Introduction au test logiciel

Fabrice Ambert, Fabrice Bouquet

prenom.nom@femto-st.fr

MiniVOL 2016
Orsay
16-18 novembre 2016

Automatisation des tests logiciels

- Résumé:
  - appréhender les méthodes et outils pour automatiser les tests logiciels (tests unitaires, fonctionnels, d'intégration, de charge) afin de gagner en productivité et en sureté pour le développement logiciel
- Durée:
  - 4 jours
- Pré-requis:
  - connaissances de base en test logiciel et capacité à effectuer des développements logiciels objet

Bienvenue

Faisons connaissance

- Votre formateur
- Pour chaque participant
  - Vos activités actuelles ?
  - Votre expérience ?
    - Sur des projets de développement de logiciels et systèmes d'information
    - Sur les activités de test (Rôles, Outils utilisés)
  - Vos attentes pour cette formation ?

Contenu de la formation

Programme standard

- Introduction, rappels sur le processus du test
- Automatisation de la gestion des tests
- Automatisation des tests unitaires
- Automatisation des tests d'intégration
- Automatisation du test fonctionnel
- Automatisation des tests système
- Synthèse

Déroulement de la formation (3 jours)

Alternance de Cours et de Pratique

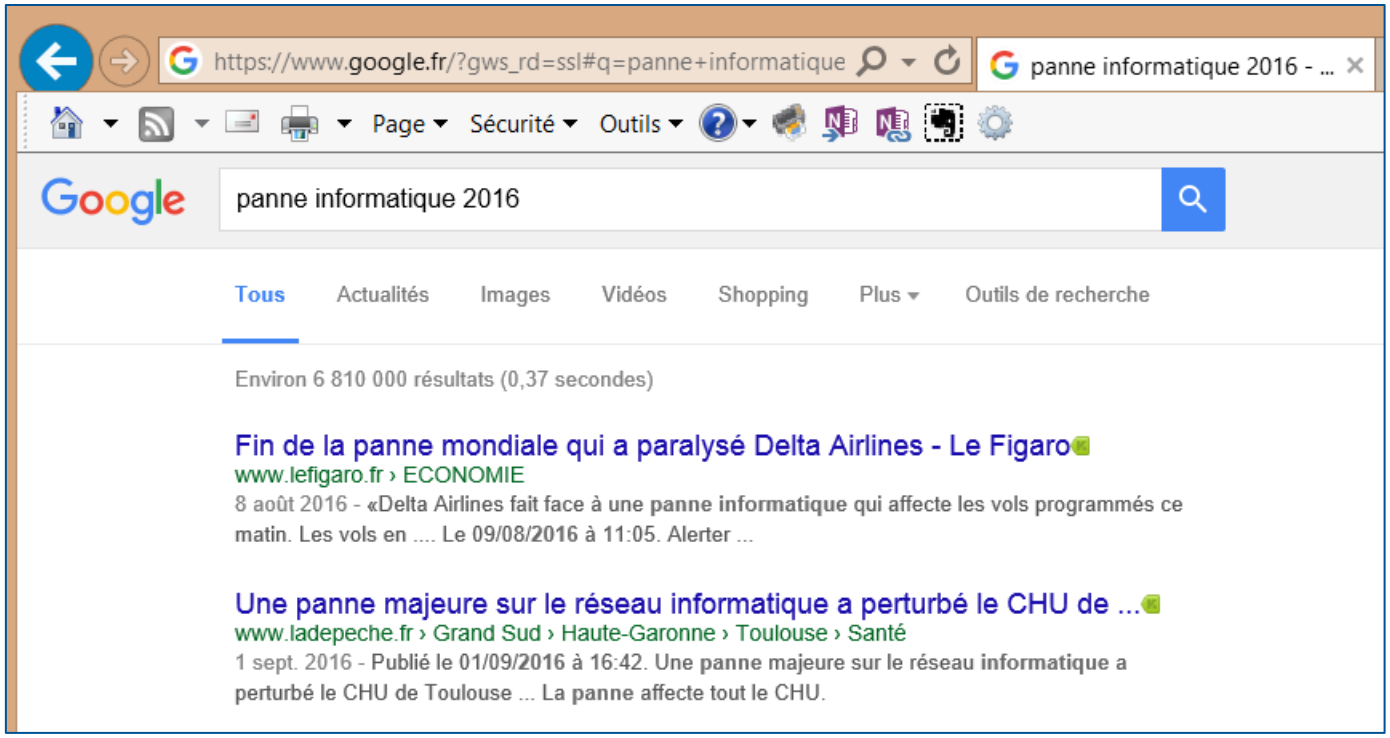
Cours	Thèmes	Durée	Exercices	Outils	Durée
1. Introduction, rappels sur le processus du test logiciel	Types, Niveaux, Méthodes	J1, 9h-11h			
2. Automatisation des tests unitaires	Couverture de Code	J1, 11h30-12h30	Intégrés au cours Cahier d'exercices	Junit, mockito, coverTools	J1 13h30-17h J2 9h-11h
3. Automatisation des tests d'intégration	Stratégie, Intégration continue	J2, 11h30-12h30	Cahier d'exercices	Maven, Jenkins, SOnarQube	J2, 13h30-16h
4. Test fonctionnel et son automatisation	Partitions, Limites, Combinatoire	J2, 16h-17h	Intégrés au cours	Concordion Selenium, HTMLUNit	9h-10h30 11h-12h30
5. Automatisation de la gestion des tests	Conception, exécution, suivi	J3, 13h30-14h30	Cahier d'exercices	Squash TM / TA	J3, 14h30-…

1. Introduction, rappels sur le processus du test logiciel

Rôle du test dans le processus de développement.
Les tests : unitaires, fonctionnels, etc.
Les différentes méthodes de test.
Processus de test et stratégie de test.
Outils et méthodes intervenant à différentes étapes

Rôle du test dans le processus de développement

Les bugs sont nombreux et couteux pour la société possédant le logiciel, ses fournisseurs, ses clients, les utilisateurs finaux…



Les failles de sécurité sont également nombreuses

Problèmes de sécurité (Figaro 21/09/16)

En avril 2015, la chaine TV Monde avait été contrainte de suspendre la diffusion de ses programmes après l'attaque informatique dont elle avait été victime. GEOFFROY VAN DER HASSELT/ANADOLU AGENCY

Les attaques informatiques sont en forte hausse

Le secteur médical est particulièrement touché par des failles de sécurité, en France et dans le monde.

Le Figaro 21 septembre 2016

LUCIE RONFAUT @LucieRonfaut

INTERNET Sur les services en ligne, au sein des gouvernements ou même dans les hôpitaux. Les attaques informatiques sont en hausse et touchent tous les endroits de notre vie numérique. Près de 974 vols de données personnelles ont été détectés au premier semestre 2016 par Gemalto, entreprise spécialisée dans la sécurité en ligne, qui publiait mardi son étude Breach Level Index. Cela représente une augmentation de 15 % en six mois, par rapport aux chiffres du dernier semestre 2015. En tout, plus de 554 millions de jeux de données ont été compromis sur les six premiers mois de 2016, contre 424 millions fin 2015. La fraude à l'identité est la première raison de ces attaques, avec 621 brèches détectées. Vient ensuite le piratage de comptes bancaires, avec 155 failles exploitées.

Le secteur médical a été particulièrement touché par le phénomène, avec plus de 263 incidents répertoriés par Gemalto début 2016, en hausse de 25 % sur les six derniers mois. Ces données sont particulièrement sensibles, et souvent mal protégées. «*Un vol impliquant 100 millions de noms d'utilisateurs sera de moindre gravité que celui qui concerne 1 million de comptes comprenant des numéros de sécurité sociale ou d'autres informations personnelles pouvant être utilisés à des fins financières*», estime Jason Hart, vice président de Gemalto. En février, une clinique américaine, en Californie, a été complètement paralysée par un virus informatique de type *ransomware*, provoquant la paralysie de l'établissement pendant plusieurs semaines et le vol de données sensibles de ses patients: fiches d'admission, dossiers médicaux... Ces attaques sont de plus en plus fréquentes mais touchent encore un nombre limité d'informations dérobées dans le monde. Les services gouvernementaux représentent plus de 37 % du nombre de jeux de données dérobés depuis le début de l'année. Vient ensuite le secteur technologique (applications, sites Web, etc.), d'où proviennent 16 % des informations volatilisées. Cet équilibre pourrait néanmoins être bouleversé au fur et à mesure de la numérisation des données de santé, un processus qui s'accélère grâce au développement des objets connectés et de l'intelligence artificielle au service du médical.

Des nouveaux réflexes

En juillet, l'Agence nationale de la sécurité des systèmes d'information (Anssi) a déclaré le secteur de la santé comme l'un des «*opérateurs d'importance vitale*» devant renforcer leur système informatique pour se protéger des attaques. Sont également concernés les gestionnaires d'eau, d'alimentation, le nucléaire et la justice. D'autres secteurs sont déjà habitués à surveiller leurs réseaux, comme les opérateurs télécoms, les banques ou l'aérospatial. Le nouveau cadre juridique français doit pousser les autres à adopter des règles plus strictes en matière de sécurité et à notifier à l'Anssi les incidents.

D'après Gemalto, l'Amérique du Nord, et plus particulièrement les États-Unis, reste la région la plus touchée par les attaques informatiques. Plus de 79 % des incidents ont touché des services américains. Vient ensuite l'Europe, concernée par plus de 9 % des failles informatiques exploitées par des pirates d'ici but 2016. Cela représente une baisse de 27 %, d'après Gemalto. L'Asie-Pacifique a polarisé près de 8 % des attaques répertoriées.

La France n'est pourtant pas épargnée par la hausse globale des attaques informatiques. Dans son rapport annuel, publié au début du mois, l'Anssi a fait état de 4000 signalements reçus pour des problèmes de cybersécurité, 50 % de plus qu'en 2014. L'Agence a également évoqué «*une véritable d'incidents majeurs de sécurité*» qu'elle a traités en 2015. Elle désigne par là des attaques de grande ampleur lancées contre la France. Les victimes sont néanmoins inconnues pour la plupart, par souci de confidentialité. Seule TV5 Monde a été forcée de rendre publique une attaque d'ampleur dont elle a fait l'objet en avril 2015, occasionnant l'arrêt de la diffusion de ses programmes et la publication de messages de soutien à l'État islamique sur les réseaux sociaux. ■

LES ATTAQUES INFORMATIQUES EN 2016
SOURCES : BREACH LEVEL INDEX, GEMALTO
974 vols de données
554 millions de jeux de données dérobés
15 % d'augmentation entre le deuxième semestre 2015 et le premier semestre 2016

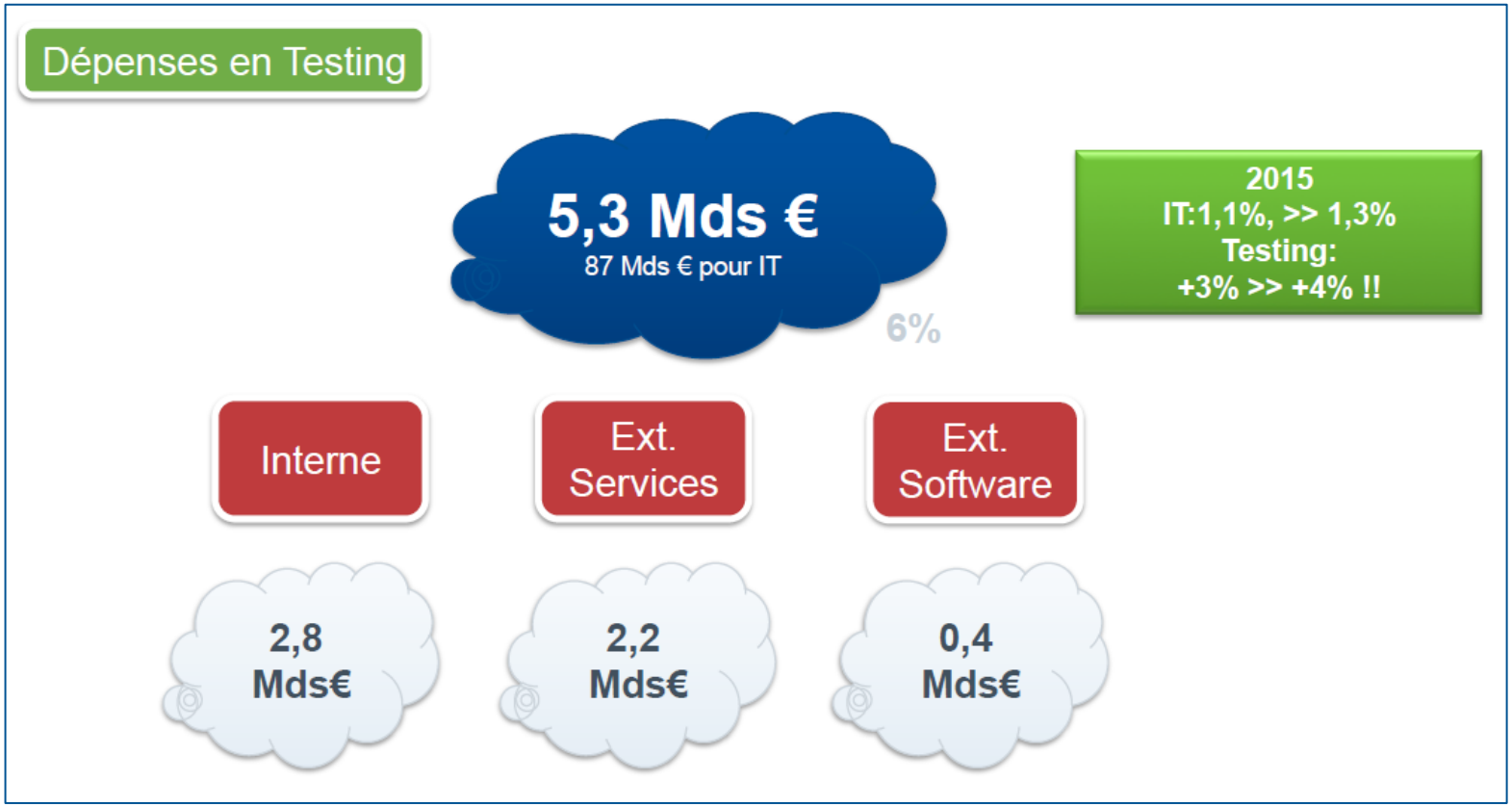
Les DSI ont des objectifs directement liés au test

Source: « World Quality Report 2016-2017, Sogeti »

65% Améliorer la sécurité → 44% Augmenter la Qualité des Solutions → 42% Améliorer l'Expérience Utilisateur → 42% Optimiser les coûts

Le Test continue a être un investissement nécessaire et important

Une place très importante dans l'IT



Source: PAC 2016, Arnold Aumasson

Définitions du test

- *« 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. »*
  IEEE (Standard Glossary of Software Engineering Terminology)

- *« Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts. »*
  G. Myers (The Art of Software testing)

- *«Testing can reveal the presence of errors but never their absence.»* Edsgar W. Dijkstra (Notes on Structured Programming)

La réalité du test

- Le test constitue aujourd'hui le vecteur principal de l'amélioration de la qualité du logiciel

- Actuellement, le test dynamique est la méthode la plus diffusée
- Il peut représenter jusqu'à 60 % de l'effort complet de développement d'un logiciel

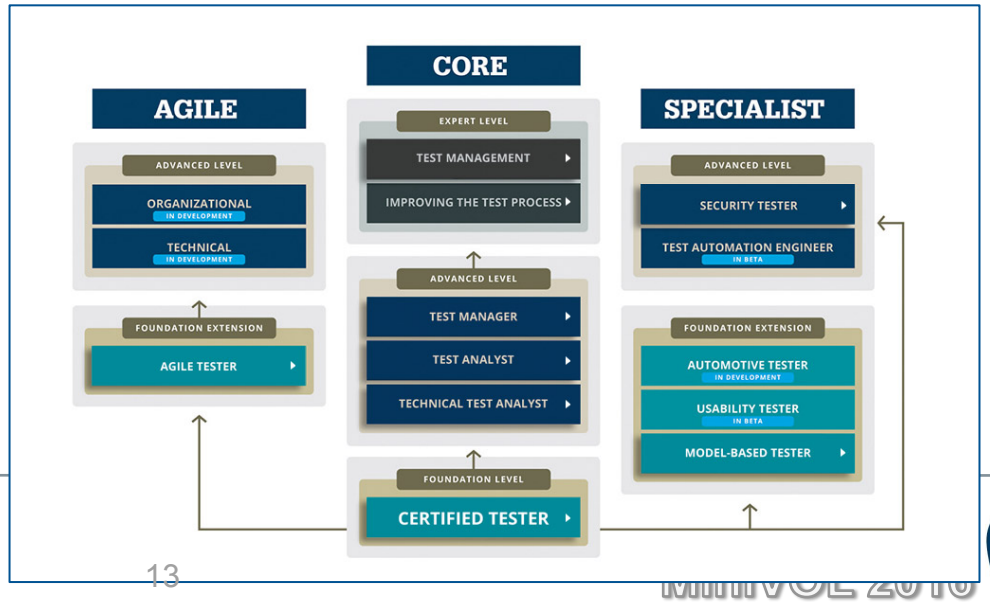
- Coût moyen de l'activité de test :
  - 1/3 durant le développement du logiciel
  - 2/3 durant la maintenance du logiciel

Industrialiser et professionnaliser le test logiciel avec des compétences spécifiques

Un schéma mondial de Formations et Certifications en Test Logiciel

- ISTQB: International Software Testing Qualifications Board
  - www.istqb.org
  - Un parcours de formations reconnu dans le monde entier
  - Près de **500 000 certificats délivrés dans le monde**

ISTQB®
International Software Testing Qualifications Board



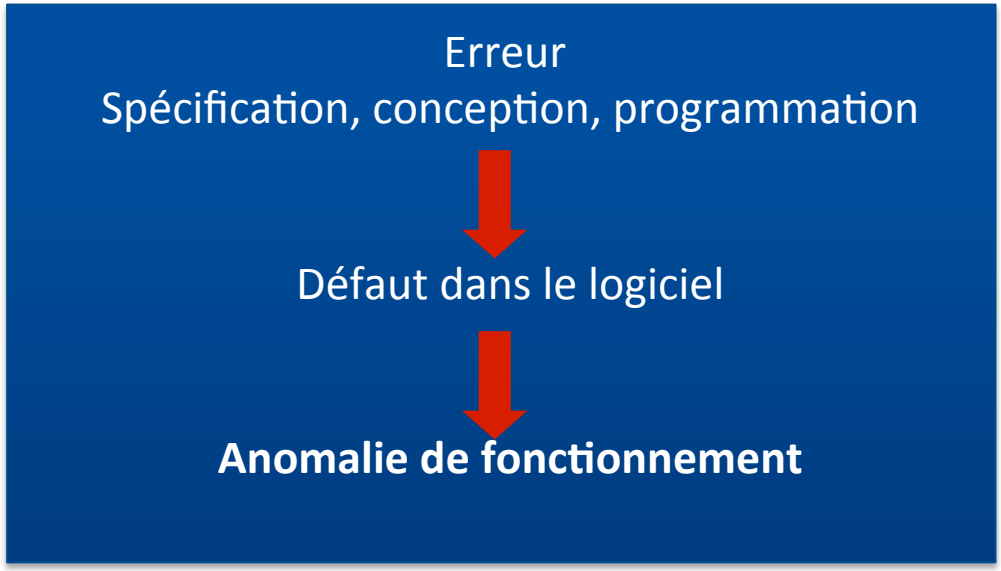
Industrialiser et professionnaliser le test logiciel avec des compétences spécifiques

Une implantation en France très importante

- CFTL: Comité Français des Tests Logiciels
  - www.cftl.fr
  - Déploiement en France
    - Des certifications ISTQB
    - De certifications connexes au test
  - Près de **10000 certificats délivrés en France**

Motivations du test

- Complexité croissante des architectures et des comportements
- Coût d'un bug (Ariane 5, carte à puces allemande bloquée, prime de la PAC…)



- Coût des bugs informatiques : ≈ 60 milliards $ / an
- 22 milliards économisés si les procédures de test de logiciels étaient améliorées.

