéliser pour Vérifier/Tester (Finite State Machines)
  - Test Fonctionnel: Test basé sur les Modèles
  - Test Structurel
- Travaux Pratiques (à rendre)

  -> à commencer en classe et dus à échéance qui vous sera communiquée

  -> à envoyer à asma.berriri@universite-paris-saclay.fr

  - TD/TP Modélisation et Test Fonctionnel
  - TD Test Structurel
  - TP Test Structurel (PathCrawler)

POLYTECH PARIS-SACLAY

Outline

Motivation: Systèmes Informatiques

-> L'informatique devenant de plus en plus omniprésente, les risques deviennent plus importants
  - systèmes de transport (Voitures, Métros, TGV), contrôles aériens, aérospatiaux
  - procédés industriels critiques, centrales nucléaires, armement,
  - technologies médicales : téléchirurgie, contrôle radiologique
  - infrastructures et réseaux critiques de télécommunications
  - e-commerce, carte bancaire sans contact, passeport électronique
  - ...

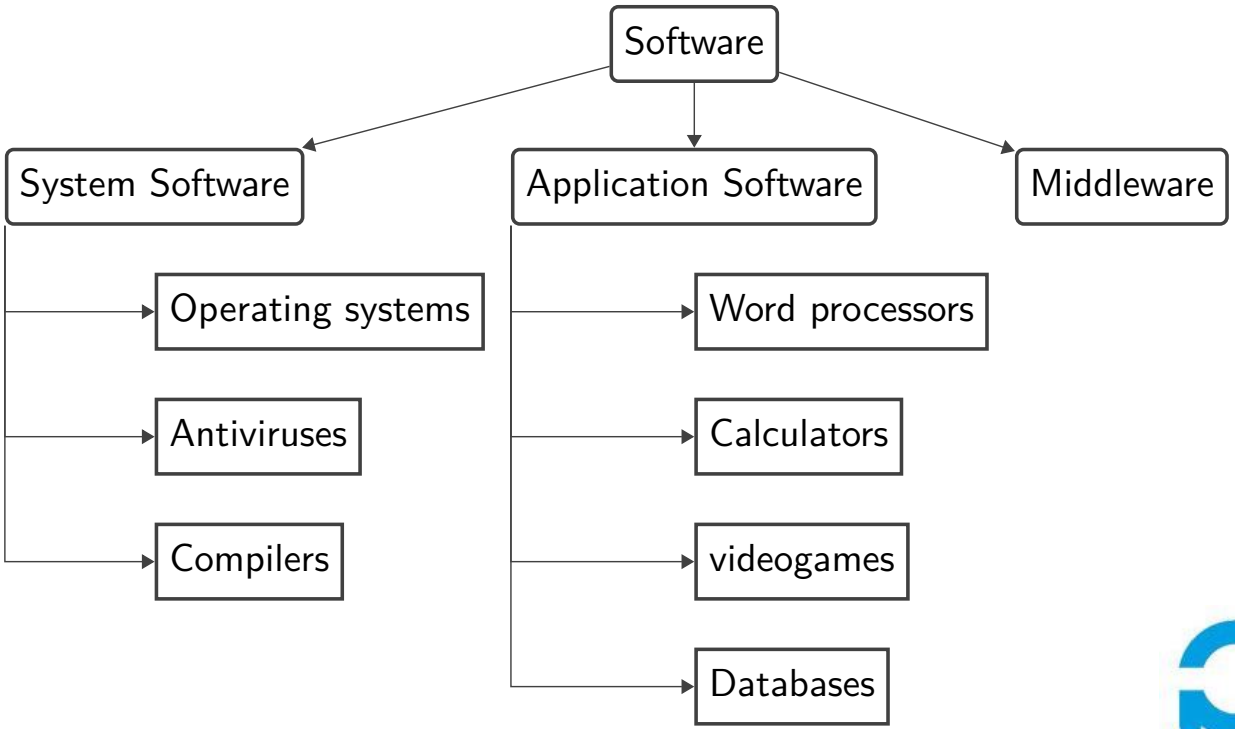
-> De plus en plus performants, de plus en plus miniaturisés
  - De plus en plus complexes

-> Systèmes critiques

POLYTECH PARIS-SACLAY

Motivation: Logiciels

- Ensemble d'entités nécessaires au fonctionnement d'un processus de traitement automatique de l'information
  - Programmes, données, documentation
- Ensemble de programmes qui permet à un système informatique d'assurer une tâche ou une fonction



POLYTECH PARIS-SACLAY

Motivation: Bugs célèbres

-> Therac 25, traitement aux rayons X des tumeurs cancéreuses
- En moins de 8s radiation $125\times$ la dose normal
- 6 décès aux états-Unis en 1986

-> AT&T
- Un patch non vérifié dans le système d'exploitation
- Une erreur dans une structure "switch" (en C)
- Le réseau téléphonique de la côte et des états-Unis a été bloqué pendant 9h!

