Conduite de Projet
Cours 7 — Testing

Stefano Zacchiroli
zack@pps.univ-paris-diderot.fr

Laboratoire IRIF, Université Paris Diderot

2015–2016

Méthodes de V&V complémentaires

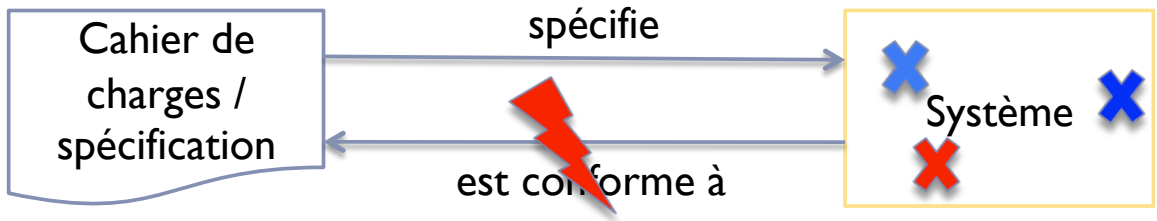
- Model-checking :
  - ✔ Exhaustif, automatique
  - ✗ Mise en œuvre moyennement difficile (modèles formelles, logique temporelle)
- Preuve :
  - ✔ Exhaustif
  - ✗ Mise en œuvre difficile, limitation de taille
- Test :
  - ✔ Nécessaire : exécution du système réel, découverte d'erreurs à tous les niveaux (spécification, conception, implantation)
  - ✗ Pas suffisant : exhaustivité impossible

Test de logiciel

Selon IEEE (Standard Glossary of Software Engineering Terminology)

« Le test est l'exécution ou l'évaluation d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus. »

- Validation dynamique (exécution du système
- Comparaison entre système et spécification



Qu'est ce qu'un « bug » ?

(vocabulaire IEEE)

- Anomalie (fonctionnement) : différence entre comportement attendu et comportement observé
- Défaut (interne) : élément ou absence d'élément dans le logiciel entraînant une anomalie
- Erreur (programmation, conception) : comportement du programmeur ou du concepteur conduisant à un défaut

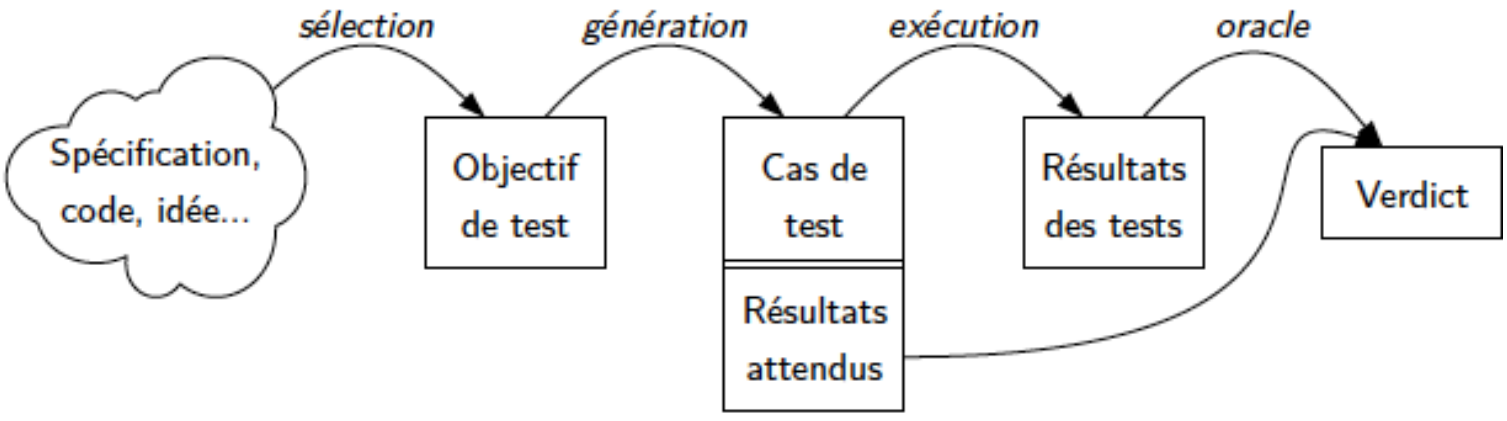
erreur → défaut → anomalie

Qu'est ce qu'un test ?

(vocabulaire IEEE)

- SUT (System Under Test) : le système testé.
- Objectif de test : comportement SUT à tester
- Données de test : données à fournir en entrée au système de manière à déclencher un objectif de test
- Résultats d'un test : conséquences ou sorties de l'exécution d'un test
  - (affichage à l'écran, modification des variables, envoi de messages...)
- Cas de test : données d'entrée **et** résultats attendus associés à un objectif de test

Qu’est ce qu’un test ?



Exemple

- Spécification : Le programme prend en entrée trois entiers, interprétés comme étant les longueurs des côtés d'un triangle. Le programme retourne la propriété du triangle correspondant : scalène, isocèle ou équilatéral.

- Exercice : Écrire un ensemble de tests pour ce programme

Exemple

Cas valides	Données	Résultat attendu
triangle scalène valide	(10,5,7)	scalène
triangle isocèle valide + permutations	(3,5,5)	isocèle
triangle équilatéral valide	(3,3,3)	équilatéral
triangle plat (a+b=c) + permutations	(2,2,4)	scalène

Cas invalides		
pas un triangle (a+b<c) + permutations	(2,1,5)	triangle invalide
une valeur à 0	(3,0,4)	triangle invalide
toutes les valeurs à 0	(0,0,0)	triangle invalide
une valeur négative	(2,-1,6)	triangle invalide/entrée invalide
une valeur non entière	('a',4,2)	entrée invalide
mauvais nombre d'arguments	(3,5)	entrée invalide

Un autre exemple : tri d'une liste

Objectif de test	Donnée d'entrée	Résultat attendu	Résultat du test
liste vide	[ ]	[ ]	[. . .]
liste à 1 élément	[3]	[3]	[. . .]
liste ≥ 2 éléments, déjà triée	[2;6;9;13]	[2;6;9;13]	[. . .]
liste ≥ 2 éléments, non triée	[7;10;3;8;5]	[3;5;7;8;10]	[. . .]

égalité ?

Problème de l'oracle

- Oracle : décision de la réussite de l'exécution d'un test, comparaison entre le résultat attendu et le résultat obtenu
- Problème : décision pouvant être complexe
  - types de données sans prédicat d'égalité
  - système non déterminisme : sortie possible mais pas celle attendue
  - heuristique : approximation du résultat optimal attendu
    - Exemple : problème du sac à dos
- Risques : Échec d'un programme conforme si définition trop stricte du résultat attendu
  - => faux positifs (false fails)