(source : NIST - National Institute of Standards and Technology)

Les tests : unitaires, fonctionnels, etc.

- V & V
  - Validation : Est-ce que le logiciel offre les services attendues ?
  - Vérification : Est-ce que les artefacts utilisés sont corrects ?

- Méthodes de V & V
  - Test statique : Revue de code, de spécifications, de documents de design
  - Test dynamique : Exécuter le code pour s'assurer d'un fonctionnement correct
  - Vérification symbolique : Run-time checking, Execution symbolique, …
  - Vérification formelle : Preuve ou model-checking d'un modèle formel

Les tests : unitaires, fonctionnels, etc

La pratique du test

- Le test appartient à l'activité de Validation du logiciel :
  est-ce-que le logiciel fait les choses bien et les bonnes choses ?

- Activité historiquement peu populaire en entreprise

- Difficultés d'ordre psychologique ou « culturel » :
  - L'activité de programmation est un processus constructif : on cherche à établir des résultats corrects
  - Le test est un processus destructif : un bon test est un test qui trouve une erreur

Les différentes méthodes de test

Test statique

Traite le code du logiciel sans l'exécuter sur des données réelles.

Test dynamique

Repose sur l'exécution effective du logiciel pour un sous ensemble bien choisi du domaine de ses entrées possibles.

Test statique

Définitions

Compilateur

- Outil logiciel qui traduit un programme exprimé dans un langage de haut niveau dans son équivalent en langage machine [IEEE 610]

Complexité

- Degré par lequel un composant ou système a une conception et/ou une structure interne difficile à comprendre, maintenir et vérifier

Test statique

L'analyse statique trouve des défauts (et non des défaillances)

- Le code n'est pas exécuté mais des outils peuvent être utilisés par des profils développeur
- La compilation peut être considérée comme du test statique outillé
- Des défauts difficiles à trouver avec des tests dynamiques peuvent être détectés
  - Dépendance dans des modèles
  - Violation de règles de sécurité et programmation
  - Maintenabilité difficile
  - Code mort
  - Variable jamais utilisée
  - Boucles infinies

Test statique

Exemples :

- *Lectures croisées / inspections*
  Vérification collégiale d'un document (programme ou spécification du logiciel)
- *Analyse d'anomalies*
  Corriger de manière statique les erreurs (typage impropre, code mort, …)

Avantages :

- Méthodes efficaces et peu coûteuses
- 60% à 95% des erreurs sont détectées lors de contrôles statiques

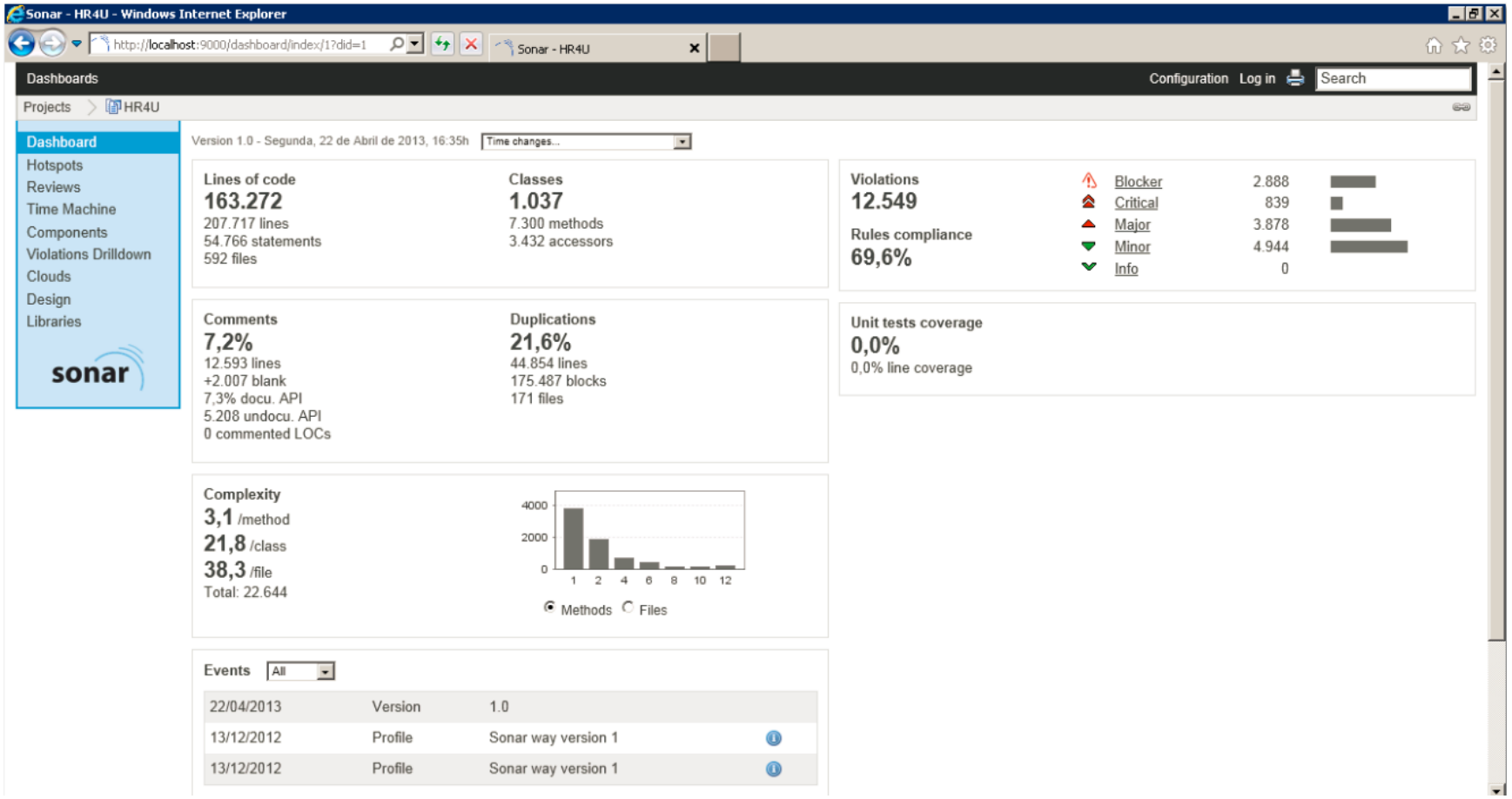
Inconvénients :

- Ne permet pas de valider le comportement d'un programme au cours de son exécution

→ Les méthodes de test statiques sont nécessaires, mais pas suffisantes

Test statique

Exemple de rapport Sonar



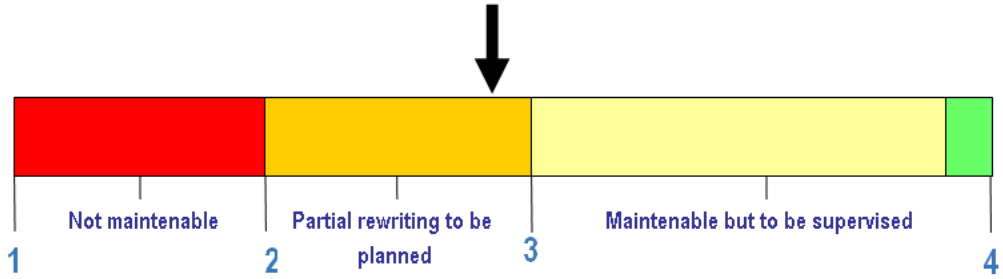
Test statique

Exemple d'analyse de code outillée

Summary	Maintenability	Transferability	Evolutivity	Robustness	Performance	Security

Technical Quality Index	2.89	2.89

Global CAST index: 2.89



Determines the cost and how the application is easily/difficultly maintenable.

Below a grade of 3, the number of exchanges between the developers and tests is multiplied by 3.

Test statique

Exemple d'analyse de code outillée

Summary | Maintenability | **Transferability** | Evolutivity | Robustness | Performance | Security

Metric grade	2.31
Name	Dead code (static)
Description	Respect of code coverage practices

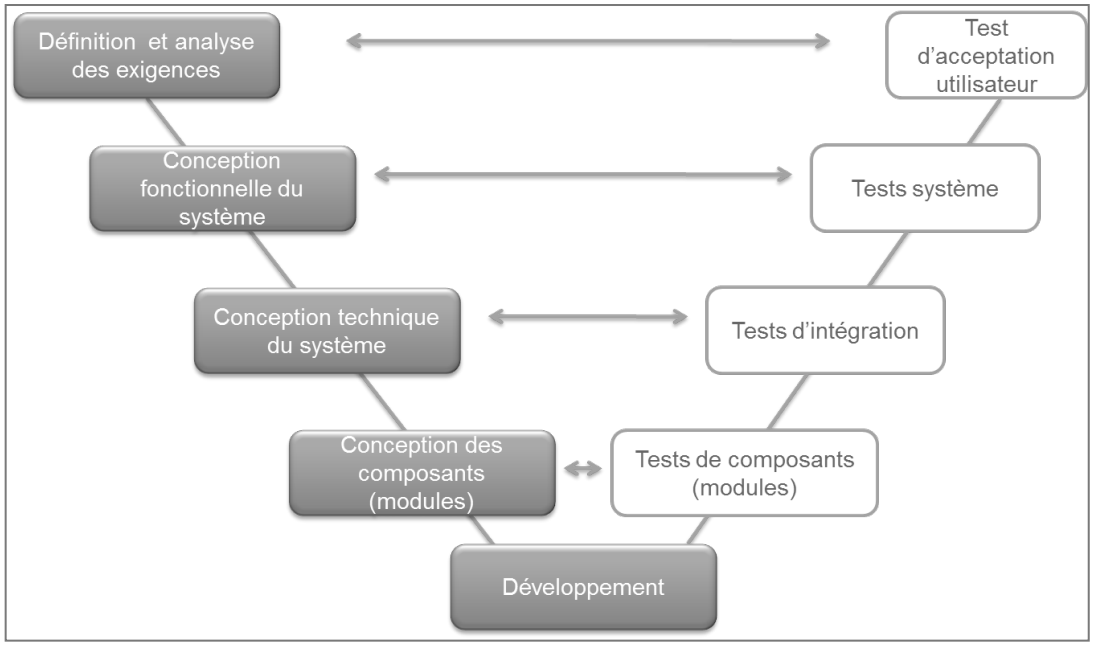
Child Metric Weight	Critical contribution	Child Metric Name	Child Metric Status	Child Metric Grade
6	No	Java: Avoid unreferenced Interfaces	Moderate Risk	3.94
4	No	JavaScript: Avoid unreferenced JavaScript Functions	Very High Risk	1
4	No	Java: Avoid unreferenced Classes	Very High Risk	1.89
4	No	Java: Avoid unreferenced Methods	High Risk	2.41
4	No	Java: Avoid unreferenced Fields	High Risk	2.7
4	No	PL/SQL: Avoid unreferenced Functions / Procedures	Very High Risk	1.31
1	No	PL/SQL: Avoid unreferenced Tables	High Risk	2.38
1	No	JSP: Avoid unreferenced JSPs	Very High Risk	1.22
1	No	PL/SQL: Avoid unreferenced views	Very High Risk	1

The dead code makes it harder to understand how the application realy works and makes its maintaining more expansive

Test dynamique - niveaux

Repose sur l'exécution du programme à tester

- 4 niveaux complémentaires
  - Test de composants (unitaire)
  - Test d'intégration des composants
  - Test du système global
  - Test d'acceptation (recette)



Test dynamique - techniques

Deux techniques :

- **Test structurel**
  Jeu de test sélectionné en s'appuyant sur une analyse du code source du logiciel (*test boite blanche / boite de verre*)

- **Test fonctionnel**
  Jeu de test sélectionné en s'appuyant sur les spécifications (*test boite noire*)

En résumé, les méthodes de test dynamique consistent à :

- Exécuter le programme sur un ensemble fini de données d'entrées
- Contrôler la correction des valeurs de sortie en fonction de ce qui est attendu

Test structurel (white box)

- Les données de test sont produites à partir d'une analyse du code source

Critères de test :
- tous les chemins,
- toutes les instructions,
- etc…

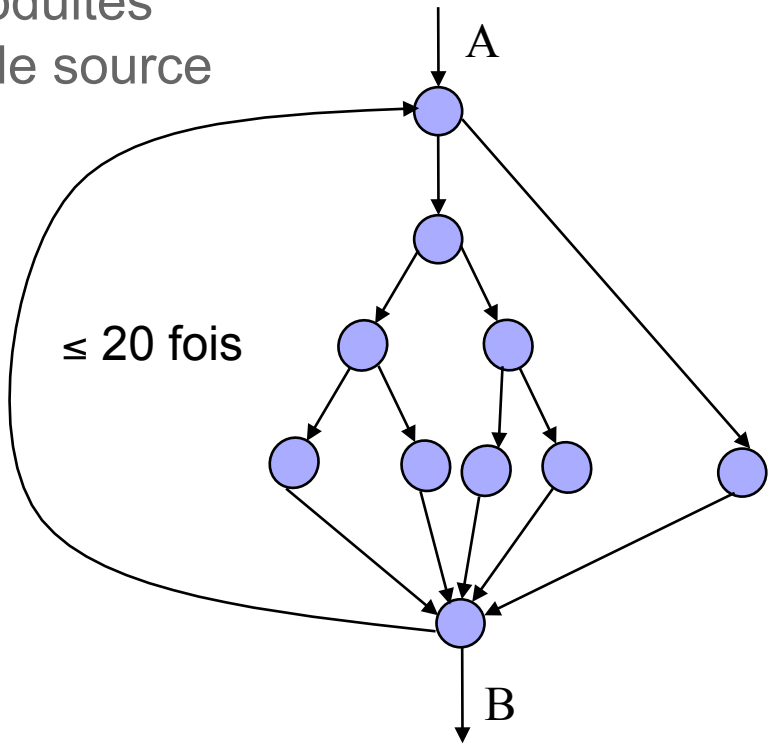
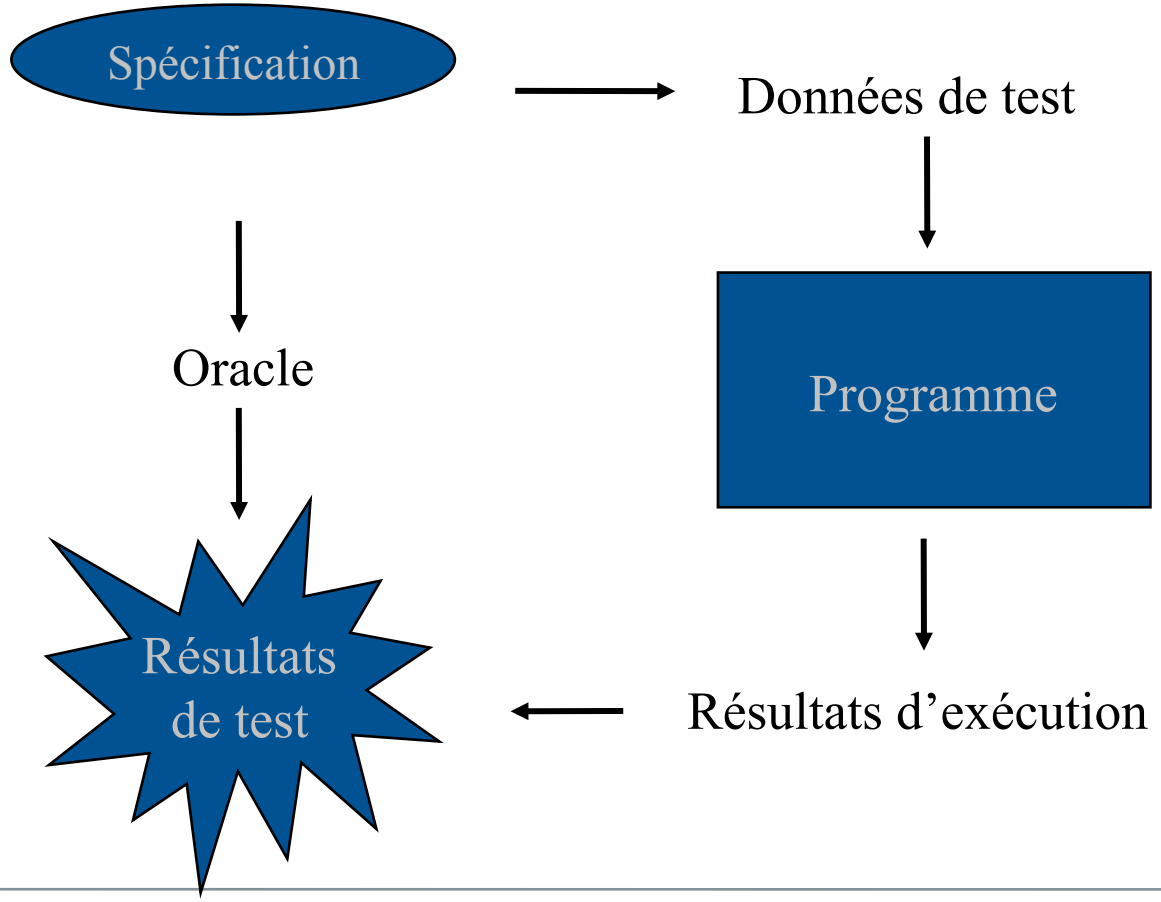


Fig. 1 : Flot de contrôle d'un petit programme

Test fonctionnel (black-box)

- Test de conformité par rapport à la spécification



Complémentarité (1) test fonctionnel / structurel

- Les 2 approches sont utilisées de façon complémentaire
- Exemple : soit le programme suivant, censé calculer la somme de 2 entiers :

```
function sum (x,y : integer) : integer;
        if (x = 600) and (y = 500)
        then
          sum := x-y
        else
          sum := x+y;
        end
```

- Une approche fonctionnelle détectera difficilement le défaut, alors qu'une approche par analyse de code pourra produire la donnée de test :

x = 600, y = 500.