-> Pentium
- Une erreur dans la division fottante du processeur
- 470 millions de $

-> Ariane-5 (1996)

POLYTECH PARIS-SACLAY

Bugs célèbres: Ariane-5

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 :

- Un bout de code d'Ariane-4 (concernant le positionnement et la vitesse de la fusée) repris dans Ariane-5
- il contenait une conversion d'un flottant sur 64 bits en un entier signé sur 16 bits
- pour Ariane-5, la valeur du flottant dépassait la valeur maximale pouvant être convertie
- défaillance dans le système de positionnement
- la fusée a "corrigé" sa trajectoire
- suite à une trop grande déviation, Ariane-5 s'est détruite !

Le coût d'un bug?

- Coût d'un bug ?
  - Quelques chiffres avancés : 300, 1600 ou même 5 000 milliards de dollars
  - 50% des coûts totaux ont été dépensés pour les tests et vérification dans les grands projets logiciels
    - 10 milliards$ /an en tests rien qu'aux US
    - plus de 50% du développement d'un logiciel critique (parfois $>$ 90%)
    - en moyenne 30% du développement d'un logiciel standard
- Quel impact ?
  - Sécurité des personnes,
  - Retour des produits,
  - Notoriété, image,
  - ...

POLYTECH PARIS-SACLAY

Motivation: Pourquoi est-il important d'avoir un logiciel correcte ?

-> Fiabilité, Sûreté et Sécurité deviennent indispensables

-> Construire des systèmes plus complexes, et les techniques de validation et de vérification sont un facteur limitant !

=> Nécessité de "vérifier" certains logiciels/système

=> Des méthodes pour valider et vérifier ? assurer la qualité des logiciels?

POLYTECH PARIS-SACLAY

Motivation: Qu'est-ce que la qualité ?

- ISO/IEC 9126 a d'abord défini la qualité logicielle
- En 2011, il a été remplacé par ISO/CEI 25010 : la sécurité et la compatibilité ont été ajoutées comme caractéristiques principales !

- Validité : réponse aux besoins des utilisateurs
- Facilité d'utilisation : prise en main et robustesse
- Performance : temps de réponse, débit, fluidité
- Fiabilité : tolérance aux pannes
- Sécurité : intégrité, protection des accès
- Maintenabilité : corriger ou transformer le logiciel
- Portabilité : changement d'environnement

Comment mesurer cette qualité ? Quand mesure-t-on la qualité ?

POLYTECH PARIS-SACLAY

Motivation: Measuring Software Quality

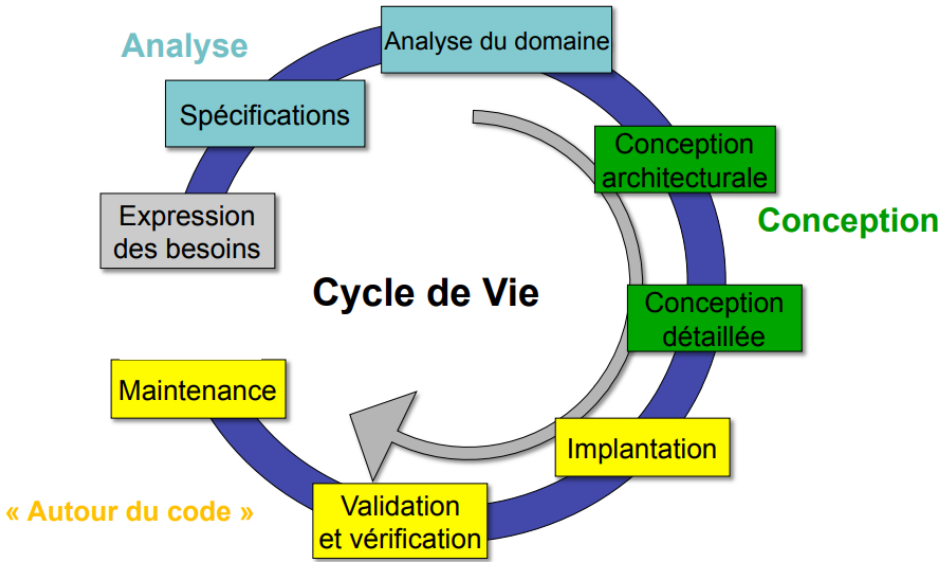
-> Les métriques sont utilisées pour mesurer la qualité logicielle

- e.g., les lignes de code (LOC) (+ lines -> + probabilité d'erreurs)

-> Quand mesurer la qualité d'un logiciel donné ?

- Une fois qu'il est « prêt » ?
- Peut-être aussi pendant l'exécution ?
- Puis-je / dois-je mesurer quelque chose pendant que je le conçois ?