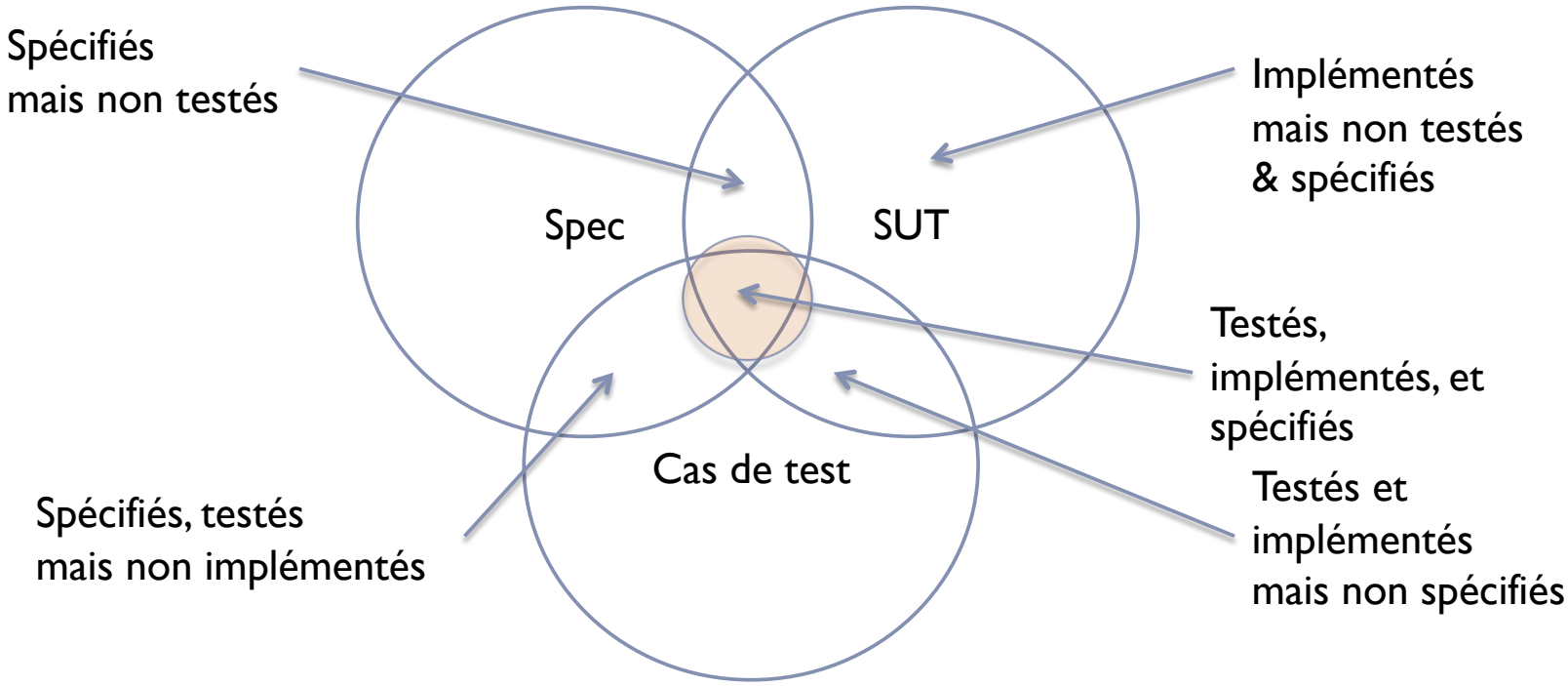
Faux positifs et faux négatifs

- Validité des tests : Les tests n'échouent que sur des programmes incorrects
- Faux positif (*false-fail) : fait échouer un programme correct*

- Complétude des tests : Les tests ne réussissent que sur des programmes corrects
- Faux négatif (*false-pass) : fait passer un programme incorrect*

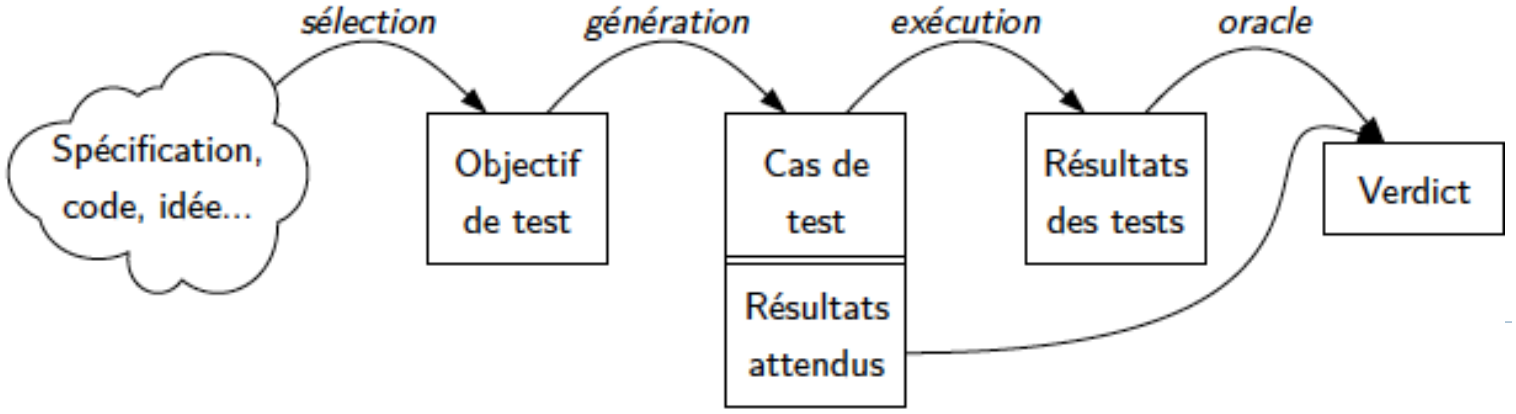
- Validité indispensable, complétude impossible en pratique
  - Toujours s'assurer que les tests sont valides

Validité et complétude des tests



Processus de test

1. Choisir les comportements à tester (objectifs de 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



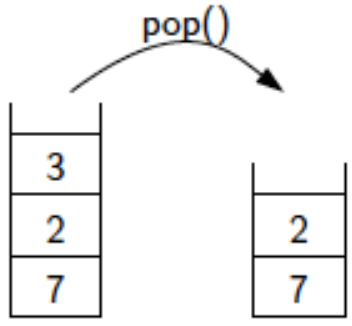
Exécution d'un test

- Scénario de test :
  - Préambule : Suite d'actions amenant le programme dans l'état nécessaire pour exécuter le cas de test
  - Corps : Exécution des fonctions du cas de test
  - Identification (facultatif) : Opérations d'observation rendant l'oracle possible
  - Postambule : Suite d'actions permettant de revenir à un état initial

Exécution de test

Ex : Pop (supprimer le sommet d'une pile)

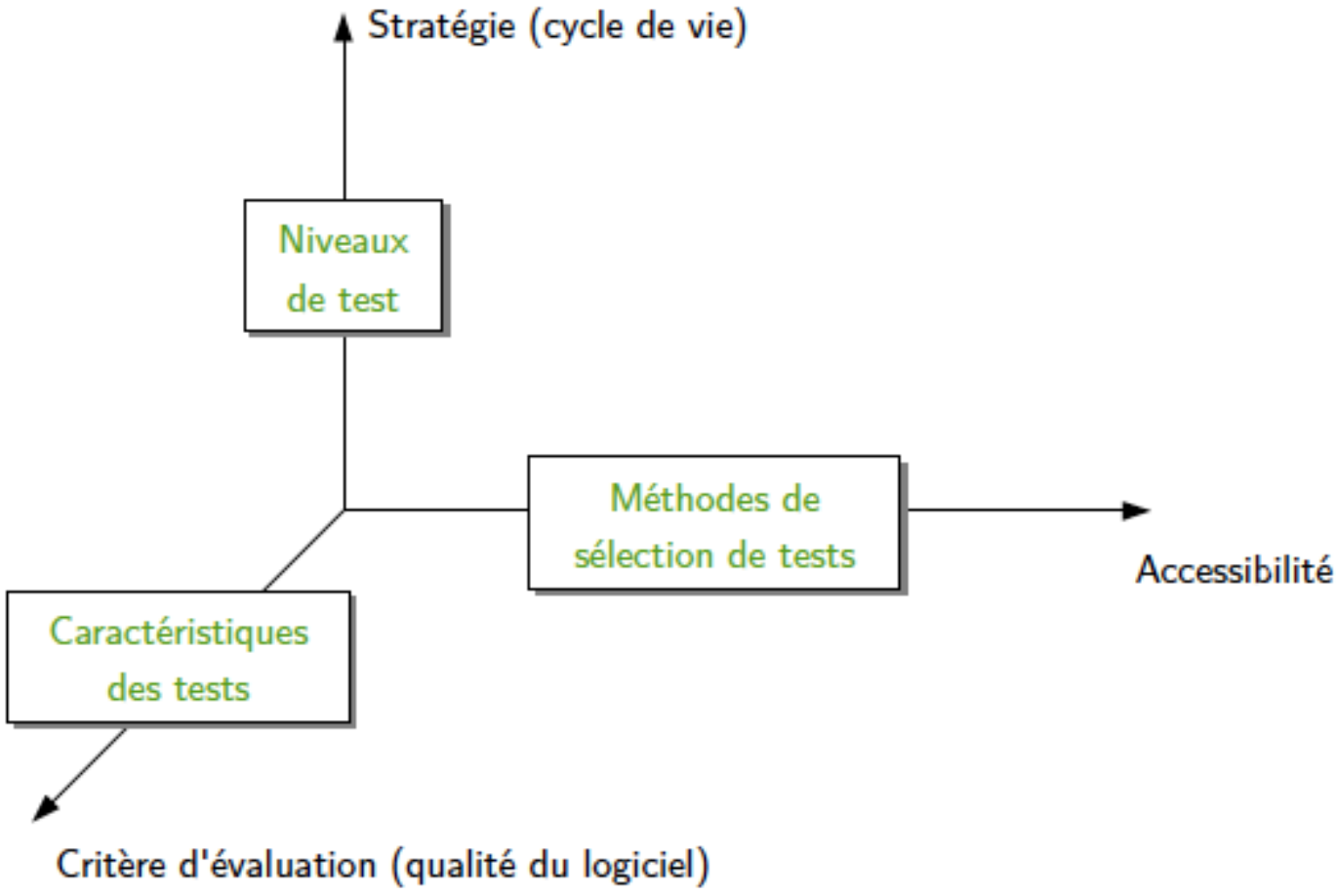
Cas de test :