Complémentarité (2) test fonctionnel / structurel

- En examinant ce qui a été réalisé, on ne prend pas forcément en compte ce qui aurait du être fait :
  - Les approches structurelles détectent plus facilement les erreurs commises dans le programme
  - Les approches fonctionnelles détectent plus facilement les erreurs d'omission et de spécification
- Une difficulté du test structurel consiste dans la définition de l'oracle de test.

Test de logiciels – auto-évaluation
L'exemple du triangle

- Soit la spécification suivante :

  Un programme prend en entrée trois entiers. Ces trois entiers sont interprétés comme représentant les longueurs des cotés d'un triangle. Le programme rend un résultat précisant s'il s'agit d'un triangle scalène, isocèle ou équilatéral (ou une erreur si les données ne correspondent pas aux longueurs d'un triangle).

  G. Myers (The Art of Software testing - 1979)

- Donnez un ensemble de cas de test que vous pensez adéquat pour tester pour ce programme…

Test de logiciels – auto-évaluation
L'exemple du triangle

- Avez-vous un cas de test pour :

Test de logiciels – auto-évaluation
L'exemple du triangle

- Chacun de ces cas correspond à un défaut constaté dans des implantations de cet exemple du triangle

- La moyenne des résultats obtenus par un ensemble de développeurs expérimentés est de 7.8 sur 14.

=> Le test est une activité complexe, a fortiori sur de grandes applications

Test dynamique – 4 activités

Sélection d'un jeu de tests : choisir un sous-ensemble des entrées possibles du logiciel

Soumission du jeu de tests

Dépouillement des résultats : consiste à décider du succès ou de l'échec du jeu de test (*verdict*): **Fail**, **Pass**, **Inconclusive**

Évaluation de la qualité et de la pertinence des tests effectués (déterminant pour la décision d'arrêt de la phase de test)

Types de tests (1)
(renseignent la nature du test mené)

Tests fonctionnels

Valide les résultats rendus par les services

Tests non-fonctionnels

Valide la manière dont les services sont rendus

Tests nominaux / de bon fonctionnement

Vérifie que le résultat calculé est le résultat attendu, en entrant des données valides au programme (*test-to-pass*)

Tests de robustesse

Vérifie que le programme réagit correctement à une utilisation non conforme, en entrant des données invalides (*test-to-fail*)

Types de tests (2) (renseignent la nature du test mené)

Test de performance

- Load testing (test avec montée en charge)
- Stress testing (soumis à des demandes de ressources anormales)

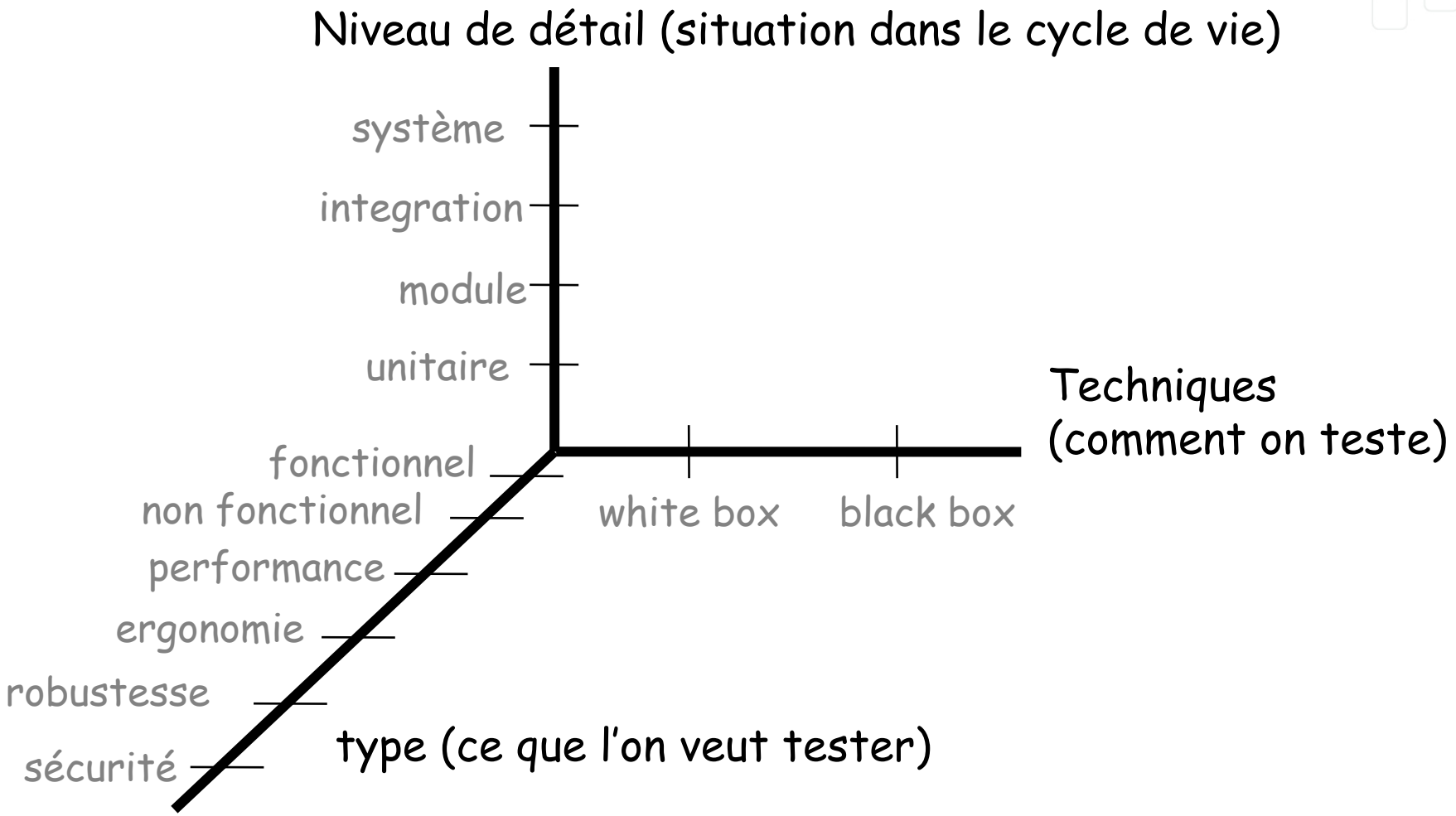
Test de non-régression

Vérifie que les corrections ou évolutions dans le code n'ont pas créé d'anomalies nouvelles

Test de confirmation

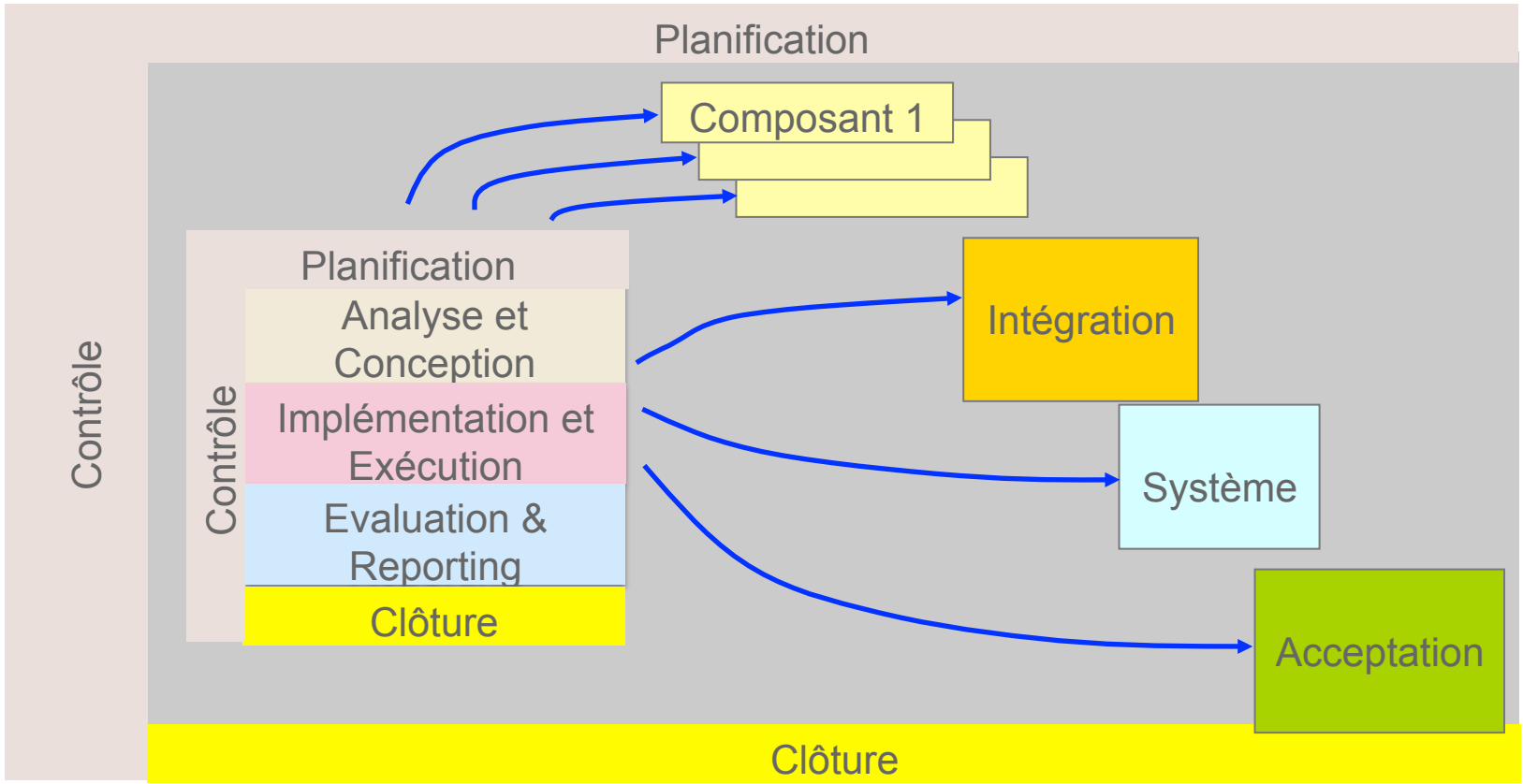
Valide la correction d'un défaut

Catégories de tests

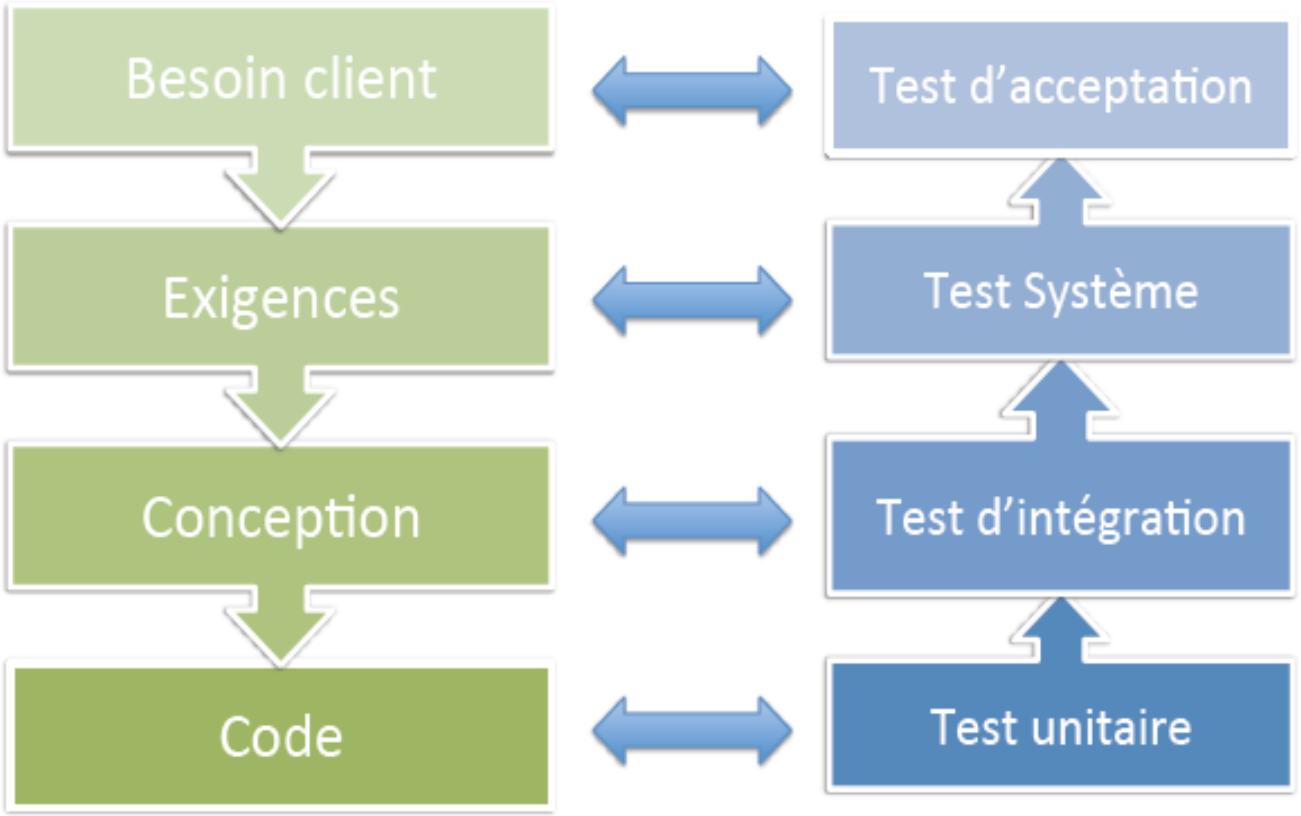


Processus de test et stratégie de test

Le Test est un ensemble d'activités à dérouler à différents niveaux



Développement et niveaux de tests



Niveaux de tests (renseignent l'objet du test)

Tests (structurels) unitaires

Test de procédures, de modules, de composants

(coût : 50% du coût total du développement initial correspondant

Tests d'intégration

Test de bon comportement lors de la composition des procédures, modules ou composants

(coût d'un bug dans cette phase : 10 fois celui d'un bug unitaire)

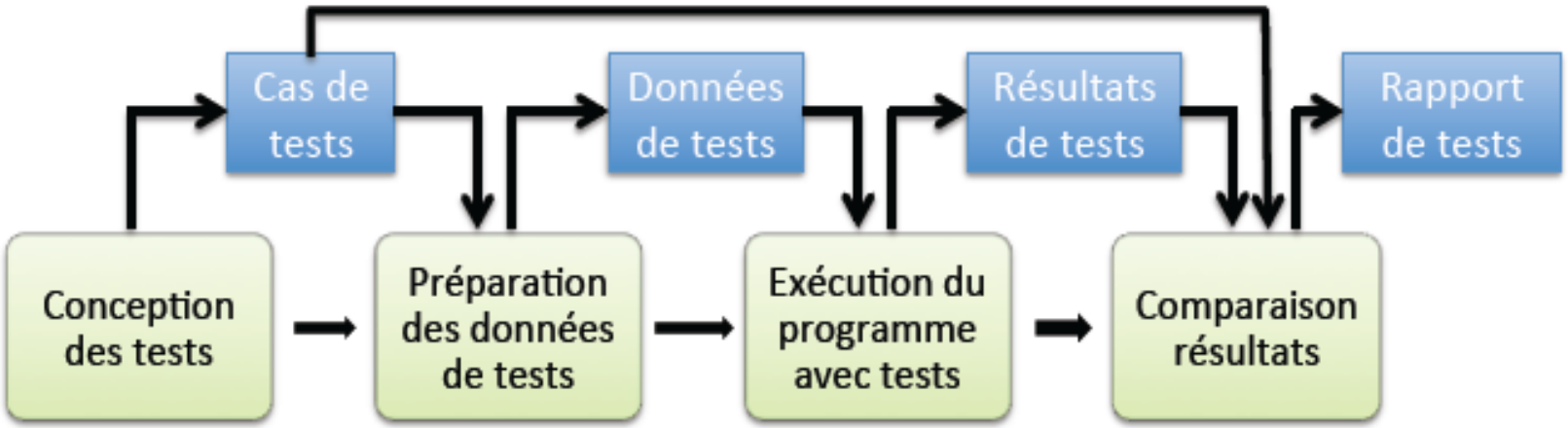
Tests système / de recette

Validation de l'adéquation aux spécifications

(coût d'un bug dans cette phase : 100 fois celui d'un bug unitaire)