System Development Life Cycle (SDLC) phases for metrics:



1. Specification (requirements)
2. Design
3. Development (implementation)
4. Testing
5. Production

POLYTECH PARIS-SACLAY

Motivation: Test, Vérification et Validation

-> 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 (theorem proving)
   - exhaustif
   - mise en œuvre difficile
3. Model-checking
   - exhaustif, partiellement automatique
   - mise en œuvre moins difficile (modèle ormel+formalisation des propriétés)

Motivation: Test, Vérification et Validation

-> Validation. The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers.

- Le système satisfait-il le besoin du client ?
- La performance est-elle suffisante ?
- Tient-il la charge ?
- A-t-on construit le bon système ?

-> Verification. The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process.

- A-t-on construit le système correctement ?
- Est-il correct (par rapport au cahier des charges, à la spécification) ?

(B. W. Boehm) Verification is building the product right, and Validation is building the right product.

POLYTECH PARIS-SACLAY

Motivation: Les Méthodes et Techniques

-> Méthodes de validation :

- Revue de spécification, de code
- Prototypage rapide
- Développement de tests

-> Méthodes de vérification :

- Test formel (à partir de la spécification) et méthodes formelles
  - ici méthodes de test basées sur les modèles formels
- Autres méthodes formelles de vérification
  - Preuve de programmes
  - Model-checking
  - ...

POLYTECH PARIS-SACLAY

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

POLYTECH PARIS-SACLAY

Méthodes formelles et certification

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

POLYTECH PARIS-SACLAY

Motivation: But du test: Trouver des erreurs

Méthode dynamique : elle requiert l'exécution du programme pour trouver des erreurs

> *Testing is the process of analyzing a software item to detect the differences between existing and required conditions, (that is, bugs) and to evaluate the features of the software items.*
>
> (IEEE–Standard Glossary of Software Engineering Terminology)

> *Testing is the process of executing a program with the intent of finding errors.*
>
> (Glenford J. Myers. The Art of Software testing. Wiley, 1979)

> *Testing can reveal the presence of errors but never their absence.*
>
> (Edsgar W.Dijkstra. Notes on structured programming, 1972)

POLYTECH PARIS-SACLAY

Motivation: Deux aspects dans le test

=> Construire la qualité du produit

- lors de la phase de conception / codage en partie par les développeurs (tests unitaires)
- but = trouver rapidement le plus de bugs possibles (avant la commercialisation)
- Ici un test réussi est un test qui permet de trouver un bug

=> Démontrer la qualité du produit à un tiers

- une fois le produit terminé, en général, par une équipe dédiée ($\neq$ de celle qui a développé le produit)
- but = convaincre (organismes de certification, hiérarchie, client)
- Ici un test réussi est un test qui passe sans problème - les tests doivent être représentatifs (systèmes critiques : audit du jeu de tests)

POLYTECH PARIS-SACLAY

Motivation: Limites des techniques de test ?

=> Hypothèses sur la testabilité

- système sous test est déterministe (ou contrôle son non-déterminisme), doit être initialisable

=> Hypothèses de test

- Comportement uniforme, régulier : hypothèse réaliste mais pas toujours

=> Limites sur le résultat obtenu

- On ne peut armer que le programme satisfait sa spécification. On ne peut le dire qu'avec un certain degré de confiance.
- Le test exhaustif (pour toutes les entrées possibles) est en général impossible.
- On ne peut exclure la possibilité d'une erreur (future) même avec un critère de couverture du code ou des specs le plus fort possible.

=> Les méthodes manuelles de test passent très mal à l'échelle en terme de taille de code et de niveau d'exigence ! fort besoin d'automatisation

POLYTECH PARIS-SACLAY

Motivation: But de la preuve formelle

Méthode statique

$=>$ Etant donnée une spécification formelle S d'un programme P, donner une preuve mathématique rigoureuse que toute exécution de P satisfait S

$$P \text{ est correct par rapport à } S$$

Que faut-il pour vérifier formellement P ?

- une spécification formelle de P (rigoureuse, mathématique),
- une méthode de preuve de correction (logique de Hoare par exemple),
- des règles de preuve pour faire des preuves rigoureuses, mathématiques : un calcul

POLYTECH PARIS-SACLAY

Motivation: Limitations de la preuve formelle

=> Pas de notion absolue de correction

- La correction est toujours relative à une spécification donnée

=> Difficile et coûteux d'écrire des spécifications formelles

- En pratique, on ne spécifie pas formellement toutes les fonctionnalités mais les propriétés de sûreté comme la bonne formation des données (accès hors des bornes, déréférencement du pointeur null, etc.), l'absence d'exceptions non détectées, les parties critiques du logiciel, etc.

=> Coûteux également de faire des preuves (mais on gagne sur le temps de test)

POLYTECH PARIS-SACLAY

Test & Validation dans les méthodes formelle

- 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 :
  - Obtenir un modèle formel du logiciel ou du système. [Si la taille le permet, le modèle peut être le logiciel ou le système]
  - Analyser le modèle formel par une technique formelle
  - Générer des test par une technique formelle
  - 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?

Le test et la vérification formelle sont complémentaires!