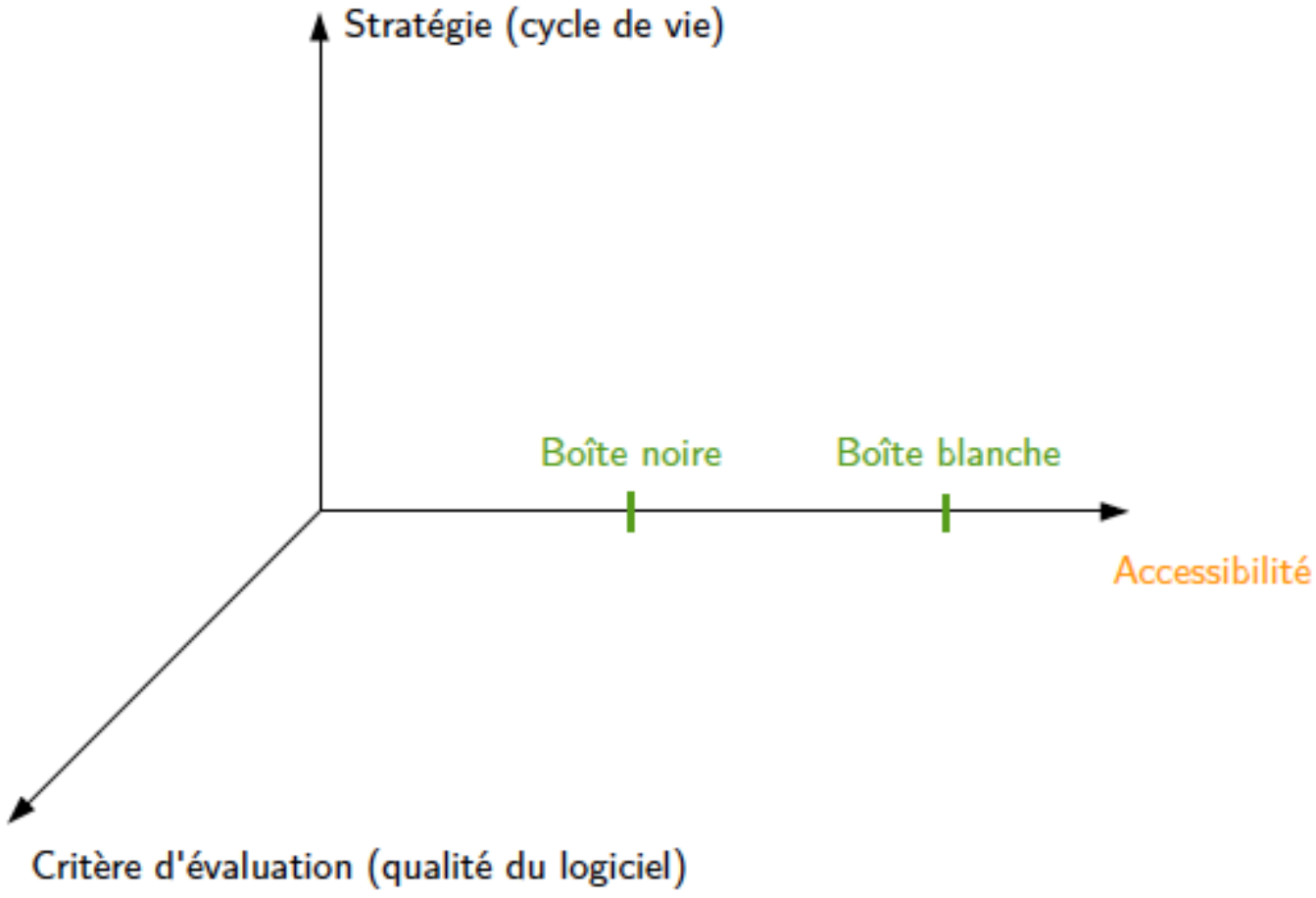
Exécution du test :

Préambule	push(7)
	push(2)
	push(3)
Corps	pop()
Identification	top() = 2
	pop()
	top() = 7
	pop()
	top() = *empty*

Types de test

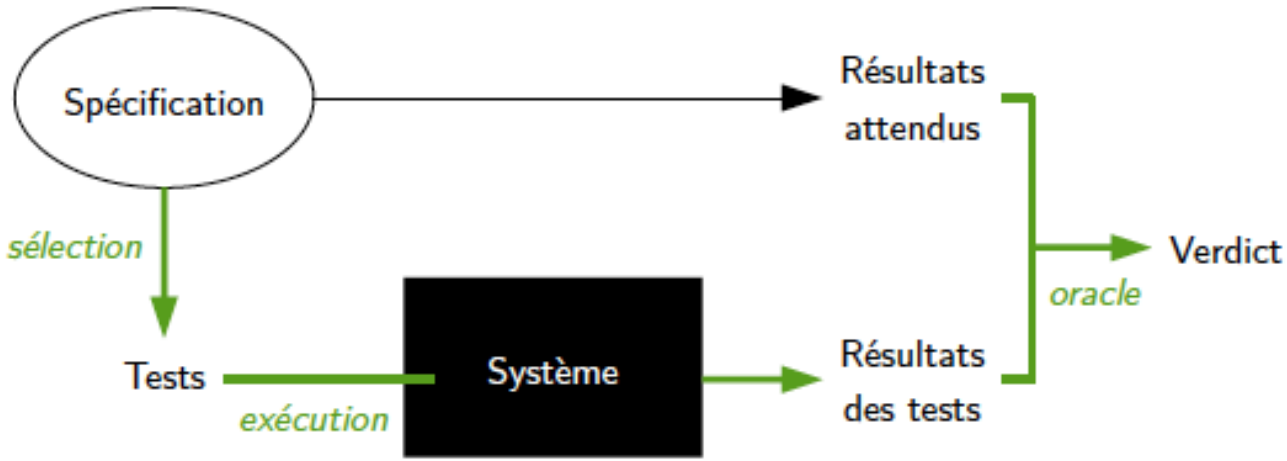


Types de test



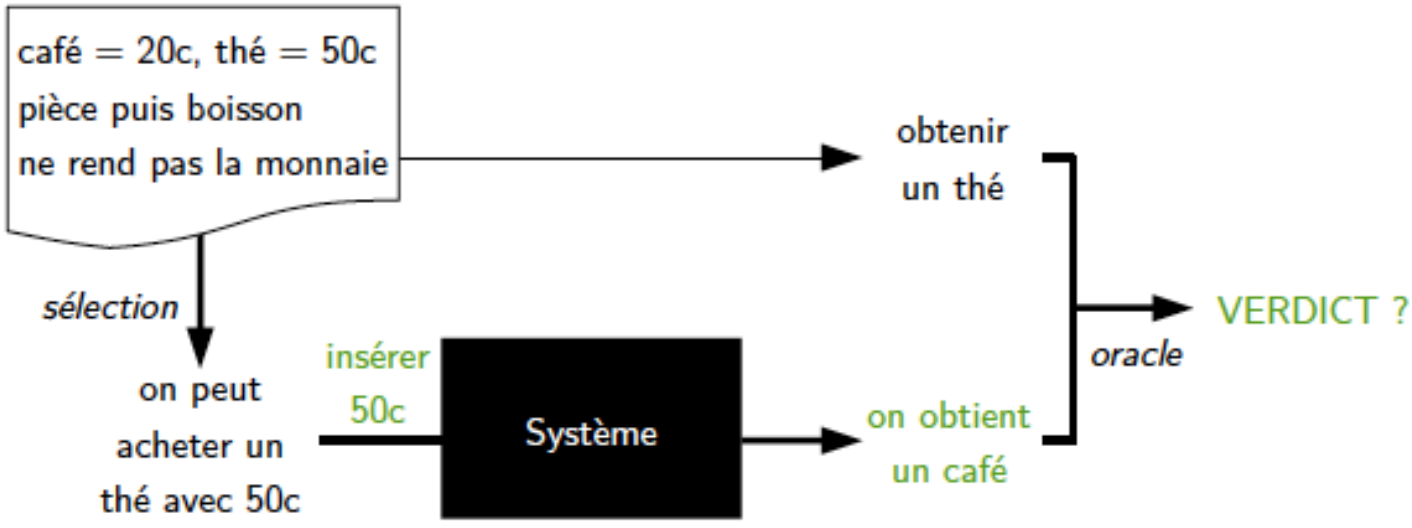
Test en boîte noire

- Sélection des tests à partir d'une spécification du système (formelle ou informelle), sans connaissance de l'implantation.
  - Test « fonctionnel »
- Possibilité de construire les tests pendant la conception, avant le codage