Processus de test et stratégie de test

Test et cycle de vie du processus de test



Difficultés du test

- Le test exhaustif est en général impossible à réaliser
  - En test structurel, le parcours du graphe de flot de contrôle conduit à une forte explosion combinatoire

    Exemple : le nombre de chemin logique dans le graphe de la figure 1 est supérieur à $10^{14} \approx 5^{20} + 5^{19} + \ldots + 5^{1}$

  - En test fonctionnel, l'ensemble des données d'entrée est en général infini ou très grande taille

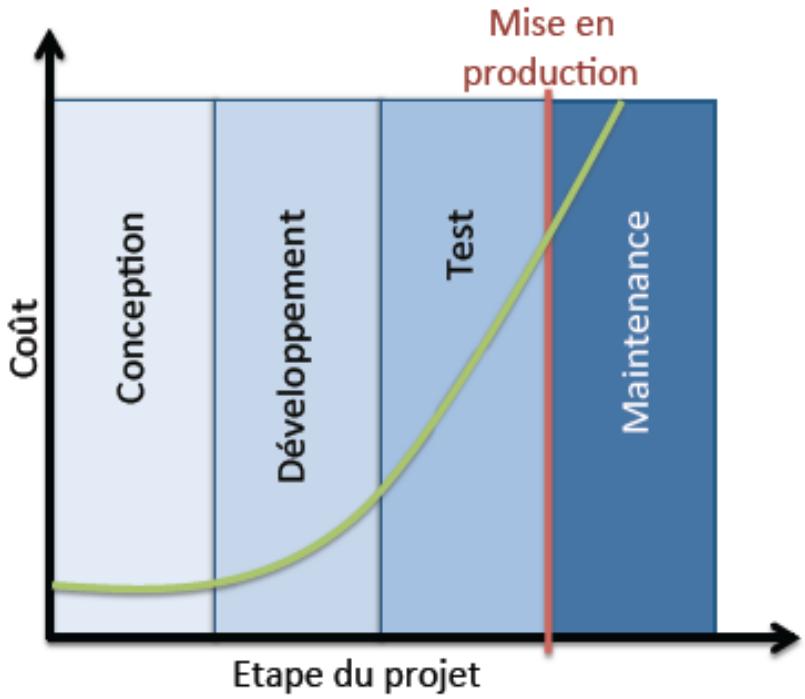
    Exemple : un logiciel avec 5 entrées analogiques sur 8 bits admet 2^{40} *valeurs différentes en entrée*

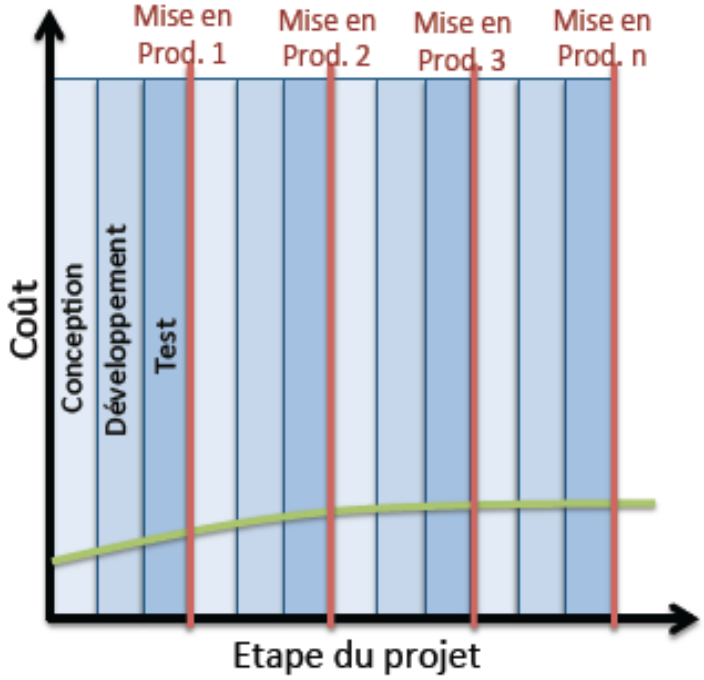
=> le test est une méthode de validation partielle de logiciels

=> la qualité du test dépend de la pertinence du choix des données de test

Coût d'un bug

Le but est de les détecter le plus tôt possible (Stratégie)





Stratégie de test

En début de projet, définition d'un Plan de Test et de Validation (PTV)

- Objectifs de test
- Techniques de test
- Phases et planning

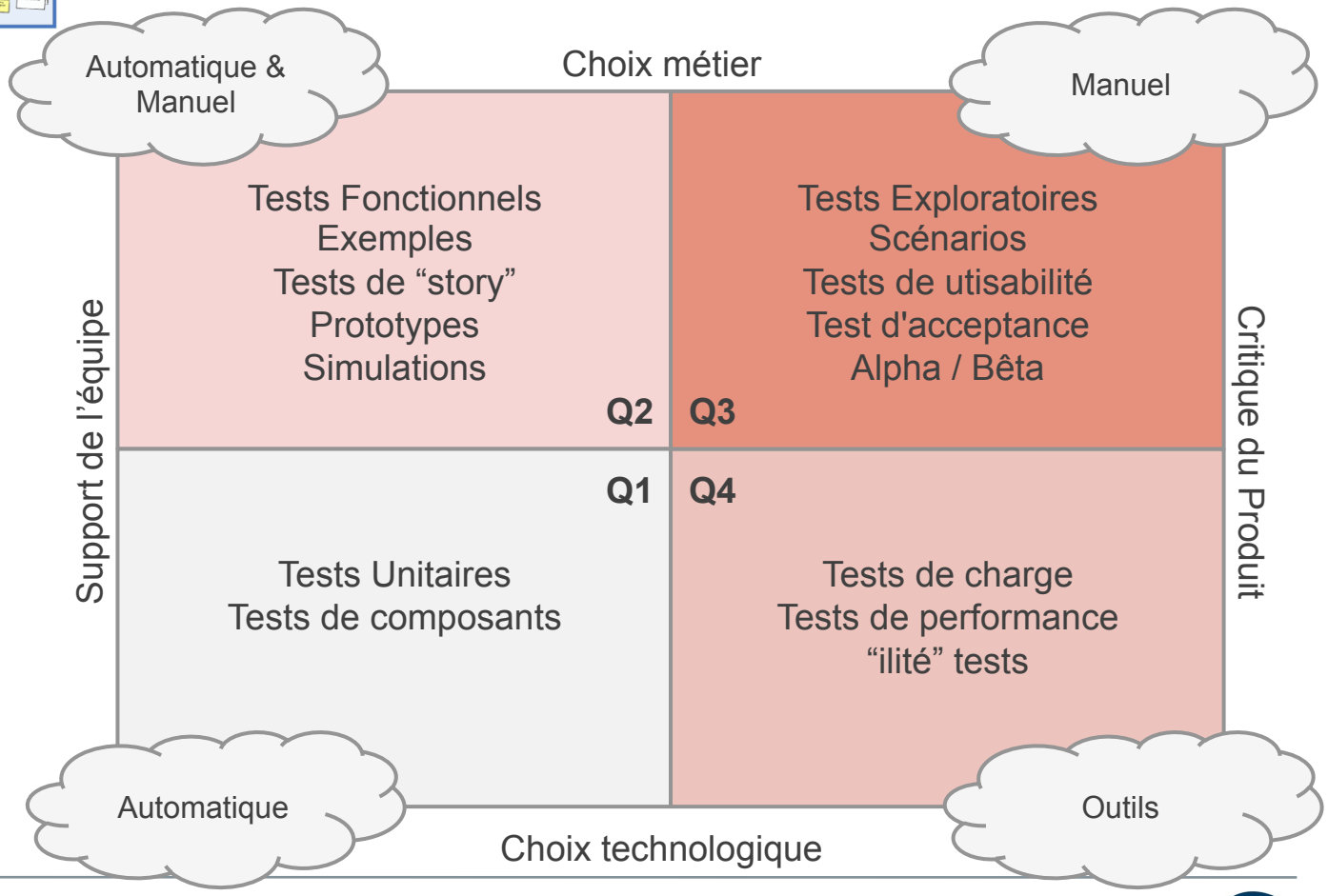
Acteurs du test

- Deux situations :
  - Je teste un programme que j'ai écrit
  - Je teste un programme que quelqu'un d'autre a écrit

- Trois questions :
  - Comment choisir la technique de test ?
    => boite blanche ou boite noire ?
  - Comment obtenir le résultat attendu ?
    => problème de l'oracle du test
  - Comment savoir quand arrêter la phase de test ?
    => problème de l'arrêt

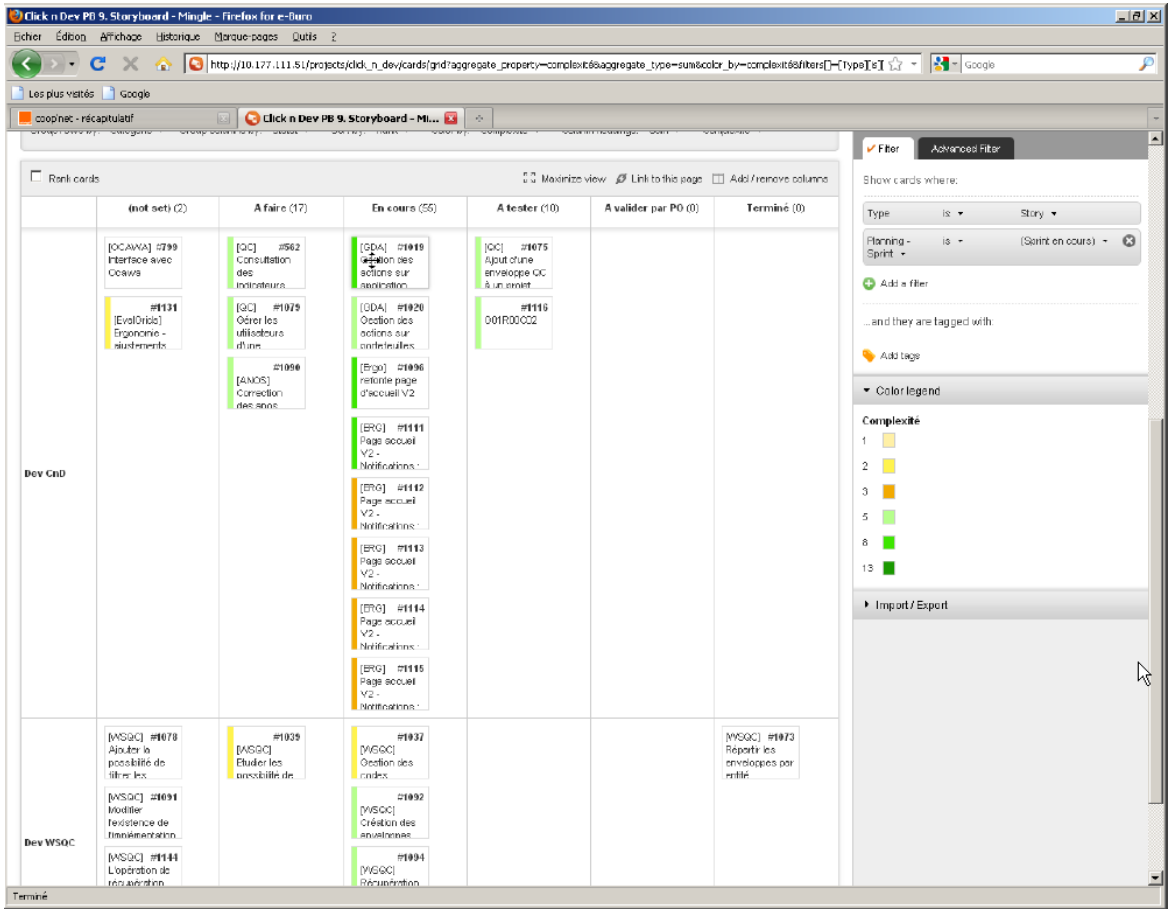
Outils et Méthodes intervenants à différentes étapes

mingle JIRA Agile ScrumDo hp Agile Manager



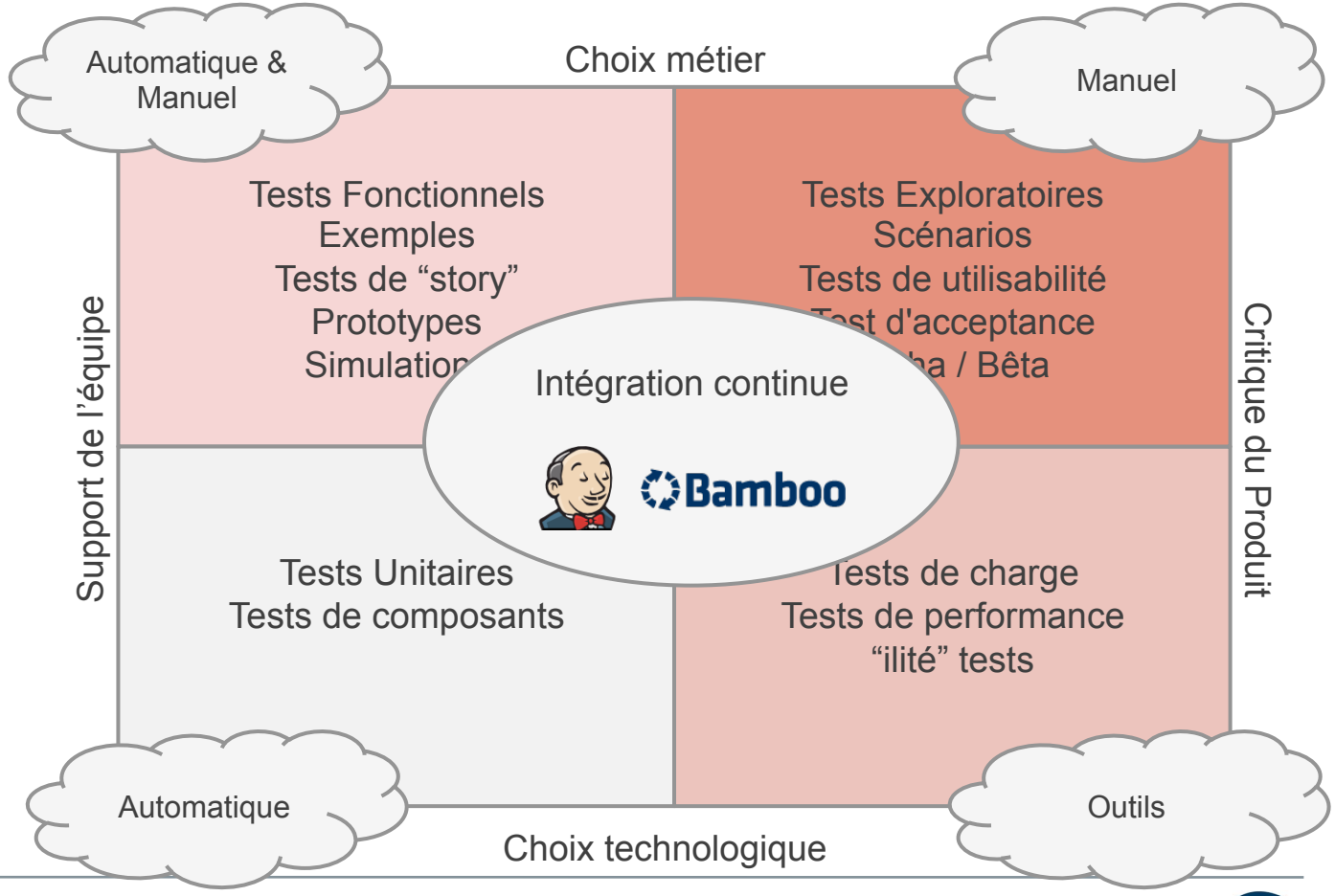
Outils et Méthodes intervenants à différentes étapes