Test du logiciel: Présentation générale

Terminologie: Quelques définitions :

- Erreur: comportement erroné (erreur humaine)
- Défaut/faute: élément du logiciel qui a provoqué l'anomalie
- Défaillance/Anomalie: caractérisée par des résultats inattendus ou un service non rendu mais le logiciel peut continuer son fonctionnement
- Panne/Bug: arrêt total ou partiel du logiciel entraînant un fonctionnement en mode dégradé

Chaine de l'erreur : Erreur $\Longrightarrow$ Défaut $\Longrightarrow$ Défaillance $\Longrightarrow$ Panne

Causes des défauts :
Erreur humaines, mauvaise compréhension/interprétation de la spécification, erreur de réalisation, oublis lors de la phase de conception

Types de défauts :

- Calcul: $x = x + 2$ au lieu de $x = y + 2$
- Logique: $if\,(a > b)$ au lieu de $if\,(a < b)$
- Entrée/Sortie: mauvaise description, mauvaise conversion ou mauvais format
- Traitement des données : mauvais accès ou mauvaise manipulation des données
- Interface : mauvaise communication entre les constituants internes du logiciel
- Définition des données : mauvaise définitions des données

Tout défaut ne conduit pas systématiquement à une défaillance, le logiciel se comporte correctement malgré un grand nombre de défauts car défauts jamais exercés en fonctionnement

Test du logiciel: 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

POLYTECH PARIS-SACLAY

Test du logiciel: Processus simplifié

Processus valable pour tout type de test

1. On définit un cas de test (CT) à exécuter, i.e. un scénario que l'on veut appliquer
2. On concrétise le cas de test en lui fournissant des données de test (DT)
3. On indique le résultat que l'on attend pour ce CT (prédiction de l'oracle)
4. On exécute le script de test testant le CT sur les DT
5. On compare le résultat obtenu à la prédiction de l'oracle (verdict)
6. On rapporte le résultat du test : succès / échec

Suite de tests : ensemble de DTs.

POLYTECH PARIS-SACLAY

Test du logiciel: Les dicultés du test

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

Choisir les cas de test / données de test

- Estimer le résultat attendu - problème de l'oracle
- Exécuter le programme sur les DTs
- Calculer le verdict d'un test
  Attention : aux systèmes embarqués et cyber-physiques
- Quand arrêter de tester ? Calcul d'une couverture
  Attention au choix du critère de couverture
- Rejouer les tests à chaque changement (test de régression).
  Attention maintenance et optimisation

POLYTECH PARIS-SACLAY

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

Test du logiciel: Oracle ?

Donne le résultat attendu

=> Oracles parfaits automatisables :

- résultat simple à vérifier (ex : solution d'une équation), comparer à une référence (table existante de résultats, logiciel existant), disponibilité d'un logiciel similaire (test dos à dos), version précédente ou version courante, mais options différentes)

=> Oracles partiels mais automatisés :

- oracle basique : pass/fail, instrumentation du code (assert) mieux programmation avec contrats (voir la suite), property-based testing (ex. Quickcheck OCaml, Haskell)
  Parfois l'oracle est l'expert du domaine ...

POLYTECH PARIS-SACLAY

Test du logiciel: Sélection de tests?

Constat : test exhaustif souvent impraticable

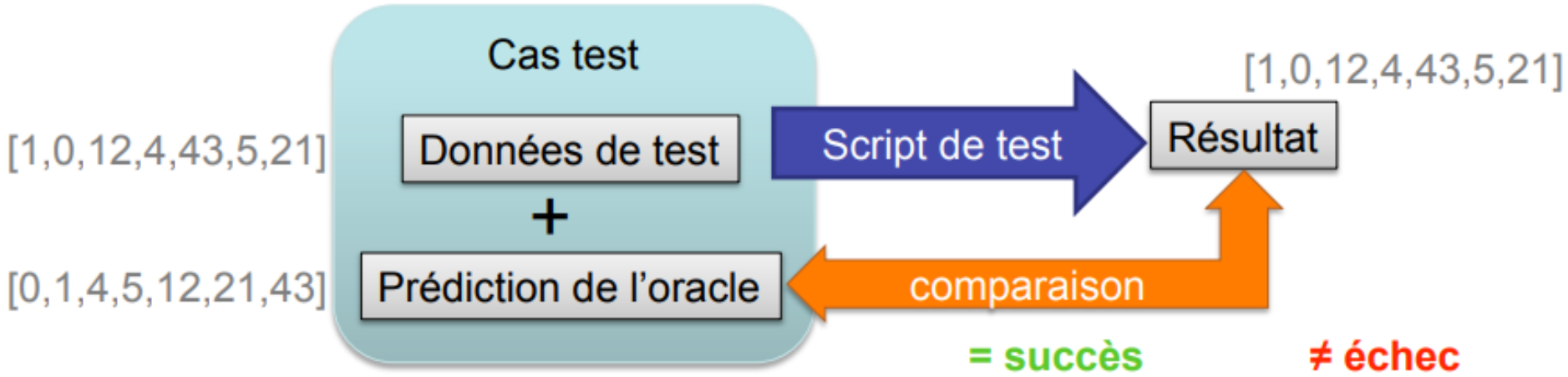
Comment choisir les scénarios de test à jouer ?