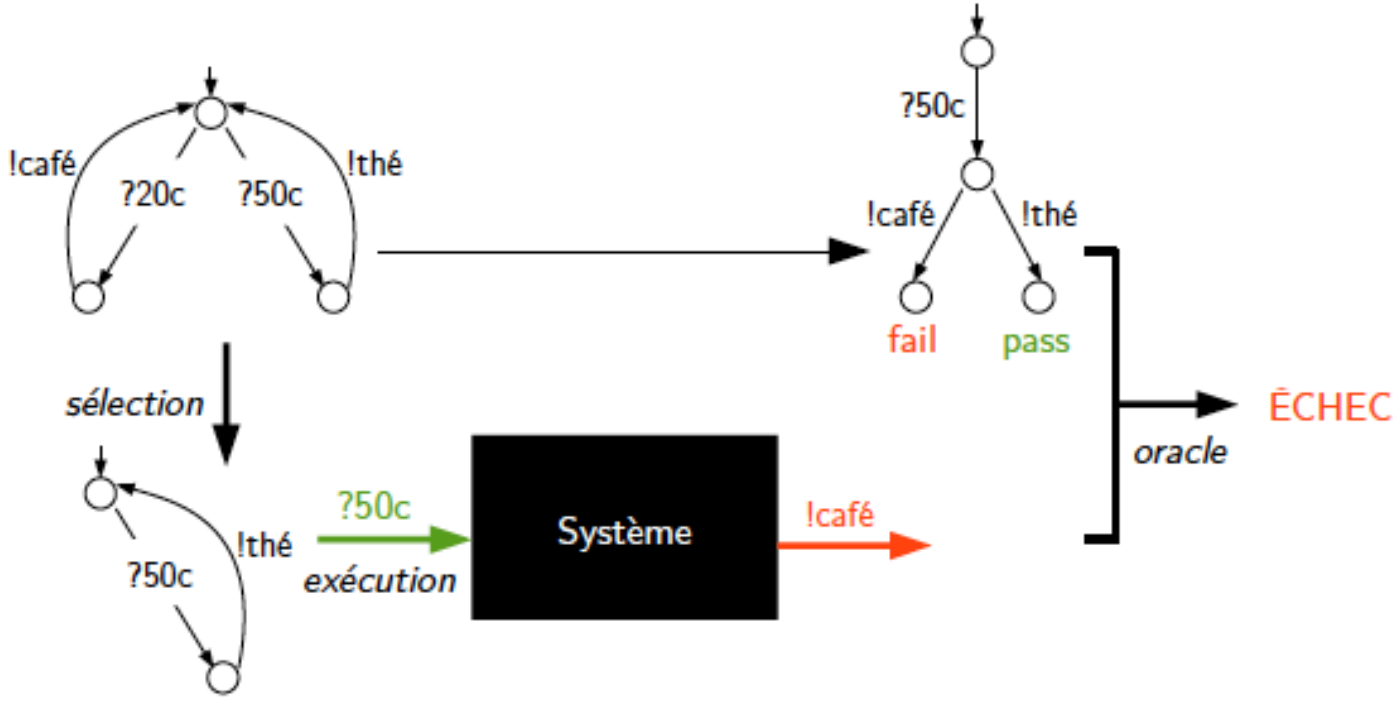
Test en boîte noire

Ex : Distributeur de café et thé



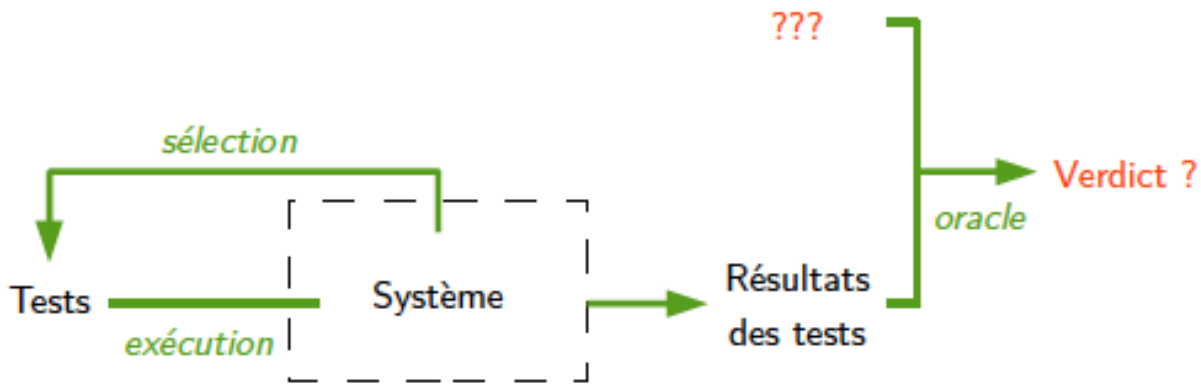
Test en boîte noire

Ex : Distributeur de café et thé



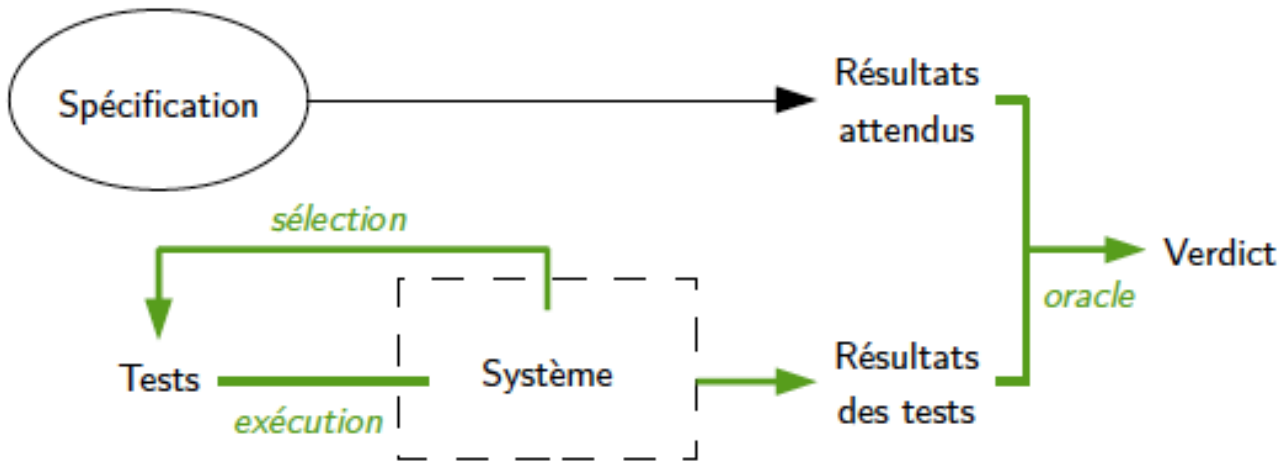
Test en boîte blanche

- Sélection des tests à partir de l'analyse du code source du système
  - Test « structurel »
- Construction des tests uniquement pour du code déjà écrit !