Exemple Mingle pour les projets Agile



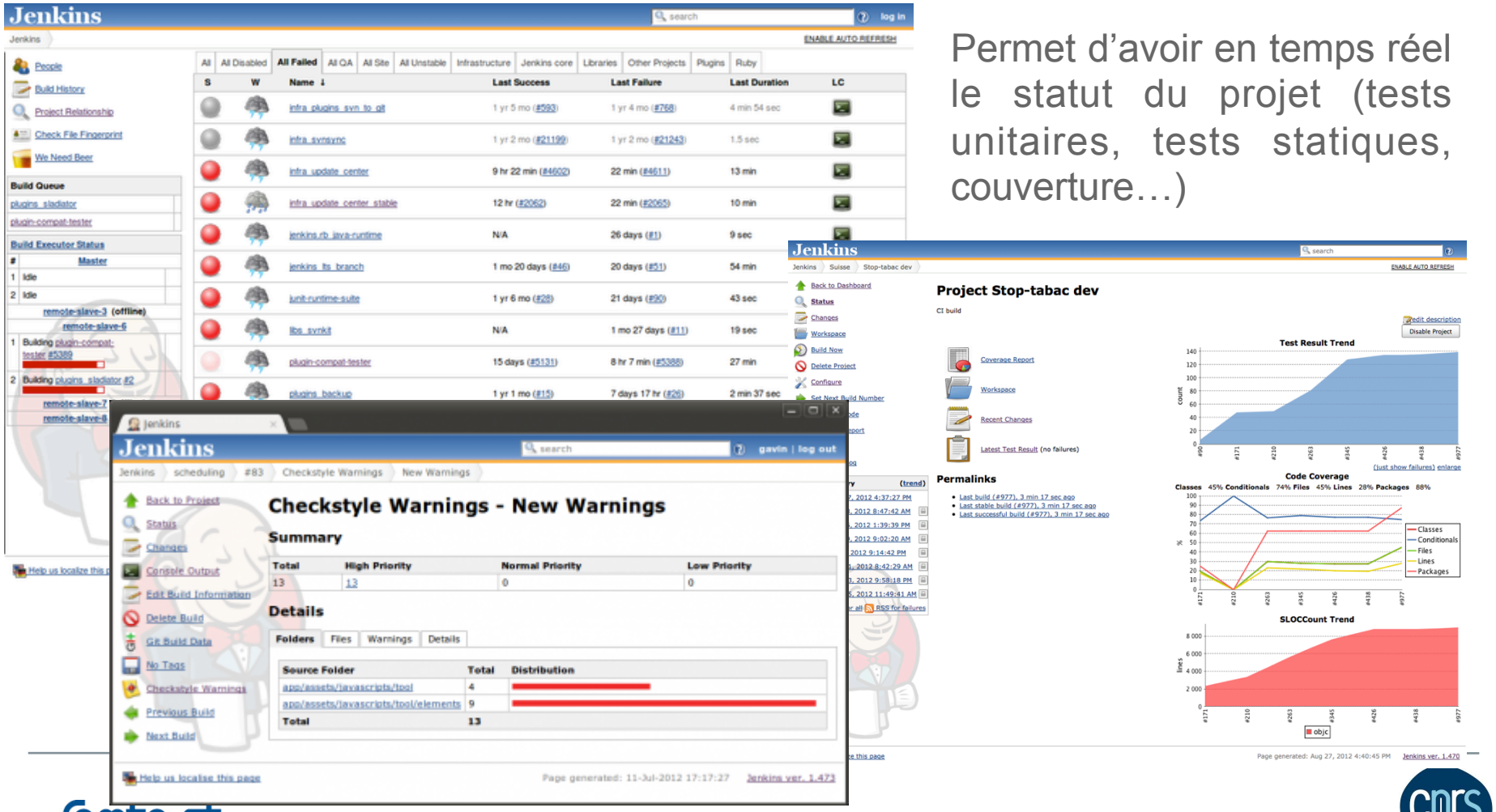
Outils et Méthodes intervenants à différentes étapes

Build et Distribution: Utile pour tout le quadrant de test



Outils et Méthodes intervenants à différentes étapes

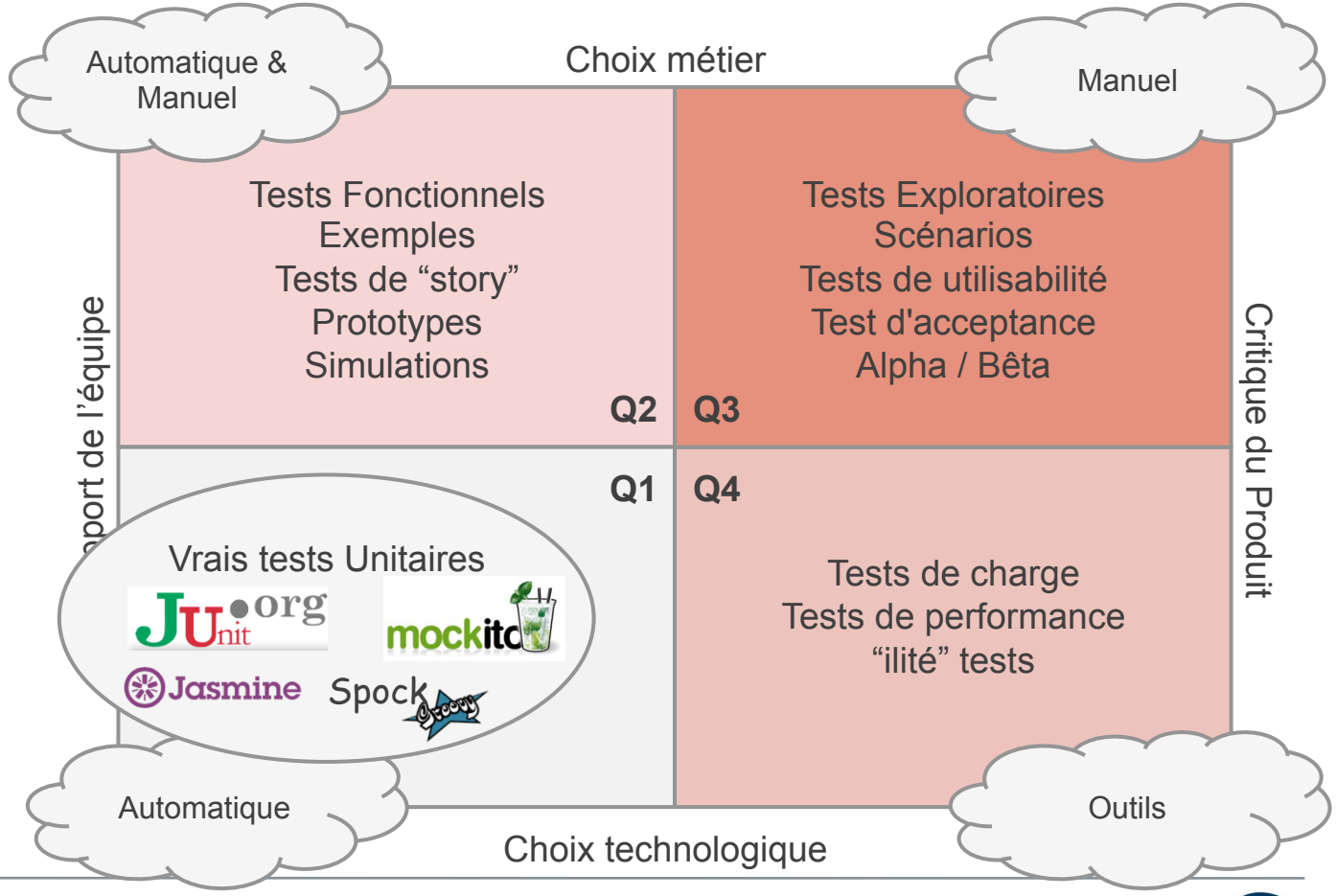
Intégration continue : Tableau de contrôle



Permet d'avoir en temps réel le statut du projet (tests unitaires, tests statiques, couverture…)

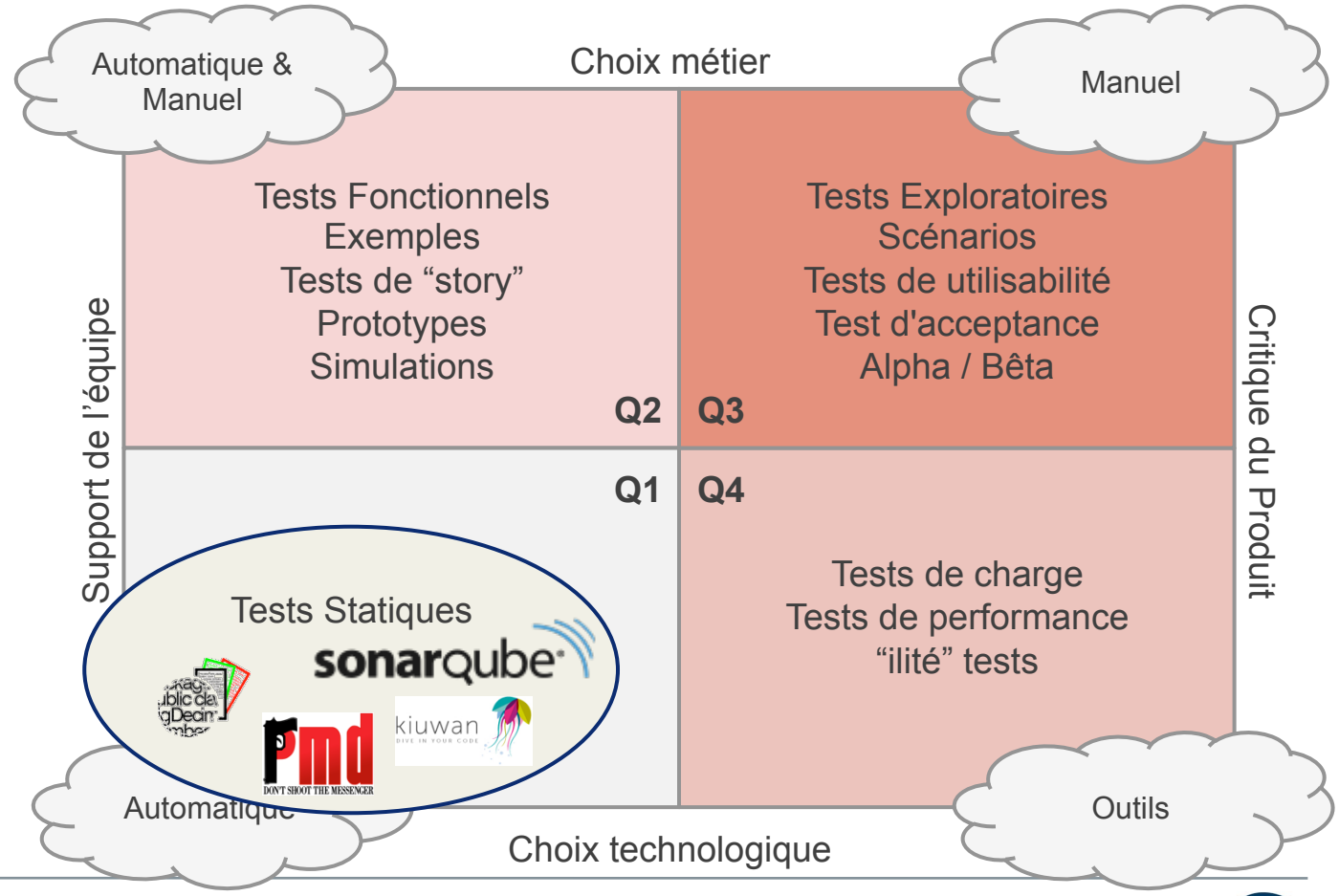
Outils et Méthodes intervenants à différentes étapes

Tests unitaires



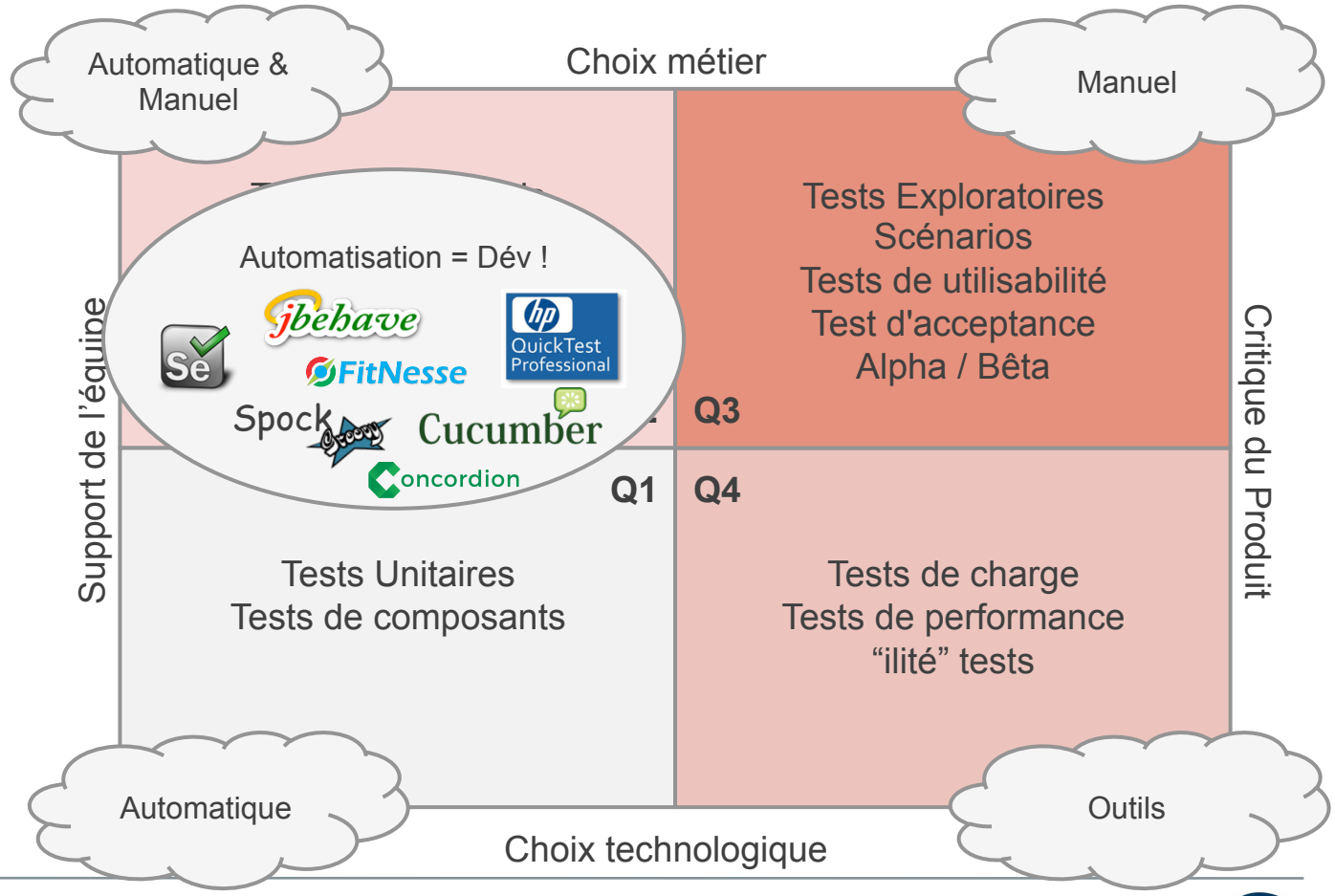
Outils et Méthodes intervenants à différentes étapes

Tests statiques !



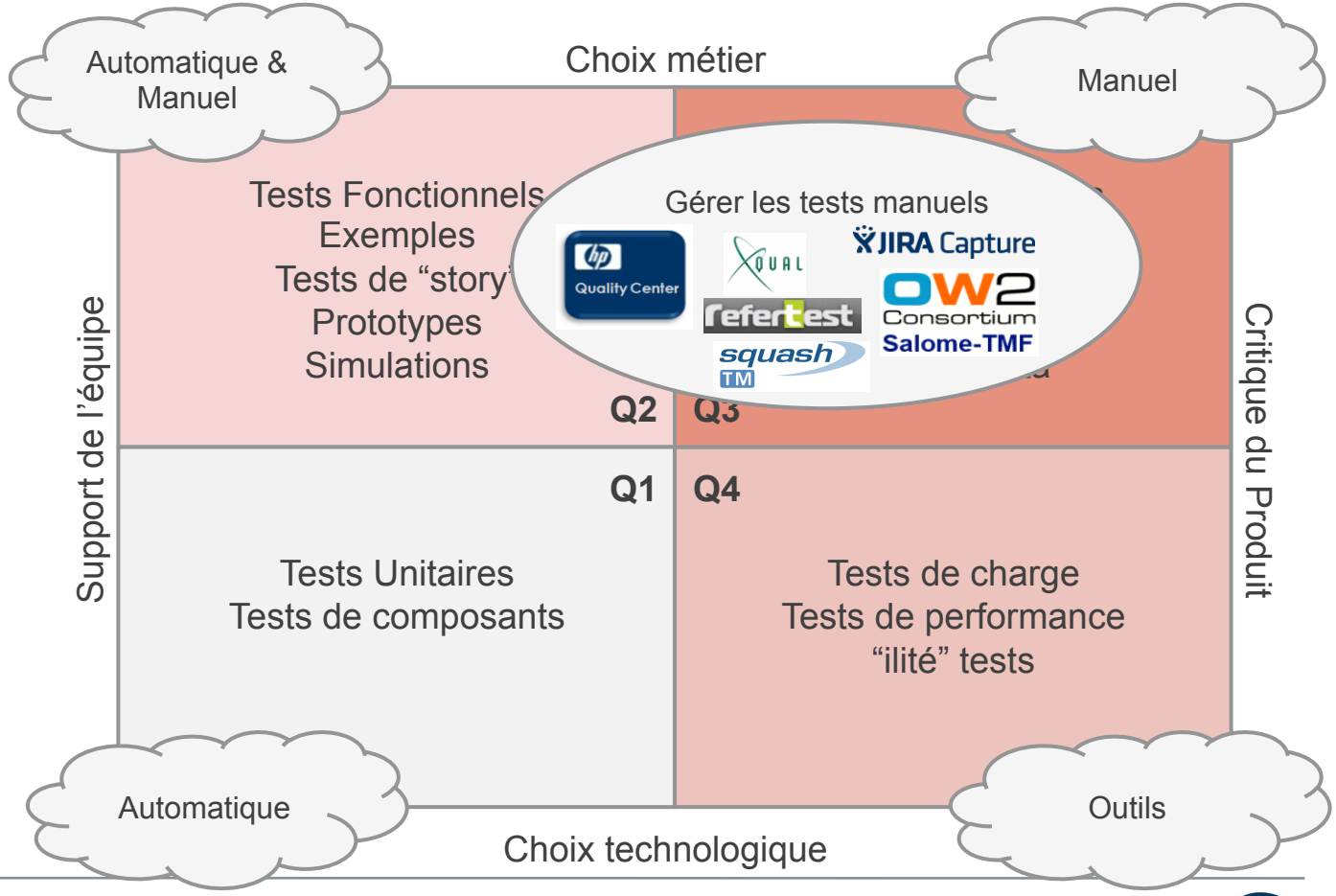
Outils et Méthodes intervenants à différentes étapes