Deux (trois ?) grandes familles de sélection de tests

1. Test fonctionnel ou boîte noire (Black Box)
   test choisi à partir des spécifications (use-case, user story, etc.)
   évaluation de l'extérieur (sans regarder le code), uniquement en fonction des entrées et des sorties sur le logiciel ou un de ses composants
   - \+ taille des spécifications, facilité oracle
   - \- spécifications parfois peu précises, concrétisation des DT
2. Test structurel ou boîte blanche (White Box)
   test choisi à partir du code source (portion de codes, blocs, branches)
   - \+ description précise, facilité script de test
   - \- oracle diffcile, manque d'abstraction
3. Cas particulier du test probabiliste
   Principe de la boîte noire (juste domaine des variables), mais spécificités propres (ex : oracle délicat)

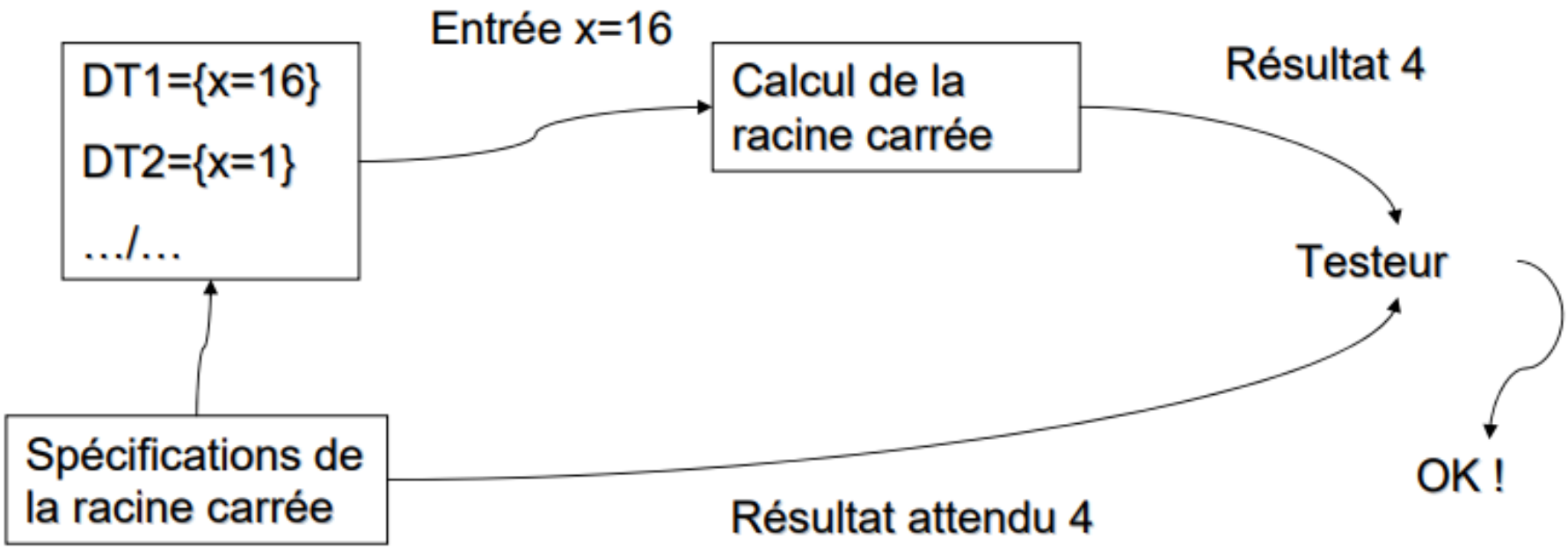
Test du logiciel: Exemple 1

Exemple : un programme de tri d'un tableau d'entiers en enlevant les redondances
CT : un tableau de N entiers non redondants