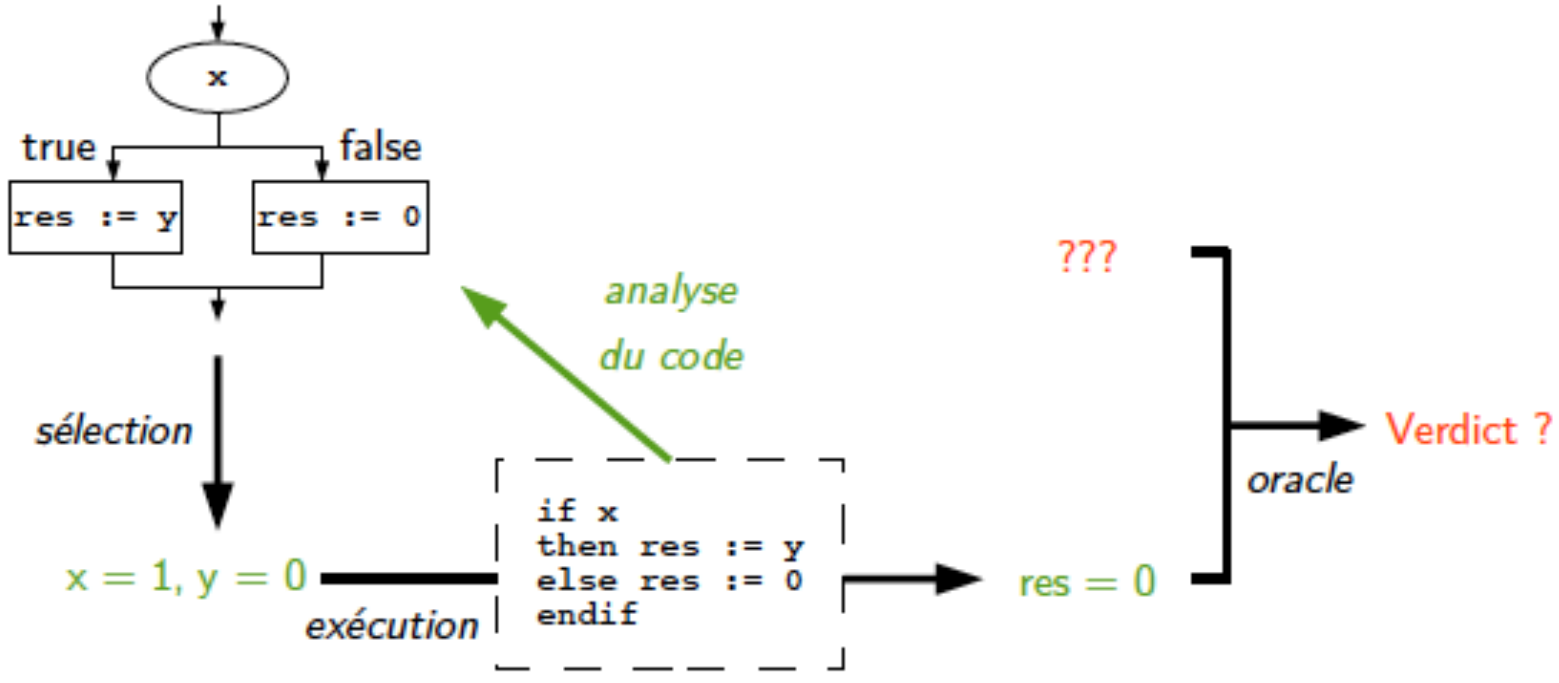


Test en boîte blanche

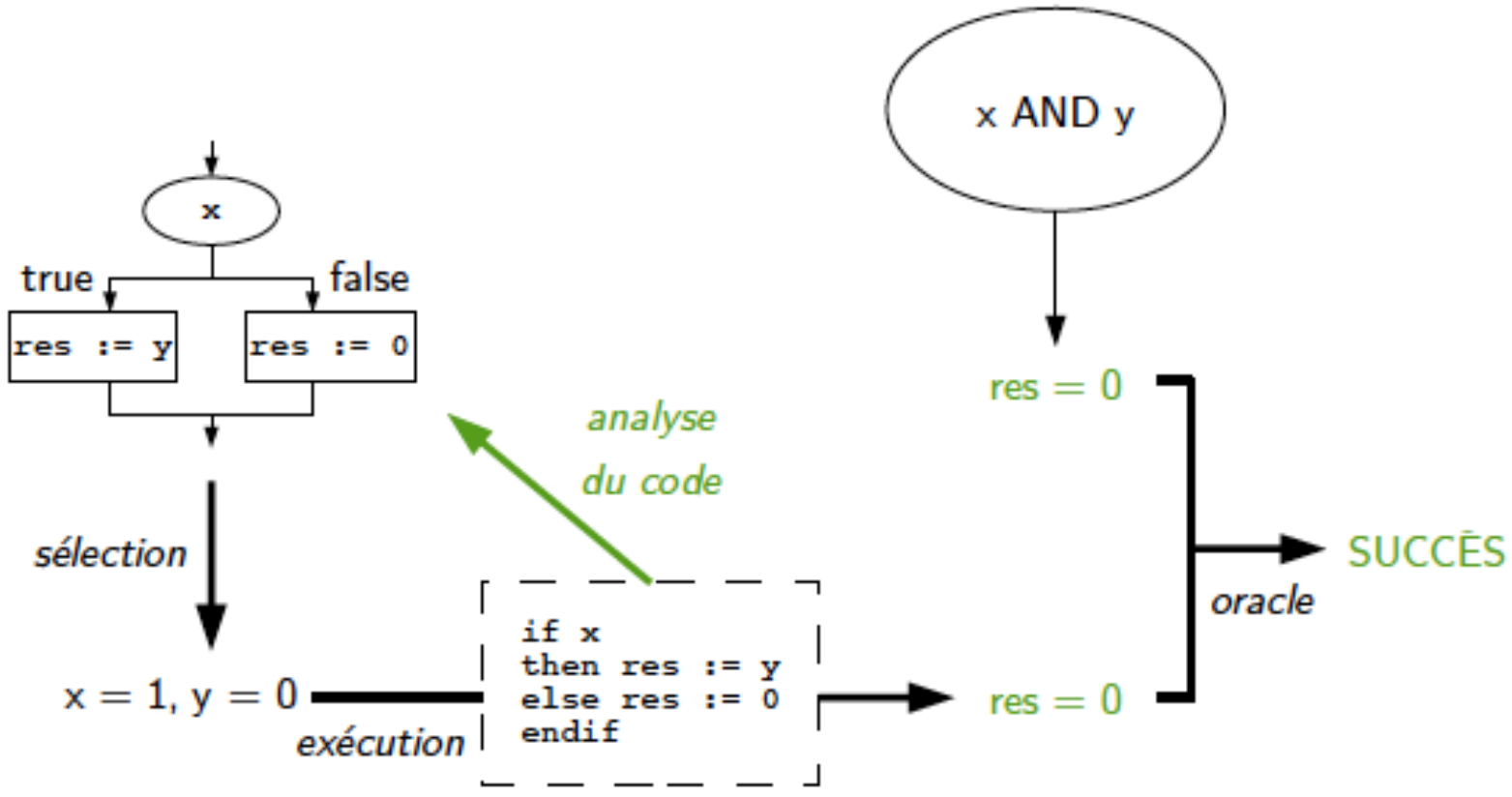
- Sélection des tests à partir de l'analyse du code source du système
  - Test « structurel »
- Construction des tests uniquement pour du code déjà écrit !