Automatisation des tests



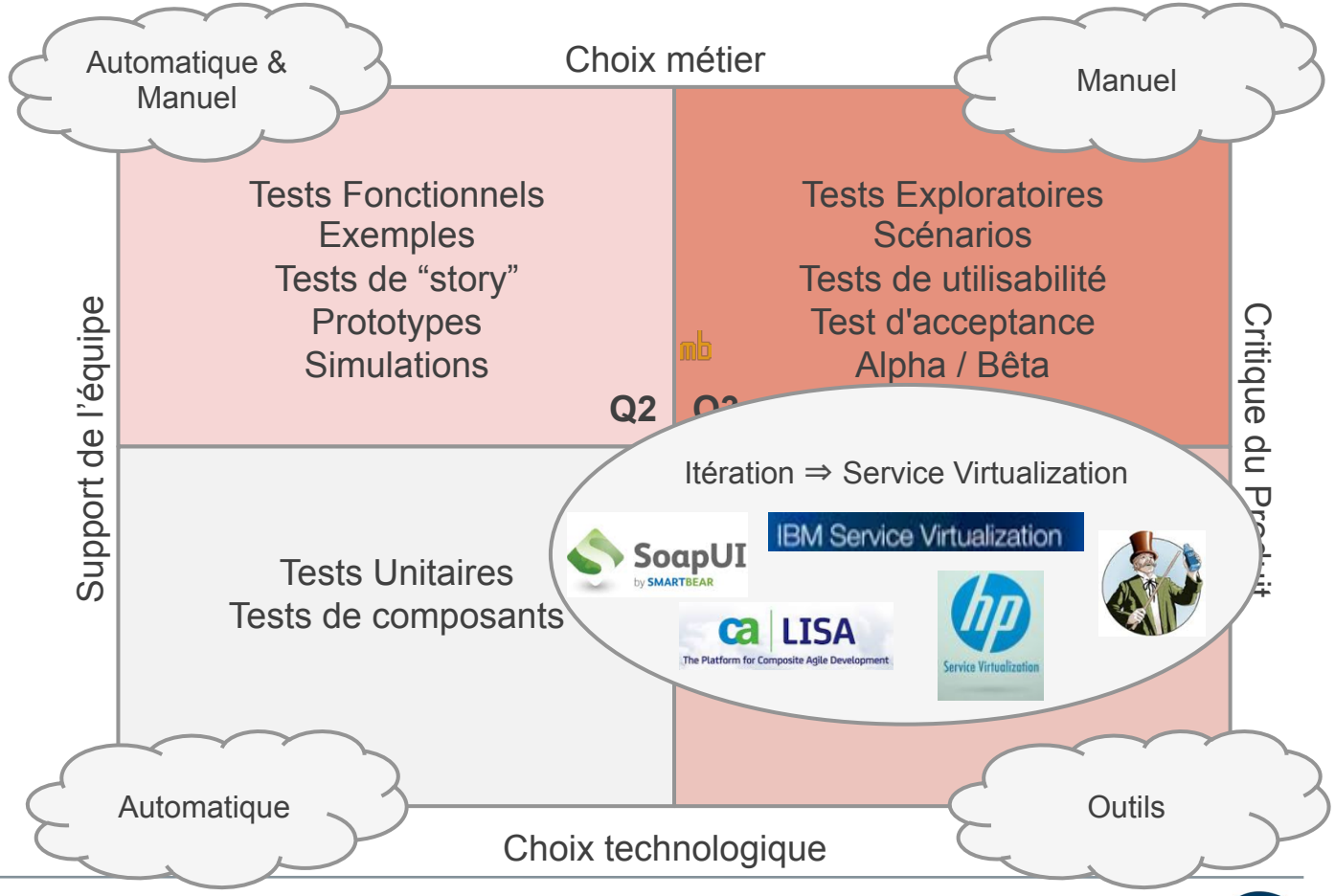
Outils et Méthodes intervenants à différentes étapes

Gestion des tests y compris exploratoires !



Outils et Méthodes intervenants à différentes étapes

Virtualisation et Cloud : un nouveau défit !



Outils et Méthodes intervenants à différentes étapes

Couverture : un métrique important

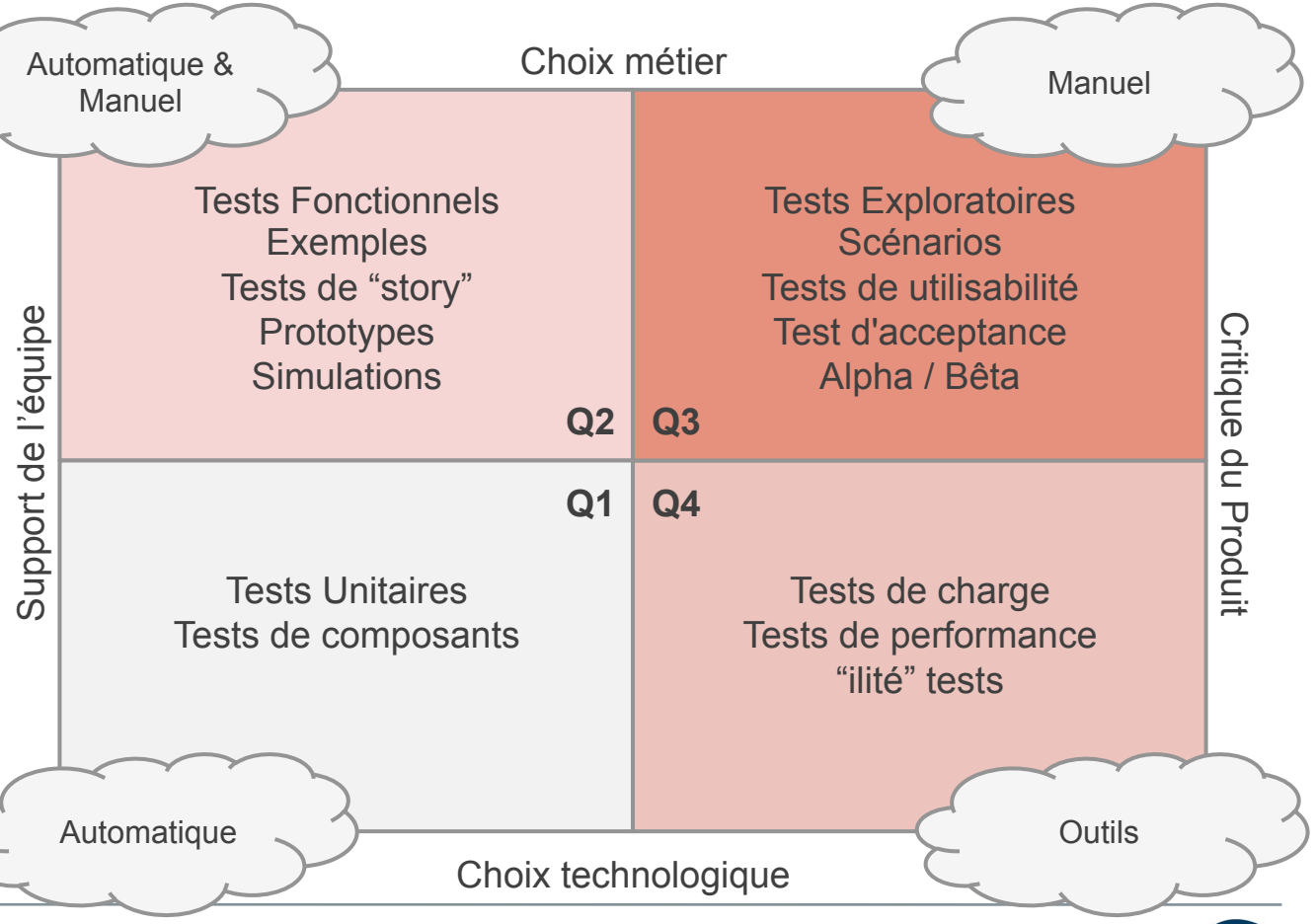
Analyse de la couverture

kalistick
INNOVATION FOR AGILE QA

Cobertura

Clover

Powered by JACOCO Java Code Coverage



Outils et Méthodes intervenants à différentes étapes

BDD avec JBehave

1. Write story

Plain text

```
Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status should be OFF
When stock is traded at 16.0
Then the alert status should be ON
```

2. Map steps to Java

POJO

```
public class TraderSteps {
    private TradingService service;   // Injected
    private Stock stock; // Created

    @Given("a stock and a threshold of $threshold")
    public void aStock(double threshold) {
        stock = service.newStock("STK", threshold);
    }
    @When("the stock is traded at price $price")
    public void theStockIsTraded(double price) {
        stock.tradeAt(price);
    }
    @Then("the alert status is $status")
    public void theAlertStatusIs(String status) {
        assertThat(stock.getStatus().name(), equalTo(status));
    }
}
```

Outils et Méthodes intervenants à différentes étapes

Tester avec FitNesse

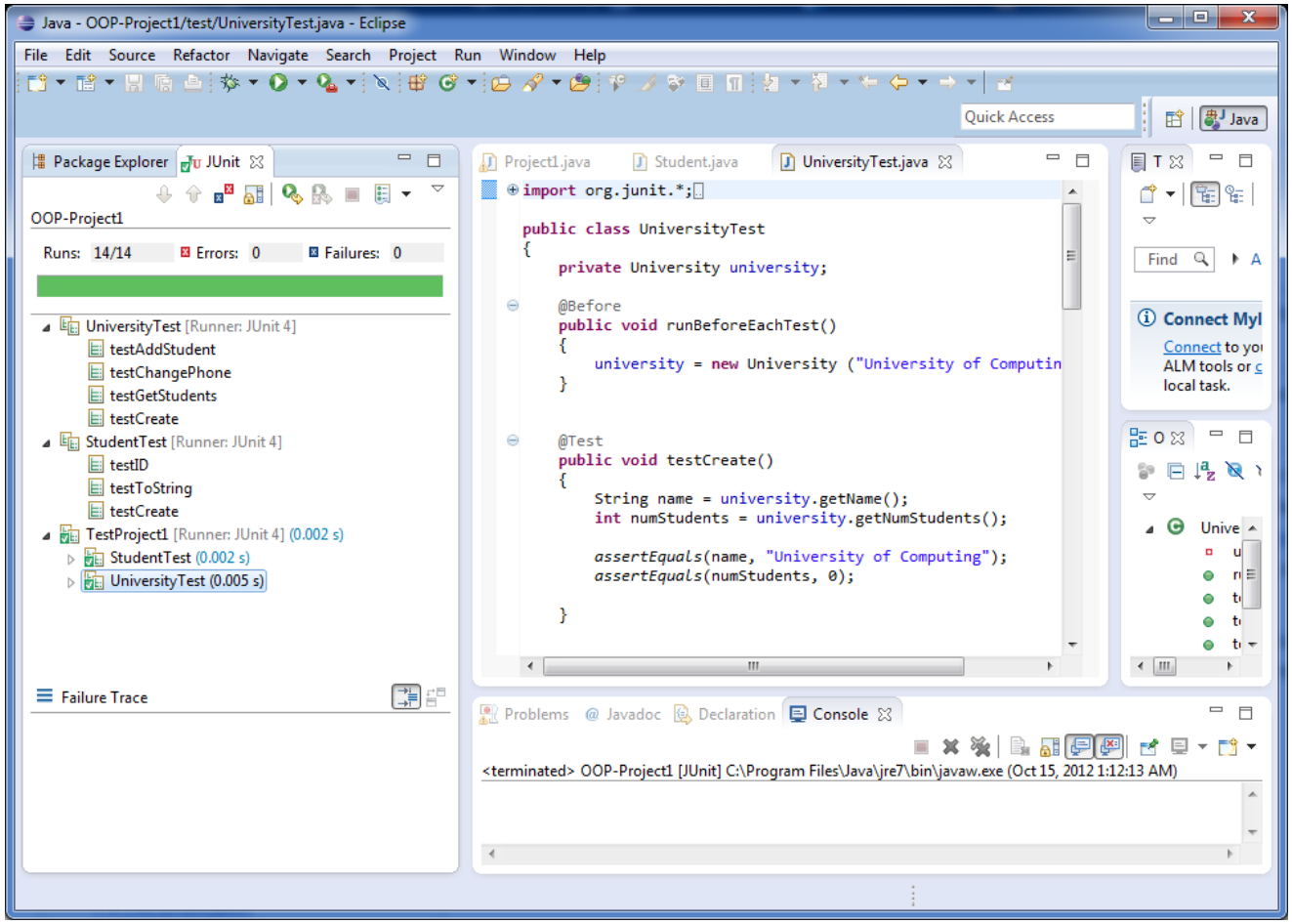
- FitNesse utilise un wiki pour définir les cas de tests : on écrit des tables de décision. FitNesse est bien adapté à des testeurs fonctionnels
- Le développeur doit écrire le “fixture” code pour appeler les fonctionnalités concernées

eg.triviaGameExample.fitnesseFixtures.AddRemovePlayerFixture		
playerName	addPlayer?	countPlayers?
Al	true	1
Bertha	true	2

```
import fit.ColumnFixture;

public class AddRemovePlayerFixture {
  private String playerName;
  private Game theGame;

  public void setPlayerName(String playerName) {
    this.playerName = playerName;
  }

  public boolean addPlayer() {
    theGame = StaticGame.theGame;
    Player thePlayer = theGame.addPlayer(playerName);
    return theGame.playerIsPlaying(thePlayer);
  }

  public int countPlayers() {
    return theGame.getNumberOfPlayers();
  }
}
```

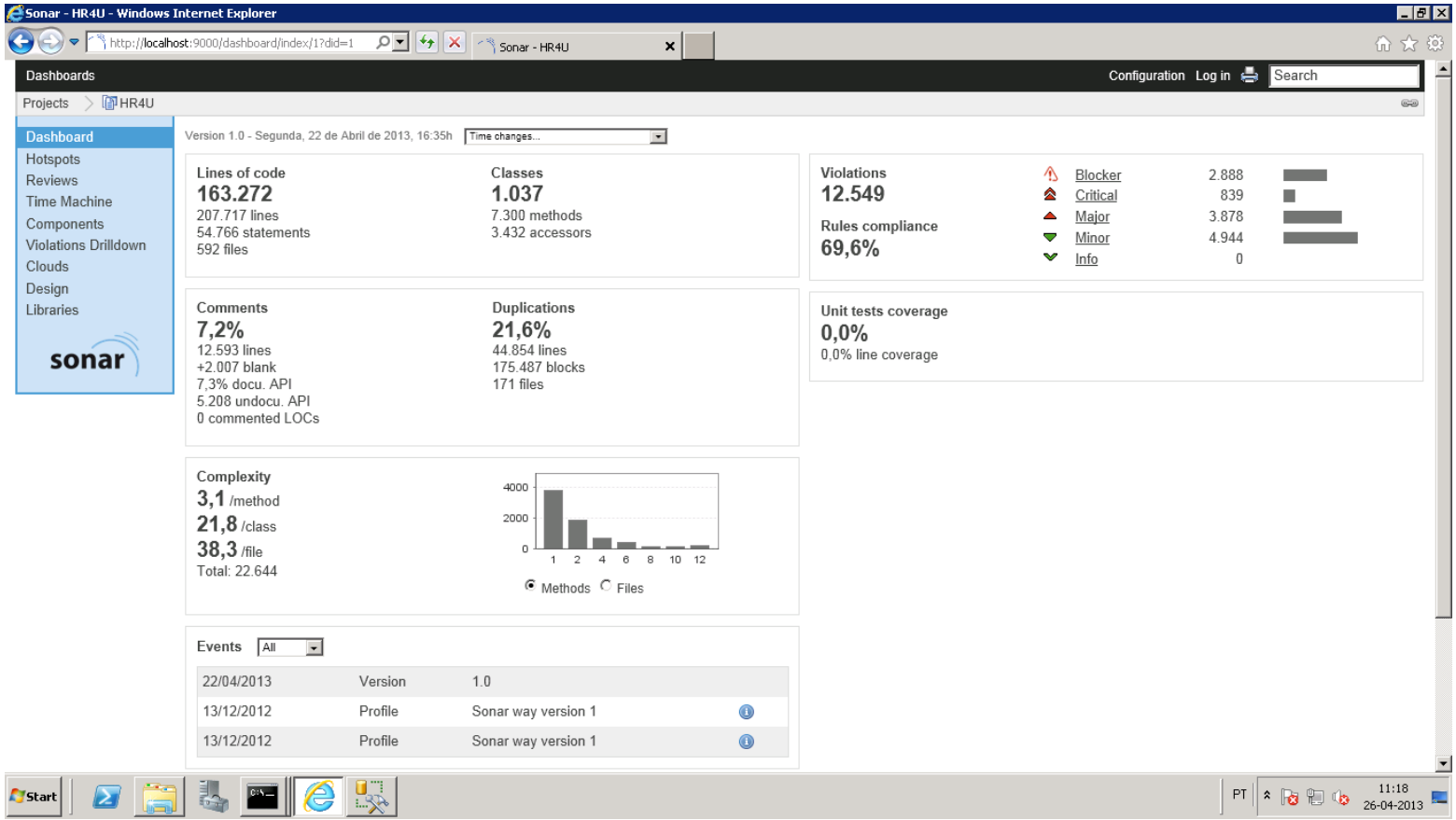
Outils et Méthodes intervenants à différentes étapes

Exemple: tests unitaires avec Junit depuis Eclipse



Outils et Méthodes intervenants à différentes étapes

Exemple: qualité du code mesurée avec Sonar



2. Automatisation des tests unitaires

Organisation et bonnes pratiques pour les tests unitaires.
Critères d'automatisation.
Tests unitaires : Tests Driven Development.
Mesure de la couverture de code : couverture d'instructions et branches.
Analyse statique de code : analyse outillée du code source hors exécution
Automatisation avec un fichier de configuration.
Analyse dynamique de code : couverture des instructions, des branches, des prédicats...
Organisation des tests unitaires, pair programming, pair testing.
Utilisation des Frameworks : gestion des scripts de tests

Test unitaire

Quoi ?