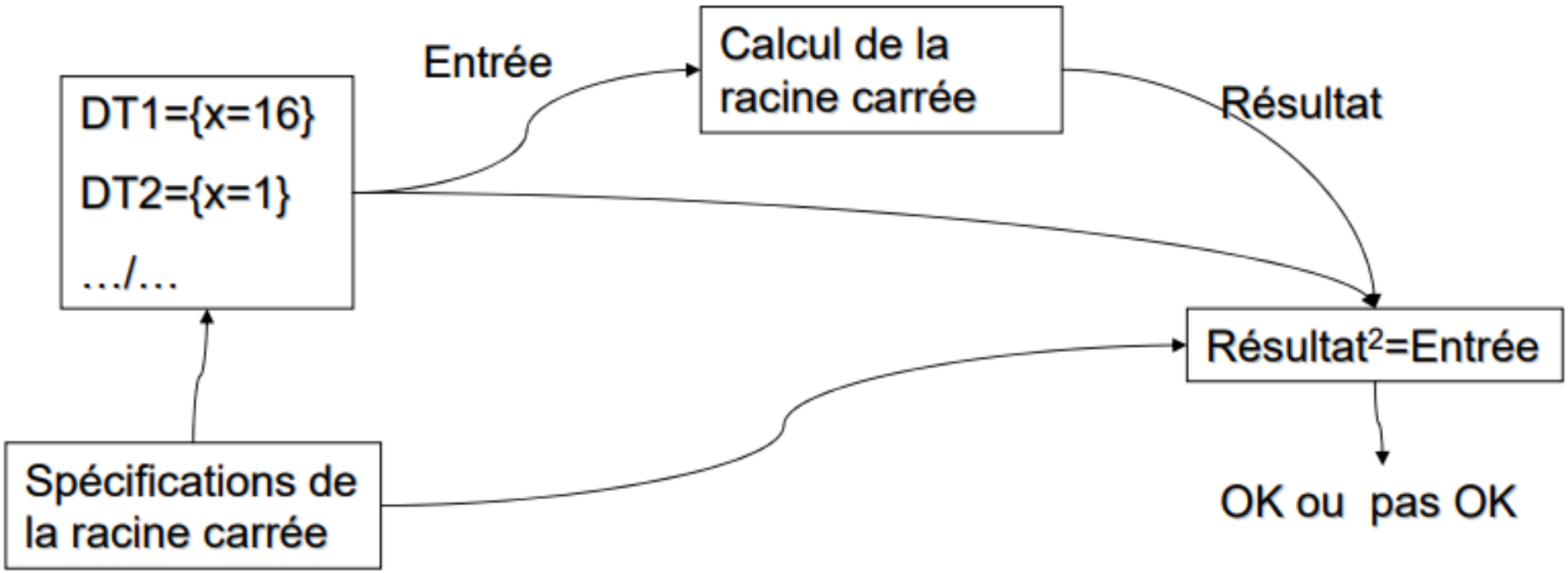


POLYTECH PARIS-SACLAY

Test du logiciel: avec oracle manuel Exemple 2



Test du logiciel: avec oracle automatique Exemple 3



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

POLYTECH PARIS-SACLAY

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 (TS) non réussi satisfaisant le critère

$$P : \textit{programme},\ S : \textit{specification}$$
$$TS \subset D \iff \forall\, t \in TS\ S(t) = P(t) \implies \forall\, t \in D\ S(t) = P(t)$$

- 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)
- Tout critère valide et fiable est complet

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

Différents types de tests

Différentes classes de tests selon :

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

Exemples de classes :

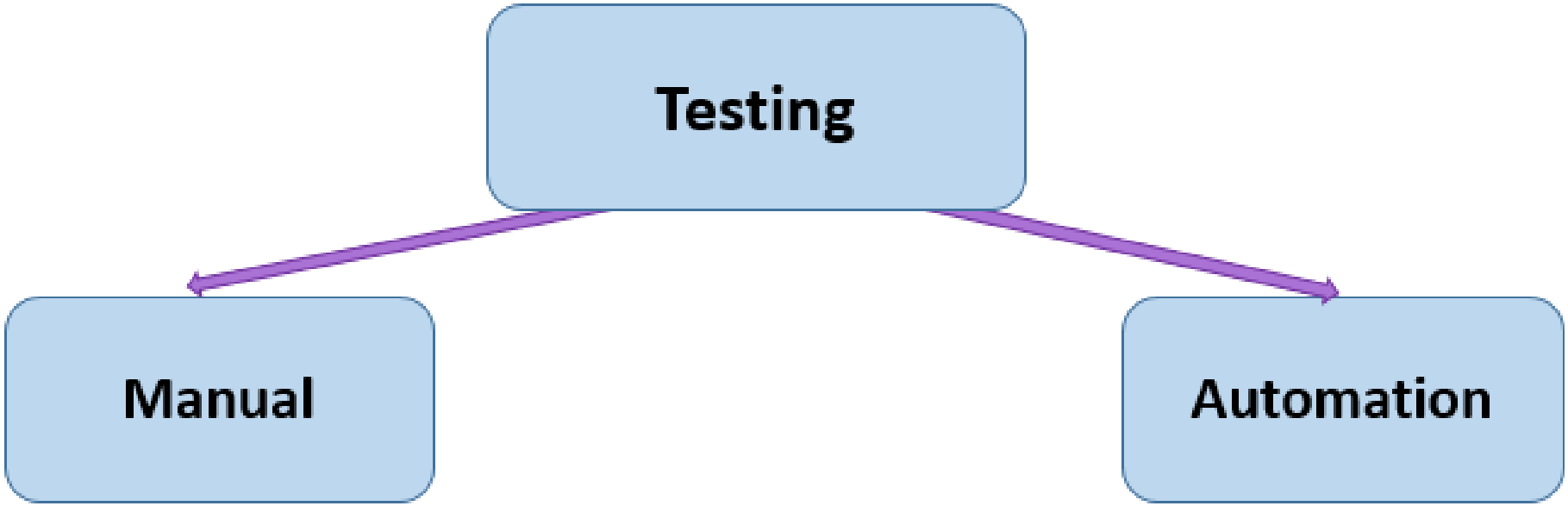
1. Les modalités de test : Statique / Dynamique
2. Les méthodes de test : Structurelle / Fonctionnelle
3. Manuel / Automatique
4. Les niveaux de tests : Unitaire / Intégration / Système / Non-régression
5. Les caractéristiques de test : Robustesse / Conformité / Performance / ...