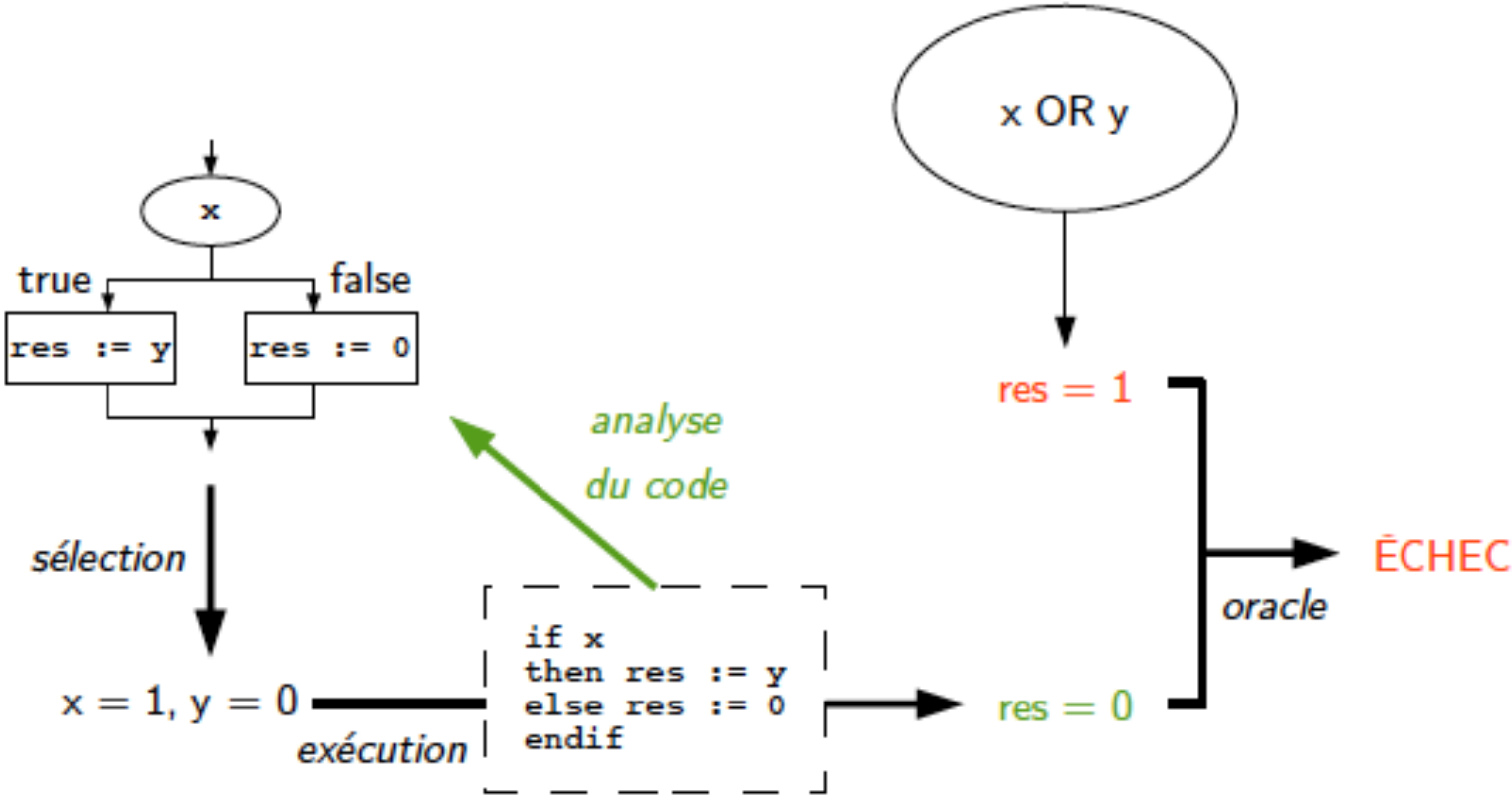
Test en boîte blanche



Test en boîte blanche



Test en boîte blanche



Test en boîte blanche

- Comment sélectionner des données de test qui détectent le plus d'erreurs ?
- Réponse : Couvrir les
  - Instructions du programme
  - Changement de contrôle du programme
  - Décisions du programme
  - Comportements du programme

simple

impossible

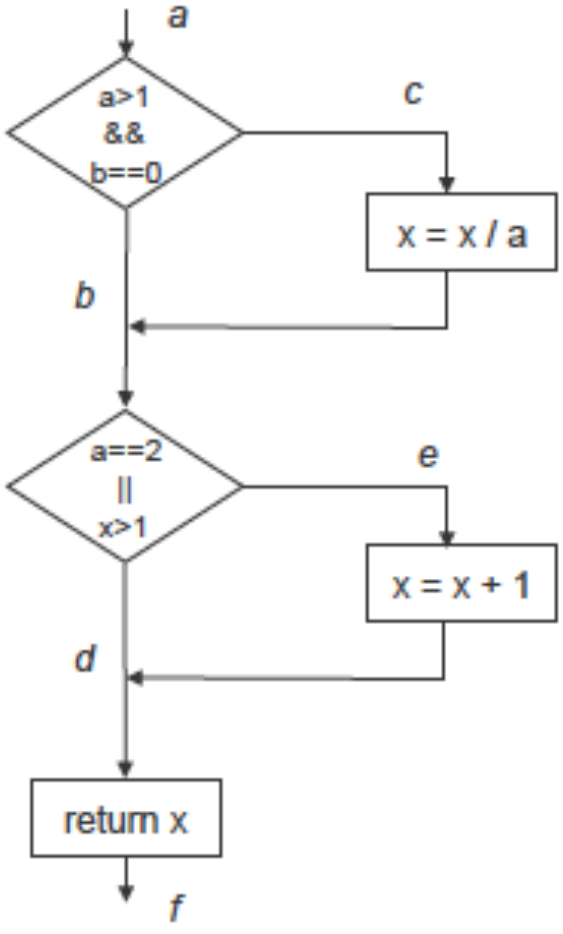
  - Flot de données (définition -> utilisation de variable)

Test en boîte blanche

```
int foo (int a, int b, int x) {
    if ((a>1) && (b==0))
        x = x div a;
    if ((a==2) || (x>1))
           x = x + 1;
    return x;
}
```

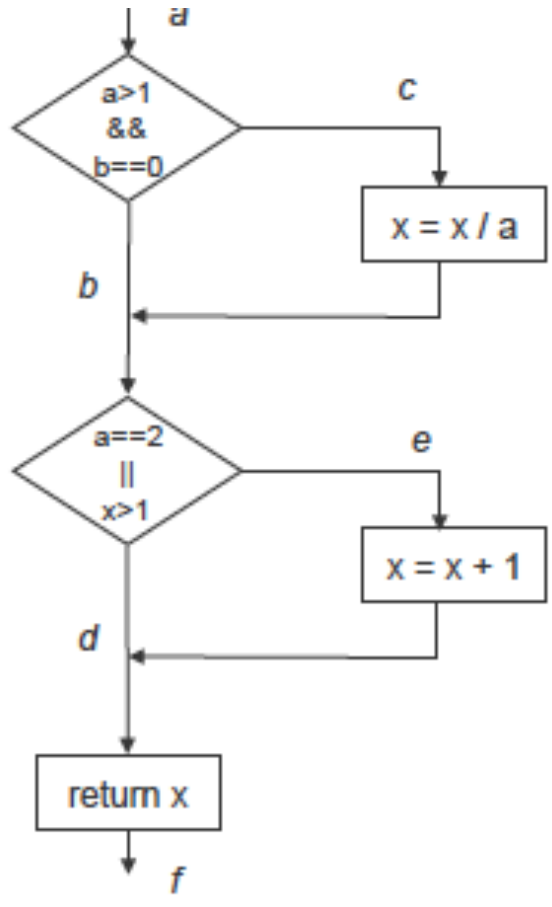
C code

Flow control graph



Test en boîte blanche

- Couverture des instructions
  - Exemple : acef
  - Données de test : a=2, b=0, x=3
  - Question : Et si le && était un || ?
  - Question : Et le chemin abdf ?
  - Très peu exhaustive, peu utilisé !



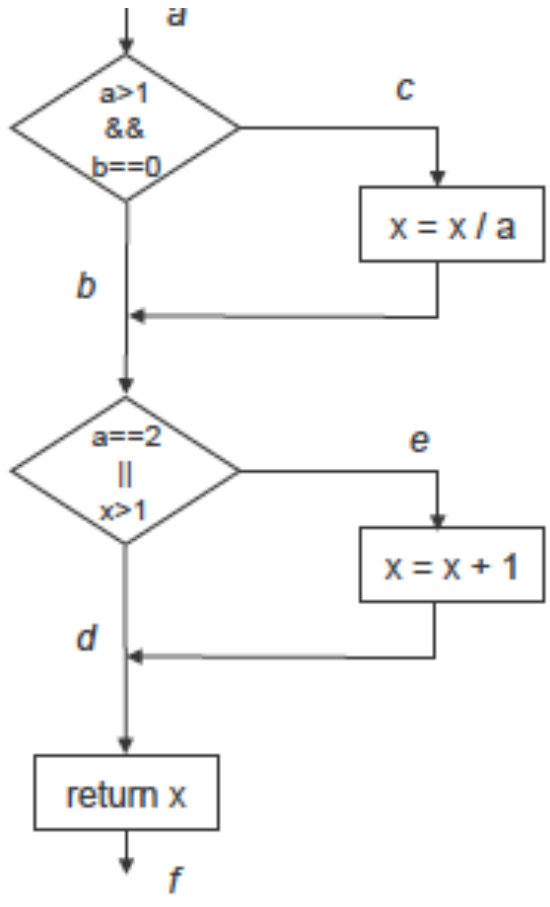
Test en boîte blanche

- Couverture des décisions

= chaque décision est vraie ou fausse

au moins une fois

  - Exemple : {acef,abdf} ou {acdf,abef}
  - Données de test :
    - a=2, b=0, x=3
    - a=2, b=1, x=1
  - Question : Et si x > 1 était incorrecte ?