- Validation du fonctionnement d'une portion d'un programme

Quand ?

- Dans un cycle en V : après la phase de développement
- Dans un cycle agile : intégré au développement (TDD)

Qui ?

- Le développeur

Comment ?

- Package disponibles dans de nombreux langages
- Java : Junit, TestNG
- PHP : PHPUnit, SimpleTest, atoum

Test structurel

- Le test structurel s'appuie sur l'analyse du code source de l'application pour établir les tests en fonction de critères de couverture

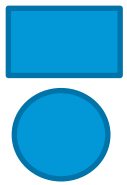
⇨ Basés sur le graphe de contrôle (toutes les instructions, toutes les branches, tous les chemins, …)

⇨ Basés sur la couverture du flot de données (toutes les définitions de variables, toutes les utilisations, …)

Graphe de contrôle

Définition

- Permet de représenter n'importe quel algorithme
- Les nœuds représentent des blocs d'instructions



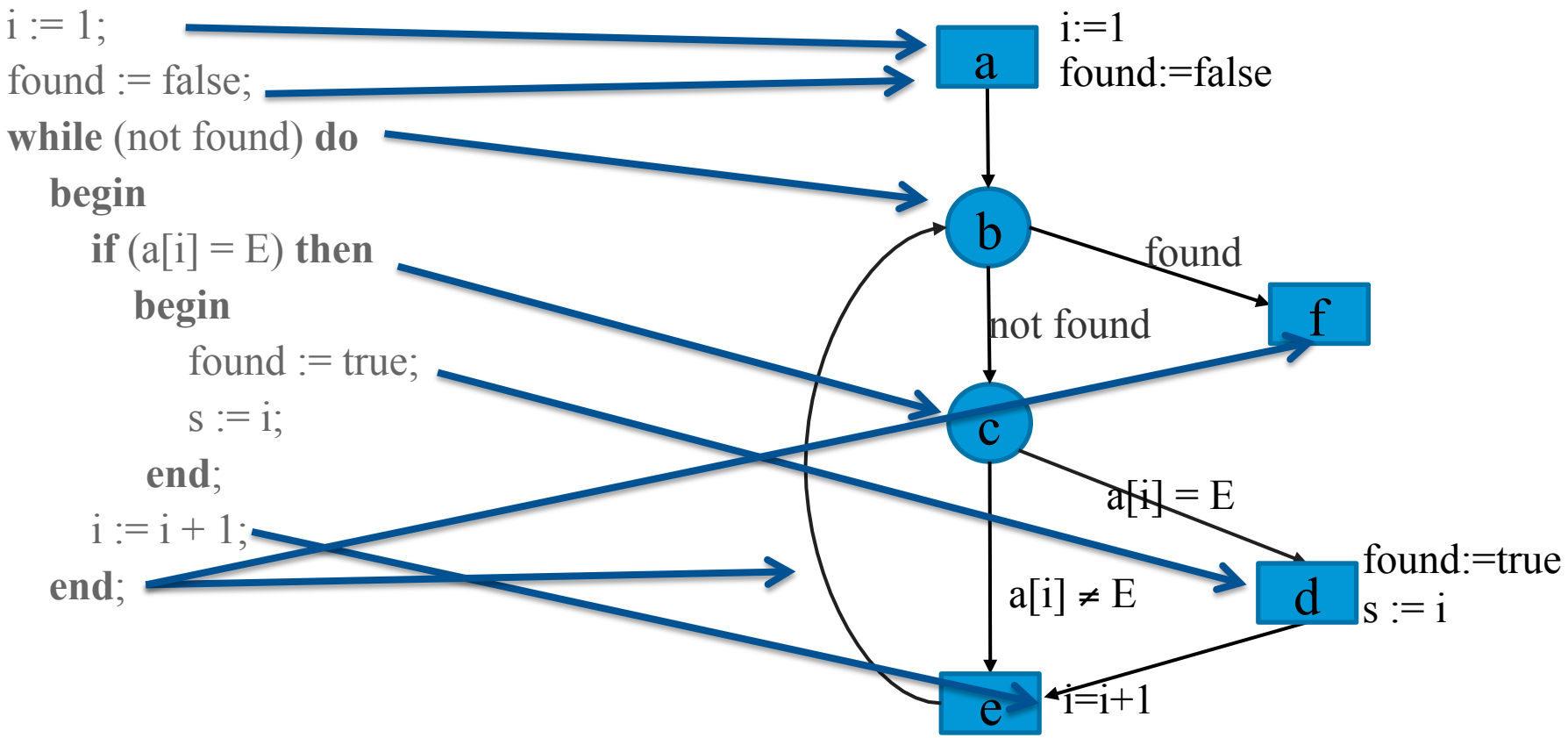
instruction ou suite d'instructions insécables

instruction de contrôle

- Les arcs représentent la possibilité de transfert de l'exécution d'un nœud à un autre
- Une seule entrée (nœud à partir duquel on peut visiter tous les autres) et une seule sortie

Graphe de contrôle

Production



Graphe de contrôle

Exercice

Produire le graphe de contrôle de la fonction :

```
public static Coordinates nextForwardPosition(Coordinates position, Direction direction) {
        if (direction == NORTH)
            return new Coordinates(position.getX(), position.getY() - 1);
        if (direction == SOUTH)
            return new Coordinates(position.getX(), position.getY() + 1);
        if (direction == EAST)
            return new Coordinates(position.getX() + 1, position.getY());
        return new Coordinates(position.getX() - 1, position.getY());
}
```

Graphe de contrôle

Exercice - correction

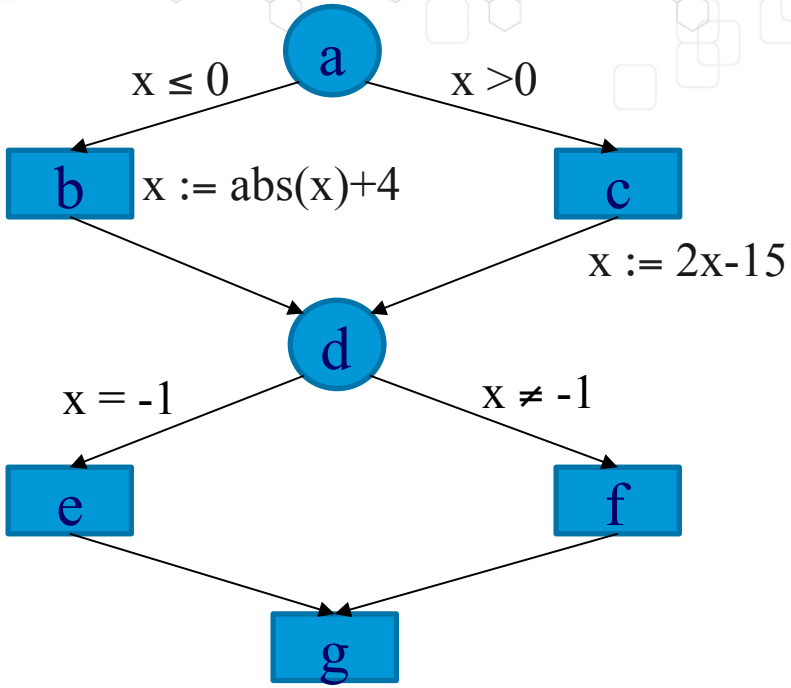
Chemins de contrôles

Définition

- Chemin dans le graphe de contrôle
- Débute au nœud d'entrée du graphe
- Termine au nœud de sortie du graphe

- Peut être activable ou non

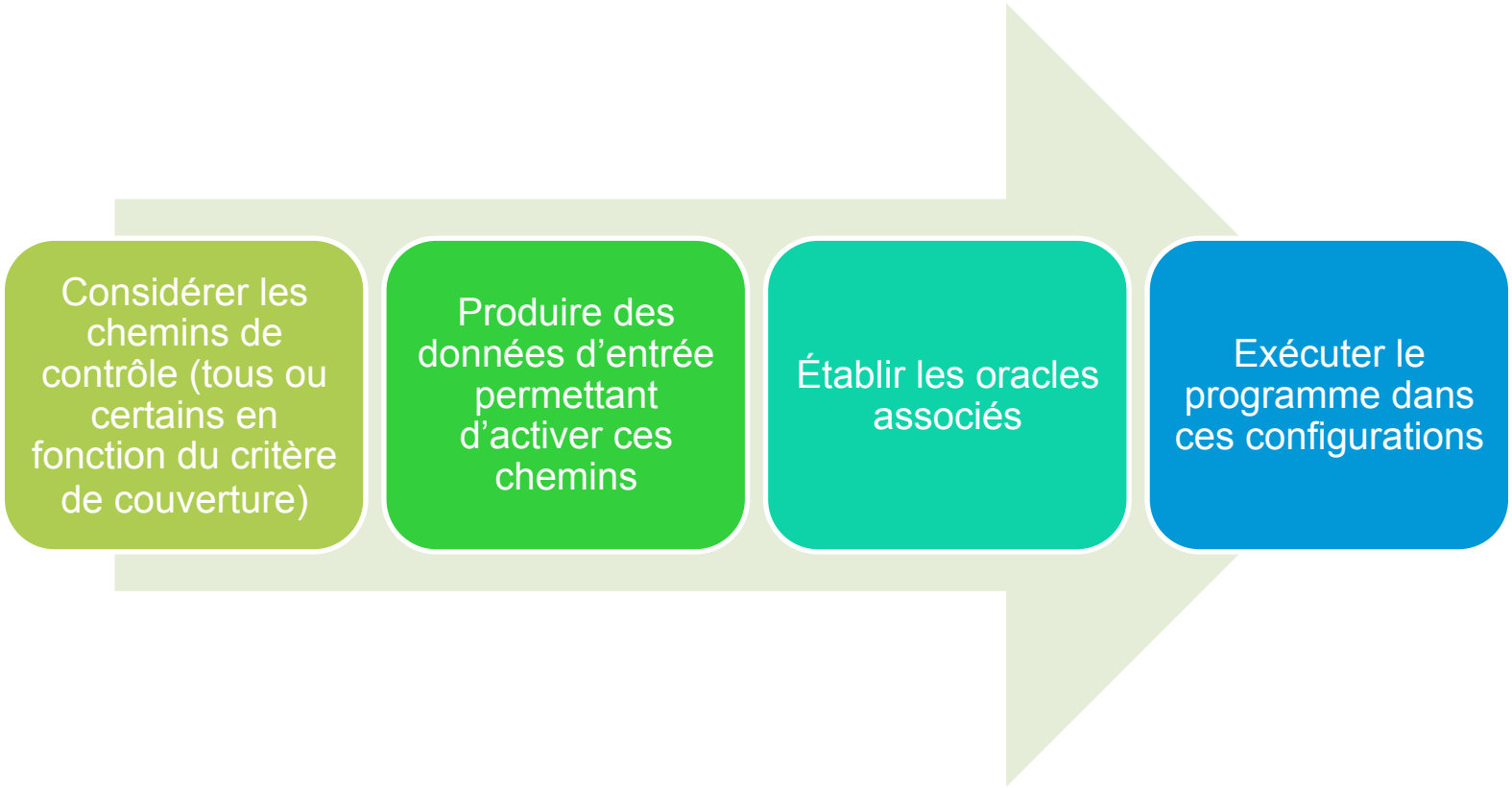
abdeg : un chemin de contrôle
bdfg : non



- 4 chemins de contrôles
- abdeg
- abdfg
- acdeg
- acdfg

Couverture structurelle

Couverture du graphe de flot de contrôle



Critères de couverture structurelle

Basés sur le graphe de contrôle

- Couverture de *tous-les-nœuds*,
- Couverture de *tous-les-arcs*,
- Couverture des *chemins limites et intérieurs*
- Couverture de *tous les i-chemins*,
- Couverture de *tous les chemins indépendants*
- Couverture de *tous les chemins*

Couverture de tous-les-nœuds

Couverture de toutes les instructions

- Chaque nœud (chaque bloc d'instructions) est atteint par au moins l'un des chemins parmi les chemins qui constituent le jeu de test

- Lorsqu'un jeu de test permet de couvrir tous les nœuds du graphe, on dit qu'il satisfait TER=1 ou TER1 (Test Effectiveness Ratio 1)

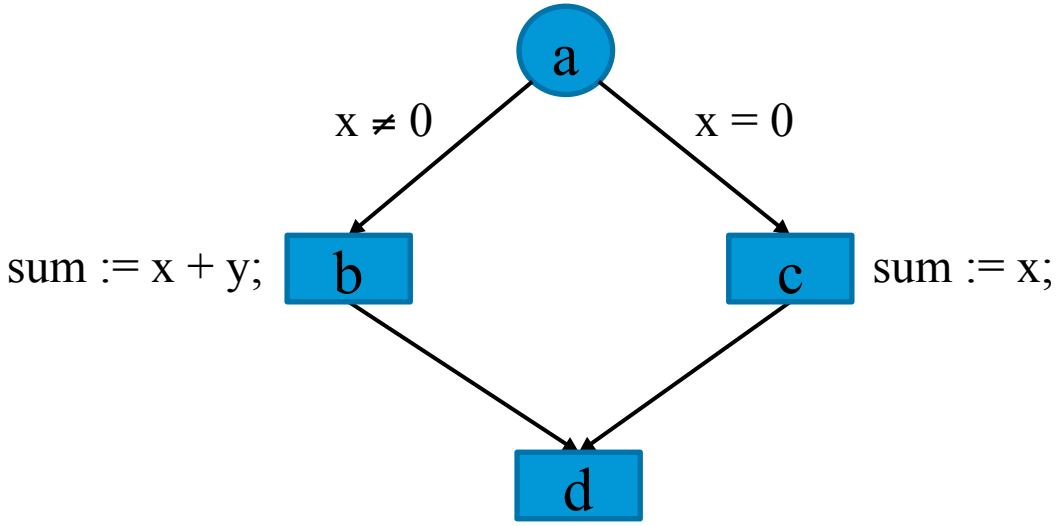
$$\frac{\text{nb de nœuds couverts}}{\text{nb total de nœuds}}$$

TER = 1 <=> le critère *tous-les-nœuds* est satisfait
<=> tous les nœuds du graphe de contrôle sont couverts
<=> toutes les instructions ont été exécutées

Tous-les-nœuds

Cas favorable

```
function sum (x,y : integer) : integer;
    begin
        if (x = 0) then
            sum := x ;
        else
            sum := x + y ;
    end ;
```



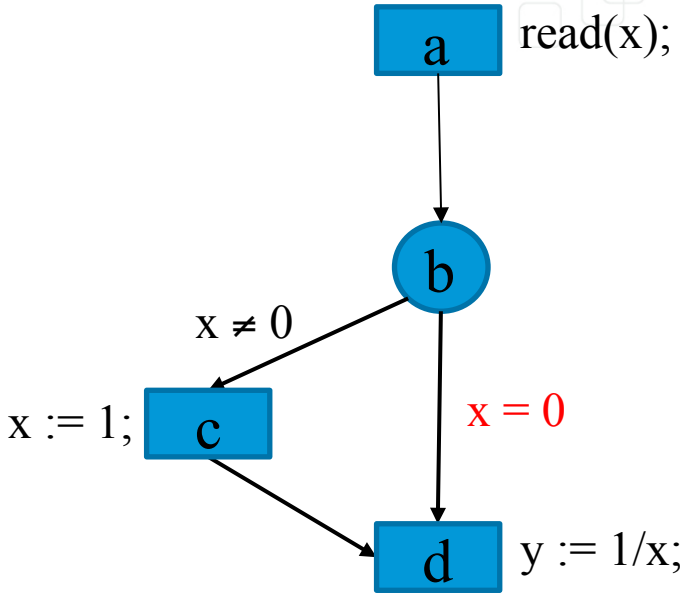
⇒ L ’erreur est détectée par l’exécution du chemin [acd]

Tous-les-nœuds

Limite du critère

Soit le programme suivant (avec erreur) :

```
read (x) ;
if (x ≠ 0) then
    x := 1;
y  := 1/x;
```



⇒ Le critère *tous-les-nœuds* est satisfait par le chemin [abcd] sans que l'erreur ne soit détectée.
L'unique donnée de test {x = 1} permet de couvrir tous les nœuds du graphe sans faire apparaître l'anomalie.