POLYTECH PARIS-SACLAY

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)

Différents types de tests: Test manuel / test automatisé



A tester is a 'real' human being...

+ Il/elle peut «deviner» les bugs potentiels

- Peu de cas de test peuvent être générés

Semi-automation

Est souvent utilisé, dans la pratique

A tester is a tool

+ Permet de générer automatiquement de nombreux cas de test

- Quelqu'un a besoin d'écrire un tel script de «test»

Différents types de tests: 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

   $\Rightarrow$ fastidieux, possibilité d'erreur humaine

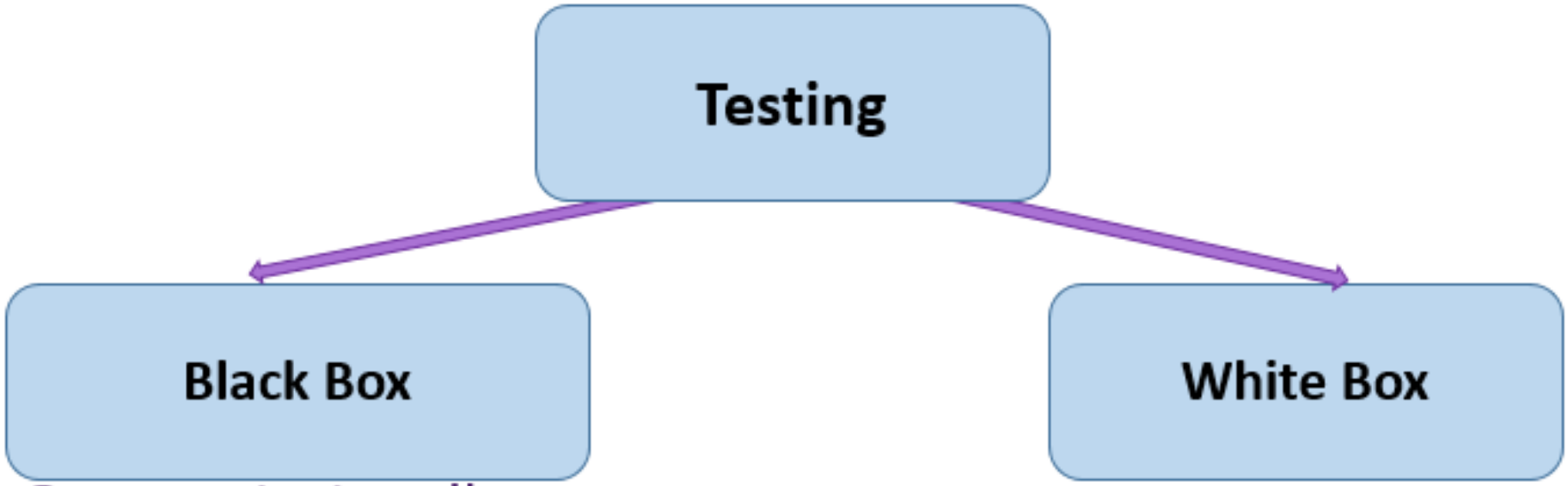
   $\Rightarrow$ ingérable pour les grosses applications
2. Test automatisé: Avec le support d'outils qui déchargent le testeur
   - du lancement des tests
   - de l'enregistrement des résultats
   - parfois de la génération de l'oracle

   $\Rightarrow$ test unitaire pour Java: JUnit

   $\Rightarrow$ 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

   $\Rightarrow$ test au plus tôt

Hypothèses de tests (Testing Assumptions)



On ne sait rien d'un *IUT*, si ce n'est ses interfaces

- Seul un fichier exécutable est fourni
- L'architecture du système ainsi que le code source restent inconnus

Grey Box

Un testeur en sait plus que les interfaces, par ex. éléments de conception

On sait tout sur un *IUT*

- Le code source est donné
- Un testeur connaît la logique et la structure de ce code

Méthodologies de tests (Testing methodologies)

Functional Testing
L'implémentation est-elle conforme à la spécification ?

Unit Testing
Les pièces individuelles fonctionnent-elles?

Regression Testing
Lorsque certaines "petites" pièces sont modifiées, comment dériver des tests ?

Integration/Interoperability Testing
La composition des pièces fonctionne-t-elle ?