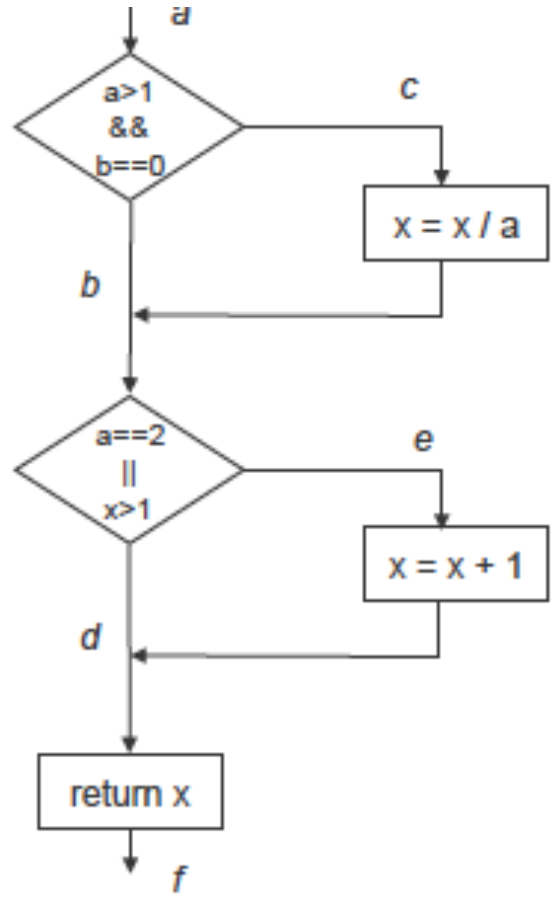


Test en boîte blanche

- Couverture de plusieurs décisions

= toutes les combinaison de valeurs pour les décisions sont testées au moins une fois

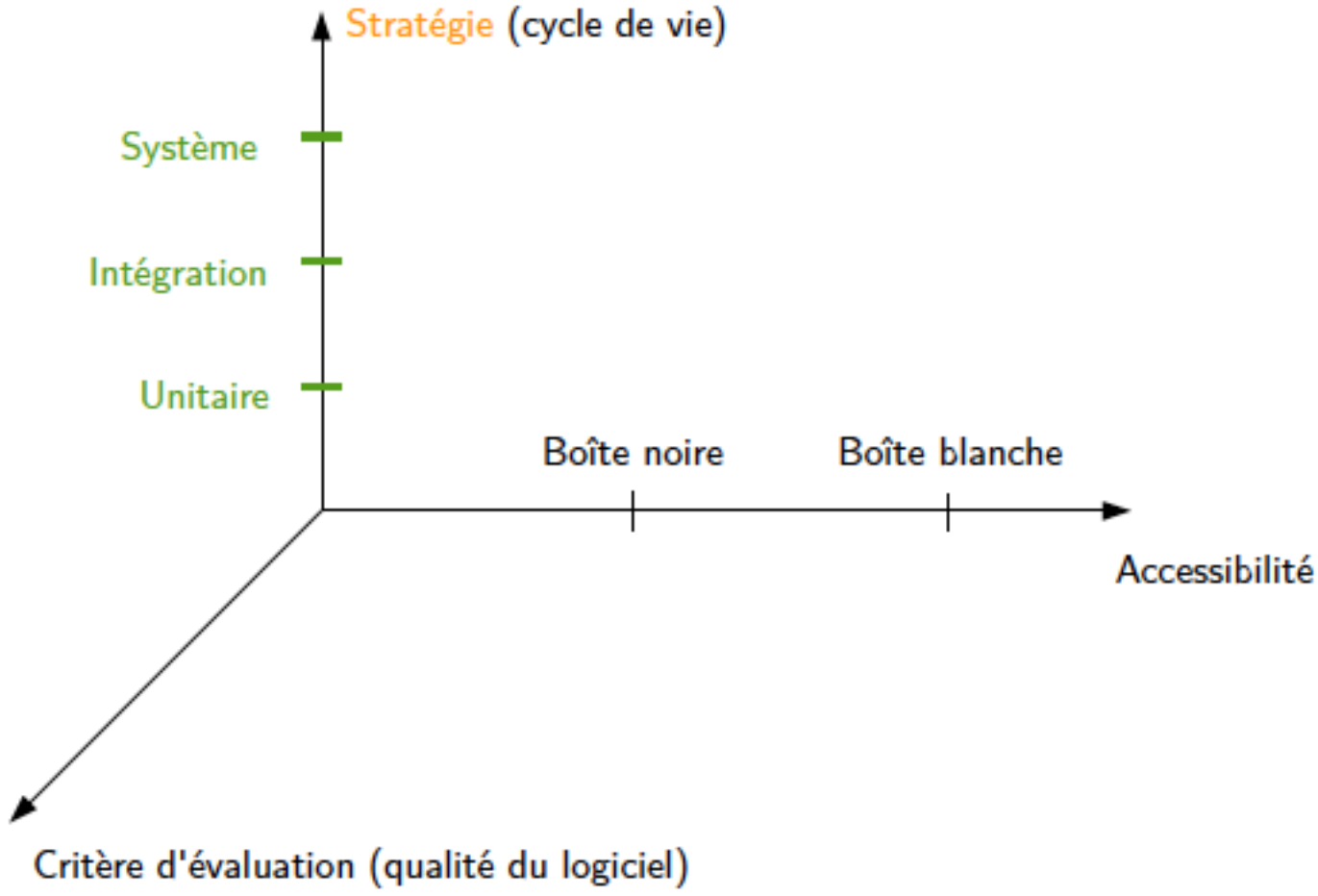
  - Exemple :
    1. a>1,b=0
    2. a=2,x>1
    3. a>1,b<>0
    4. a=2,x<=1
    5. a<=1,b=0
    6. a<>2,x>1
    7. a<=1,b<>0
    8. a<>2,x<=1
  - Données de test :
    - a=2, b=0, x=4 couvre 1 & 2
    - a=2, b=1, x=1 couvre 3 & 4
    - a=1, b=0, x=2 couvre 5 & 6
    - a=1, b=1, x=1 couvre 7 & 8
  - Toutes les exécutions sont couvertes ?



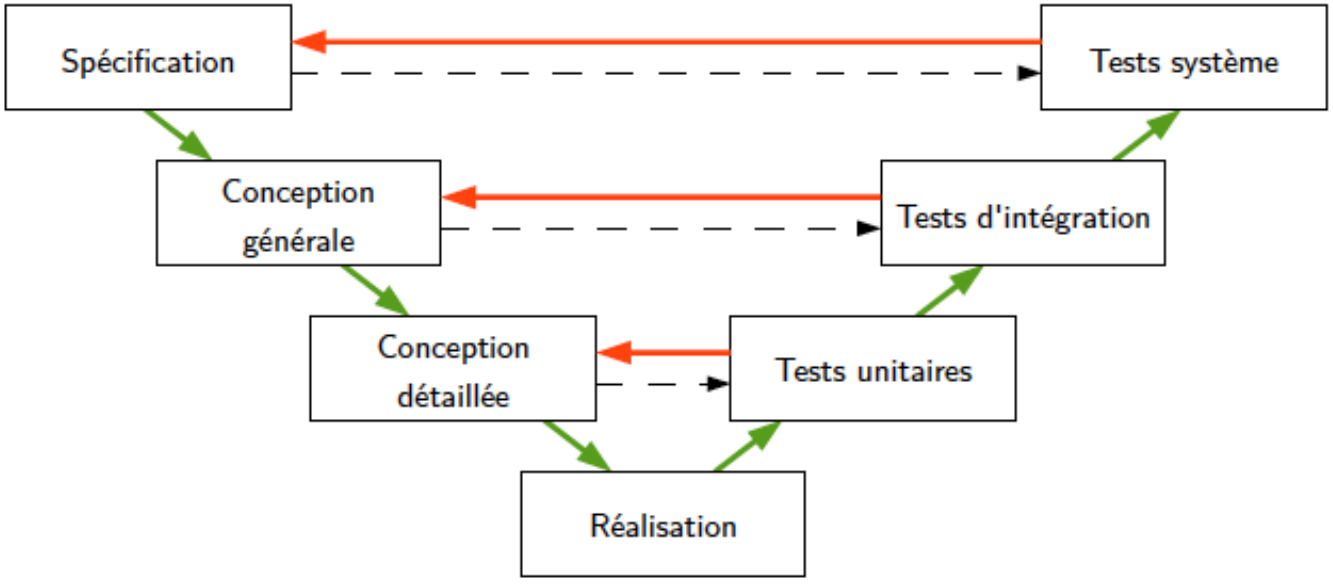
Boîte noire vs. boîte blanche

- Complémentarité : détection de fautes différentes
  - Boîte noire : détecte les oublis ou les erreurs par rapport à la spécification
  - Boîte blanche : détecte les erreurs de programmation