Security Testing
Le logiciel n'a -t- il pas de vulnérabilités ?

Stress Testing
Comment le logiciel se comporte-t-il dans des conditions anormales (extrêmes) ?

Performance Testing
Le logiciel est-il évolutif ?

...

Méthodologies de tests (Testing methodologies)

- 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)

POLYTECH PARIS-SACLAY

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

POLYTECH PARIS-SACLAY

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

POLYTECH PARIS-SACLAY

Classification des tests

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

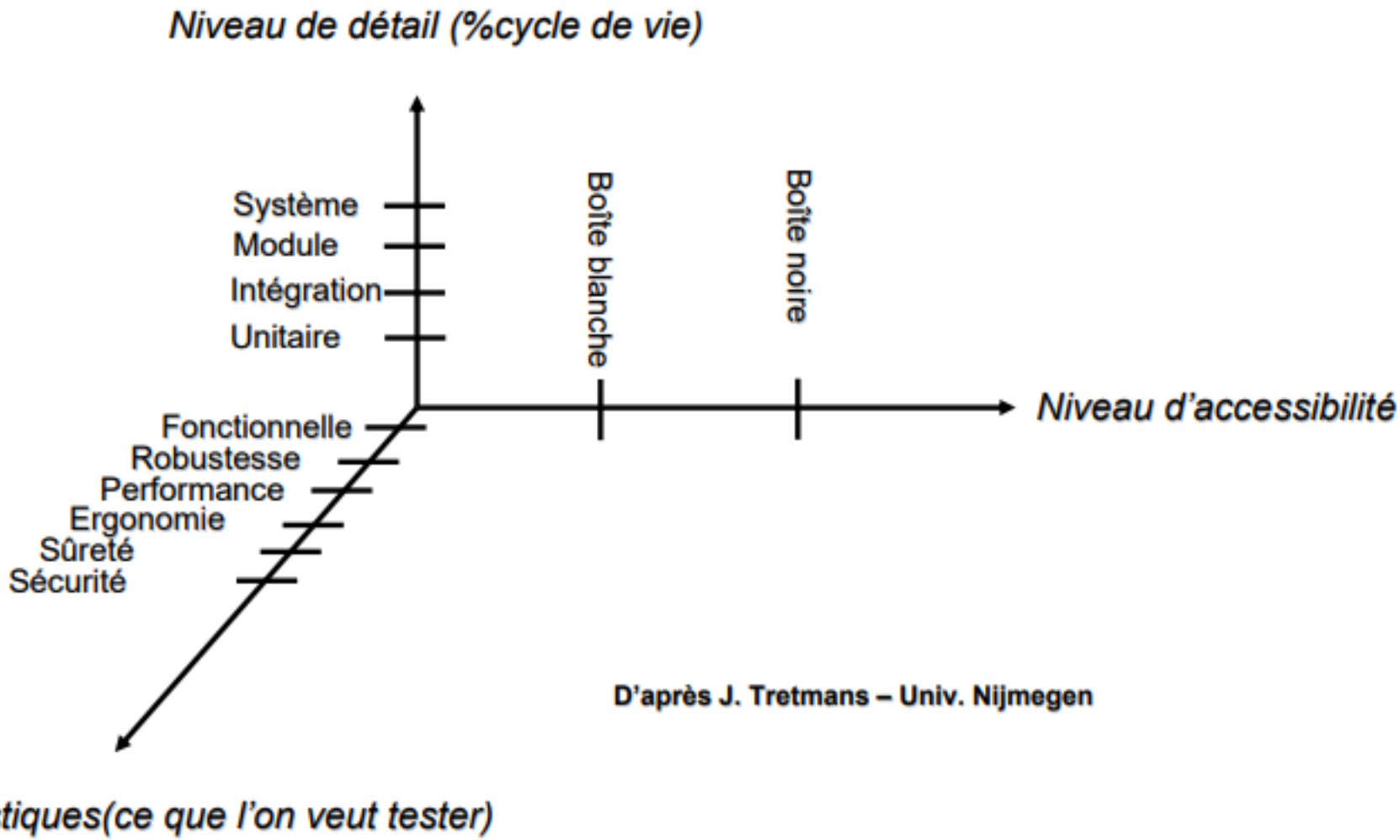
- Boîte noire : à partir d'entrée définie on vérifie que le résultat final convient
- Boîte blanche : on a accès à l'état complet du système que l'on teste
- Boîte grise : on a accès à certaines information de l'état du système que l'on teste

Caractéristique :

- test fonctionnel
- test de robustesse
- test de performance

POLYTECH PARIS-SACLAY

Classification des tests



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

POLYTECH PARIS-SACLAY

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
- ...

POLYTECH PARIS-SACLAY

Test du logiciel: Aller plus loin ...

(Ammann et Offutt) Introduction to software testing

(Mathur) Foundations of Software Testing

(Myers) Art of Software Testing

(Sommerville) Software Engineering

POLYTECH PARIS-SACLAY