Types de test

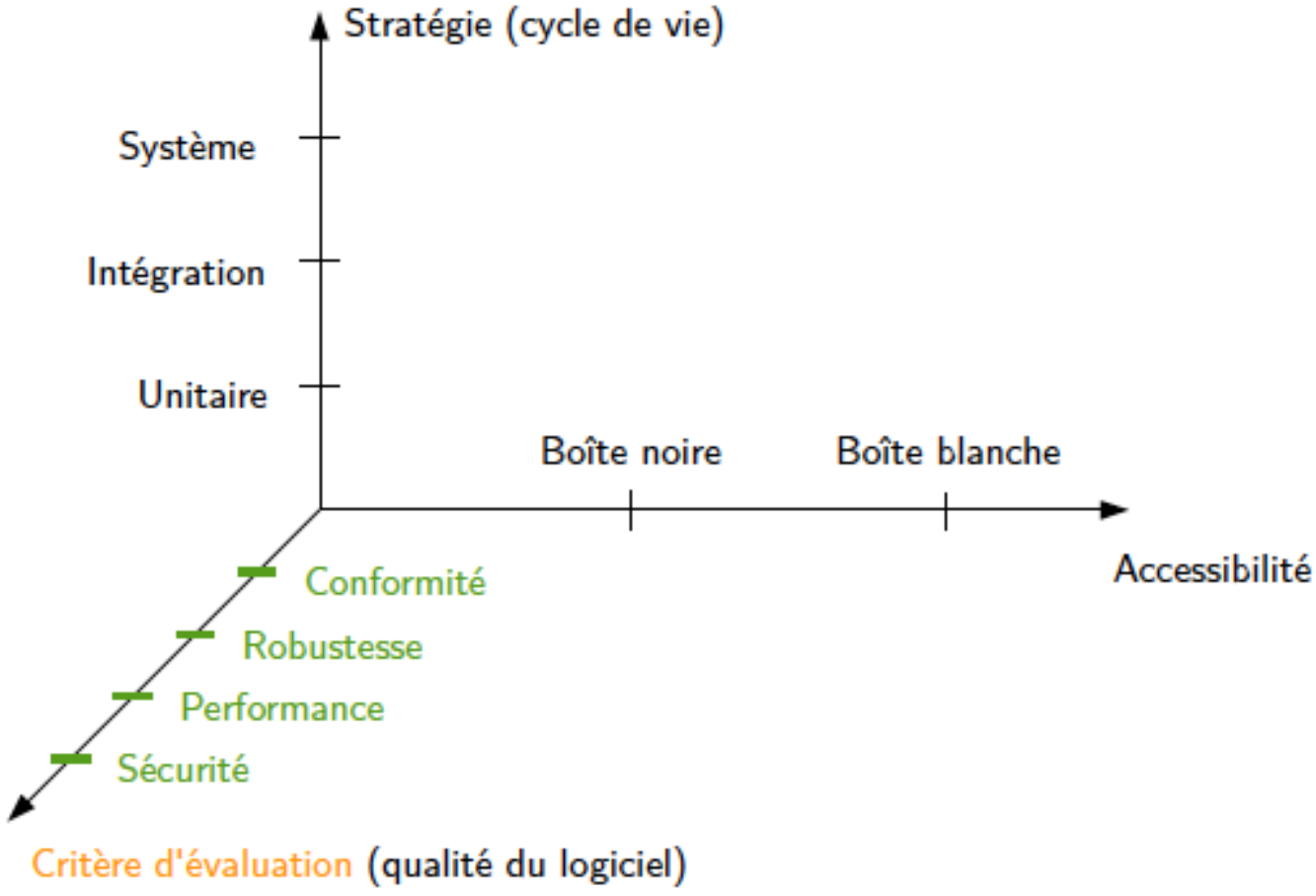


Phases de production d'un logiciel



- Test unitaire = test des (petites) parties du code, séparément.
- Test d'intégration = test de la composition de modules.
- Test du système = de la conformité du produit fini par rapport au cahier des charges, effectué en boîte noire.

Types de test



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
- *Ex : Service de paiement en ligne*
  - 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...)
- *Ex : Service de paiement en ligne*
  - Login dépassant la taille du buffer
  - Coupure réseau pendant la transaction

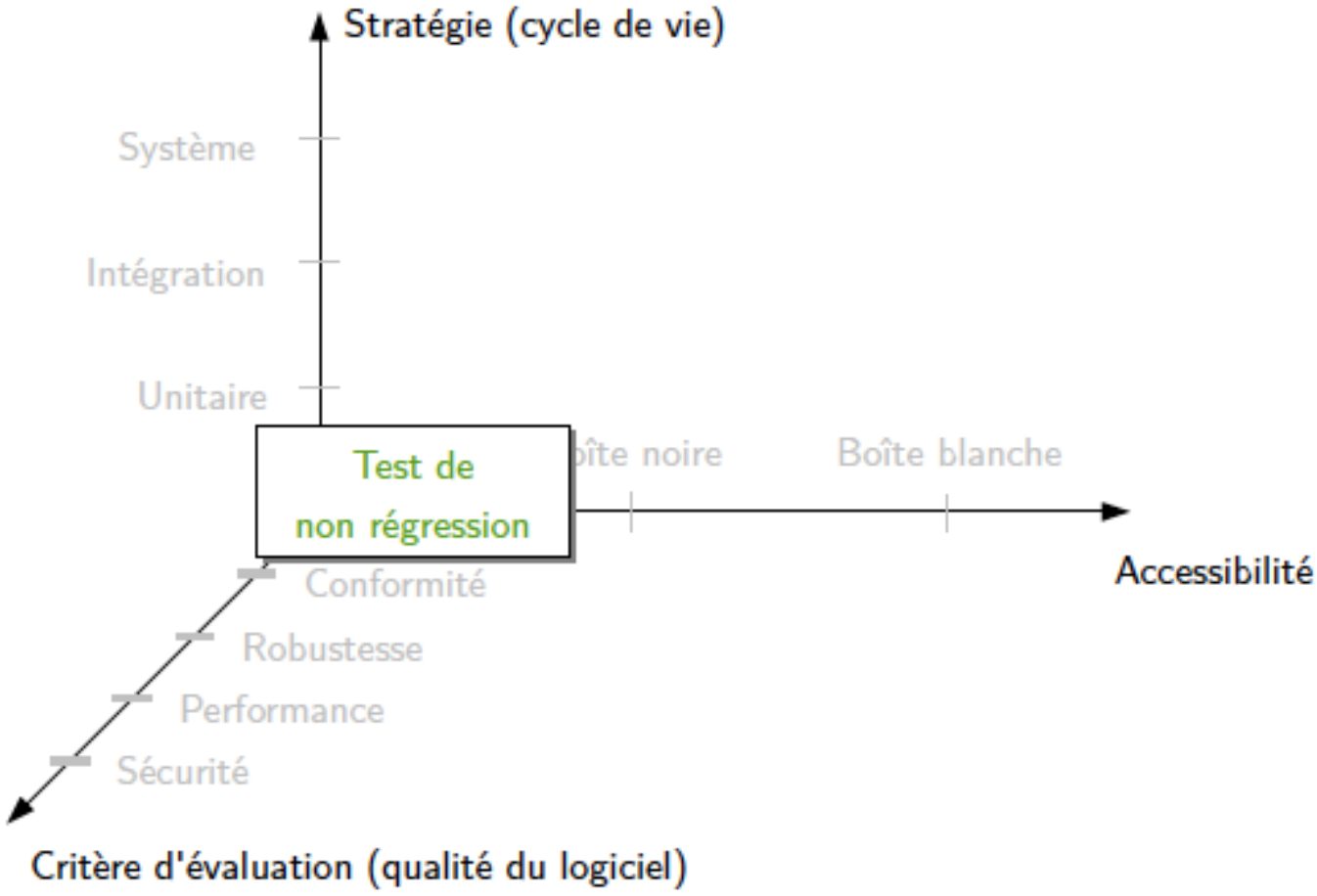
Test de sécurité

- 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
- *Ex : Service de paiement en ligne*
  - 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...
- *Ex : Service de paiement en ligne*
  - Lancer plusieurs centaines puis milliers de transactions en même temps

Types de test



Test de non régression

- But : Assurer que les corrections et les évolutions du code n'ont pas introduit de nouveaux défauts
- Méthode : À chaque ajout ou modification de fonctionnalité, rejouer les tests pour cette fonctionnalité, puis pour celles qui en dépendent, puis les tests des niveaux supérieurs
  - Lourd mais indispensable
  - Automatisable en grande partie

Automatisation des tests

- Outils :
  - Générateur de test : aléatoire ou guidé par propriétés (BN) ou par des critères de couverture (BB).
    - Microsoft : DART, SAGE
    - INRIA : TGV
  - Analyseur de couverture : calcule le pourcentage de code couvert durant le test.
    - Coverlipse, gcov
  - "Record & playback" (Exécutif de test) : enregistre les actions de l'utilisateur pour pouvoir les rejouer à la demande ; utile pour le test des IHM et le test de régression.
  - Gestionnaire de test : maintient des suites de test, leurs résultat et produit des rapports.
    - Xunit (avec X=C, Java, Python)