Couverture de tous-les-arcs

Couverture des branches

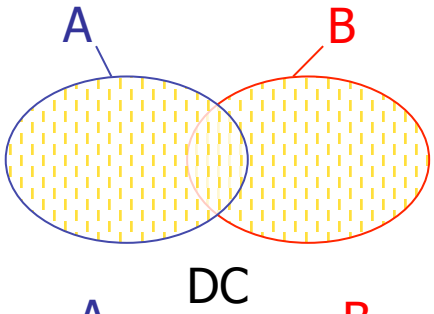
- Chaque arc est couvert par au moins l'un des chemins parmi les chemins qui constituent le jeu de test
- La valeur de vérité de chaque nœud de décision a été au moins une fois vraie et une fois fausse

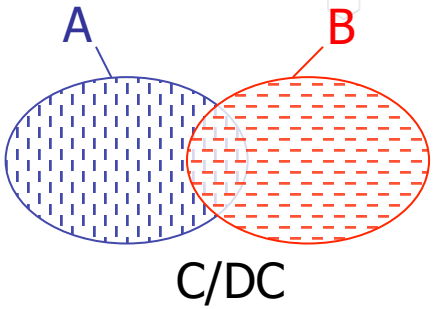
- Lorsqu'un jeu de test permet de couvrir tous les arcs du graphe, on dit qu'il satisfait TER=2 ou TER2 (Test Effectiveness Ratio 2)

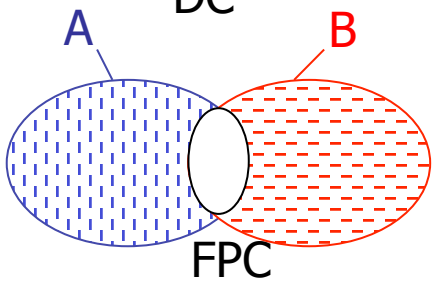
$$\frac{\text{nb d'arcs couverts}}{\text{nb total d'arcs}}$$

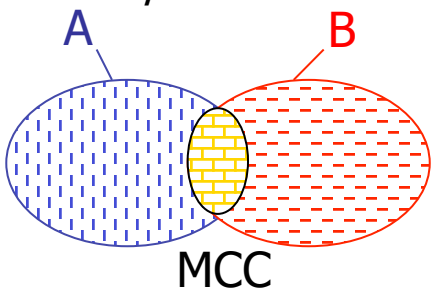
TER = 2 <=> le critère *tous-les-arcs* est satisfait
<=> tous les arcs du graphe de contrôle ont été couverts
<=> toutes les décisions ont été exécutées

Couverture des Conditions / Decisions









Décision (A v B) :

- Couverture des Décisions → A v B
- Couverture des Conditions dans les Décisions → A , B
- Couverture des Conditions Exclusives → A ∧ ¬B , ¬ A ∧ B
- Couverture des Conditions Multiples → A ∧ B , A ∧ ¬ B , ¬ A ∧ B

Tous-les-arcs

Limite du critère

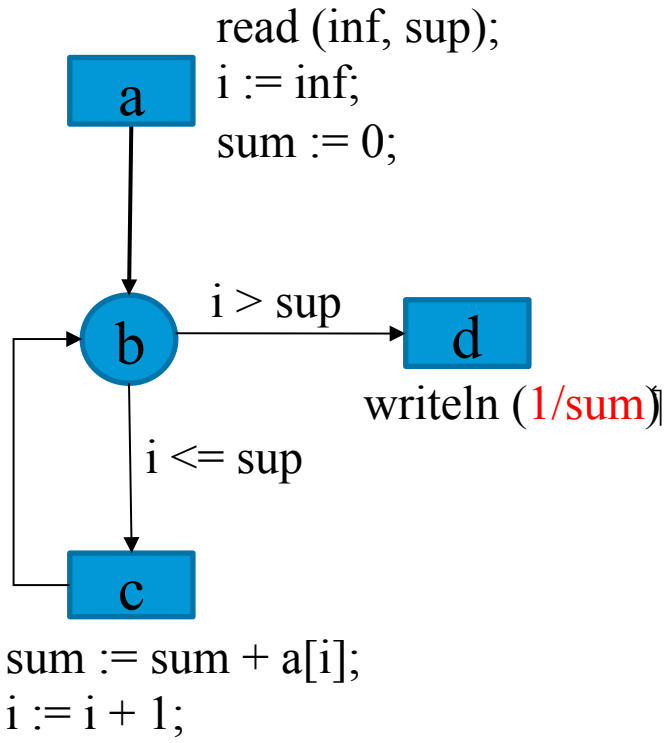
Pas de détection d'erreurs **en cas de non-exécution d'une boucle**

⇒*tous-les-arcs* est satisfait par le chemin [abcbd]

⇒ La donnée de test suivante couvre le critère *tous-les-arcs* :

DT1= {a[1]=50, a[2]=60, a[3]=80, inf=1,sup=3}

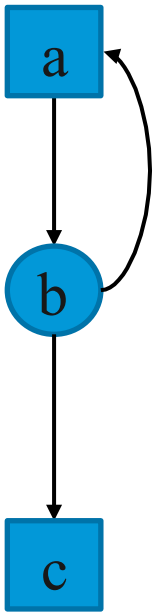
⇒ Problème non détecté par le critère *tous-les-arcs* : si inf > sup, erreur sur 1/sum



Couverture des boucles

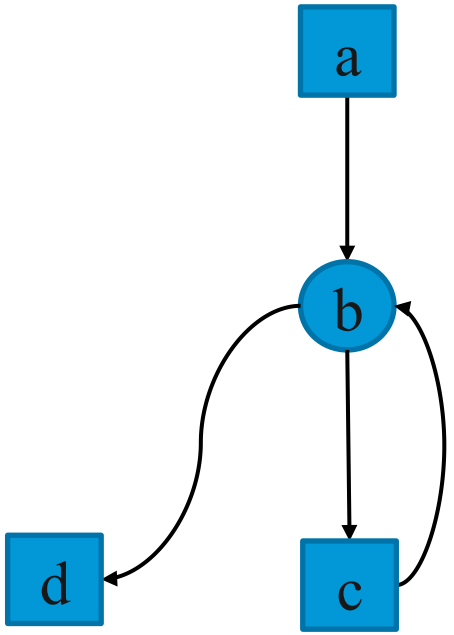
Chemins limites et intérieurs

- Chemins limites : traversent la boucle, mais ne l'itèrent pas
- Chemins intérieurs : itèrent la boucle une seule fois



Chemin limite : [abc]
chemin intérieur : [ababc]

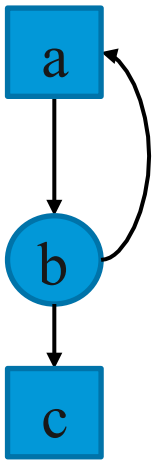
chemin limite : [abd]
chemin intérieur : [abcbd]



Couverture de tous les i-chemins

Généralise couverture chemins limites et intérieurs

- Couverture de tous les chemins possibles passant de 0 à i fois dans chaque boucle du graphe de contrôle
- La couverture de *tous les i-chemins* (pour i > 0) garantit les critères TER1, TER2 et le critère de couverture des chemins limites et intérieurs



Le jeu de tests constitué de données de test permettant de couvrir les chemins [abc], [ababc] et [abababc] satisfait le critère *tous les 2-chemins.*

Couverture de tous les chemins indépendants

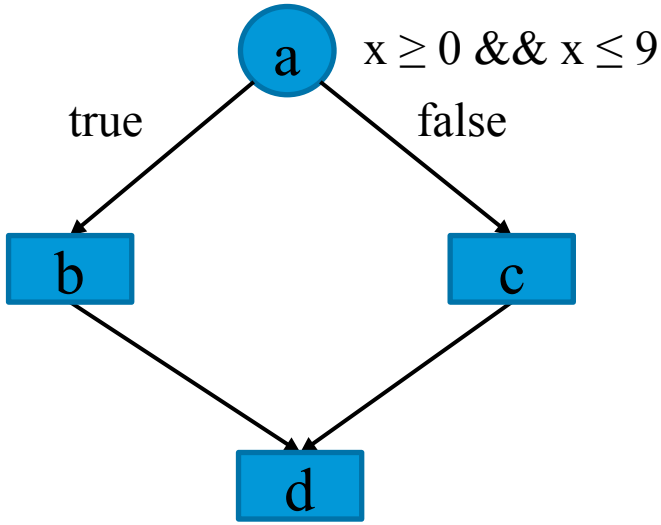
- Déterminer un ensemble de chemins représentatif de tous les comportements
- Construire une base de chemins qui par combinaison linéaire donne tous les chemins
- Considérer les vecteurs des chemins par rapport aux arcs

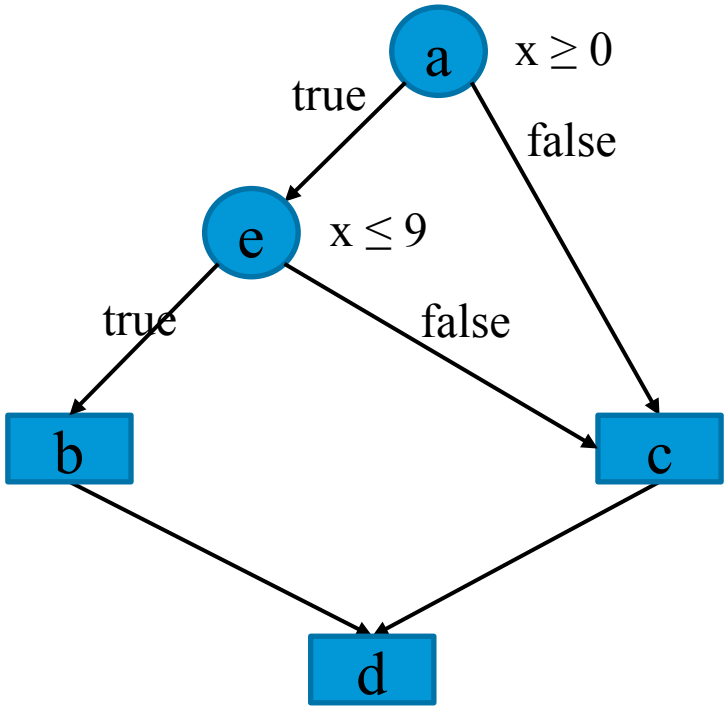
- Le nombre de chemins nécessaire est donné par la complexité cyclomatique du graphe (Mc Cabe)

$$\nu(G) = \text{nombre de conditions élémentaires} + 1$$

Chemins indépendants

Conditions élémentaires





2 conditions élémentaires
3 chemins linéairement indépendants

Chemins indépendants

Algorithme

1. Evaluer V(G)
2. Choisir un chemin dans le graphe en privilégiant dans l'ordre : les chemins extérieurs, puis les chemins intérieurs, puis les 2-chemins… dans les boucles
3. Produire une donnée de test couvrant le chemin choisi. En cas d'impossibilité, reprendre à l'étape 2 en affaiblissant le critère
4. Choisir un chemin couvrant au moins un arc non encore couvert par les données de test précédemment choisies, tout en essayant d'en couvrir le moins possible (les stratégies utilisées à l'étape 1 restent valides)
5. Produire une donnée de test couvrant le chemin choisi à l'étape 4. En cas d'impossibilité, rejeter le chemin choisi et reprendre l'étape 4 pour choisir un autre chemin

…

Chemins indépendants

Algorithme

6. Vérifier l'indépendance du chemin calculé par rapport aux autres chemins déjà calculés (2 chemins sont indépendants ssi il existe un arc qui est couvert par l'un et pas par l'autre)
7. Recommencer les étapes 4 à 6 jusqu'à la couverture de toutes les décisions
8. S'il n'y a plus de décisions à étudier, que le nombre de chemins trouvés est inférieur à V(G) et que lors des étapes 3 et/ou 5 des chemins ont été rejetés, il est probable que la combinaison des décisions dans le code rendent la couverture inatteignable en l'état. Il faut alors envisager une reprise du code.

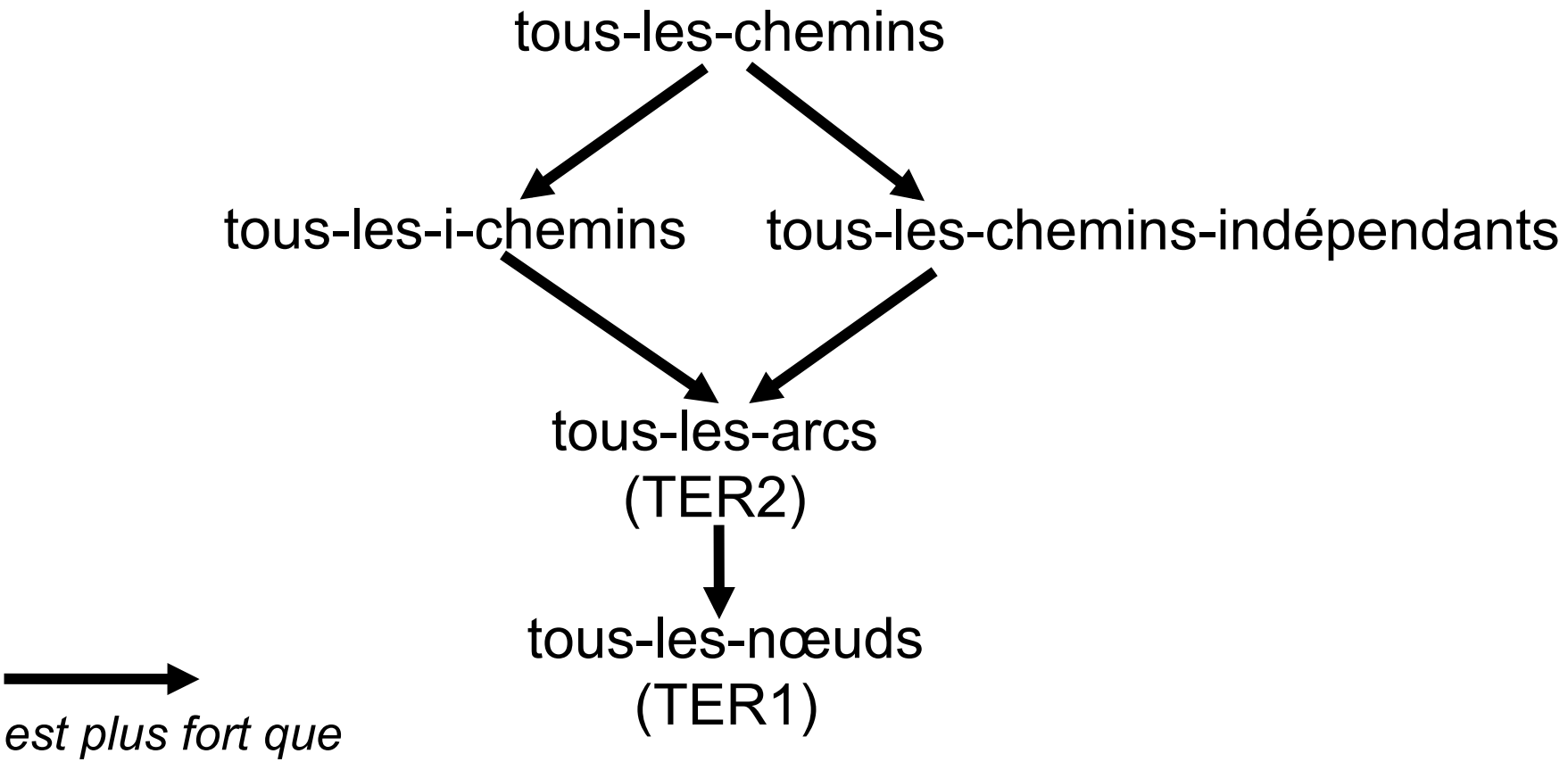
Couverture de tous les chemins

- En posant *i=n* (avec *n* le nombre maximal d'itérations possibles) pour le critère *tous les i-chemins* revient à exécuter tous les chemins possibles du graphe de contrôle

  => Critère *tous-les-chemins*

- En pratique, ce critère est rarement envisageable sur des applications même modestes (explosion du nombre de données de test nécessaires)

Hiérarchie des critères basés sur le graphe de contrôle



Le flot de données

- Le flot de données est représenté en annotant le graphe de contrôle par certaines informations sur les manipulations de variables par le programme :

  - Définition de variables : la valeur de la variable est modifiée

  Exemple : membre gauche d'une affectation, paramètre d'une instruction de lecture…

  - Utilisation de variables : accès à la valeur de la variable

  Exemple : membre droit d'une affectation, paramètre d'une instruction d'écriture, indice de tableau, utilisée dans une condition dans une instruction conditionnelle, dans une boucle…

- Si une variable utilisée l'est dans le prédicat d'une instruction de décision (if, while, etc…), il s'agit d'une *p-utilisation*
- Dans les autres cas (par exemple dans un calcul), il s'agit d 'une *c-utilisation.*

Définition et utilisation de variables

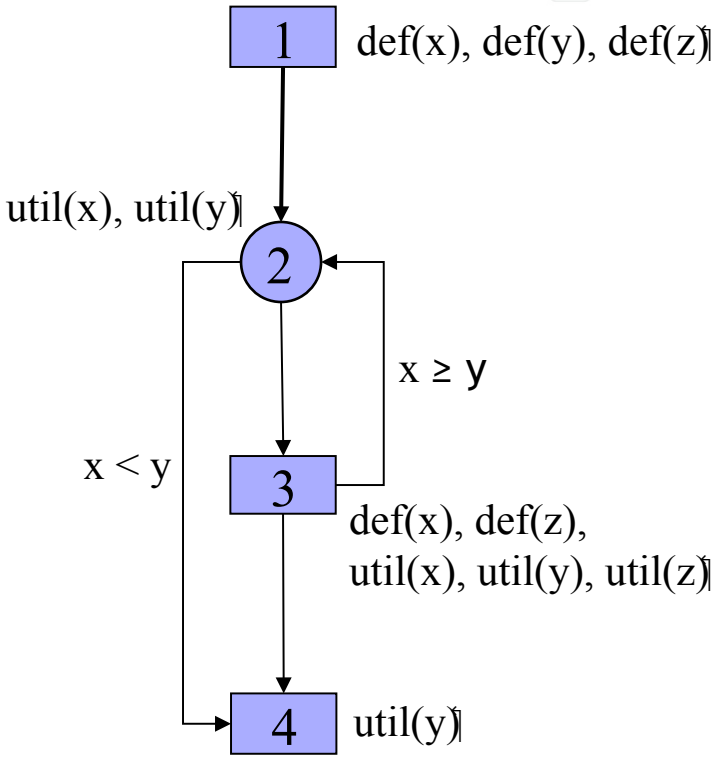
```
open (fichier1) ;
read (x,fichier1) ;
read (y,fichier1) ;
z := 0 ;
---------------------------
while x ≥ y do
---------------------------
    begin
        x := x-y ;
        z := z+1 ;
    end
---------------------------
open (fichier2) ;
print (y,fichier2) ;
close (fichier1) ;
close (fichier2) ;
```

Bloc 1
Définitions : x, y, z
Utilisations : aucune

Bloc 2
Définitions : aucune
Utilisations : x, y

Bloc 3
Définitions : x, z
Utilisations : x, y, z

Bloc 4
Définitions : aucune
Utilisations : y



Critères de couverture sur le flot de données

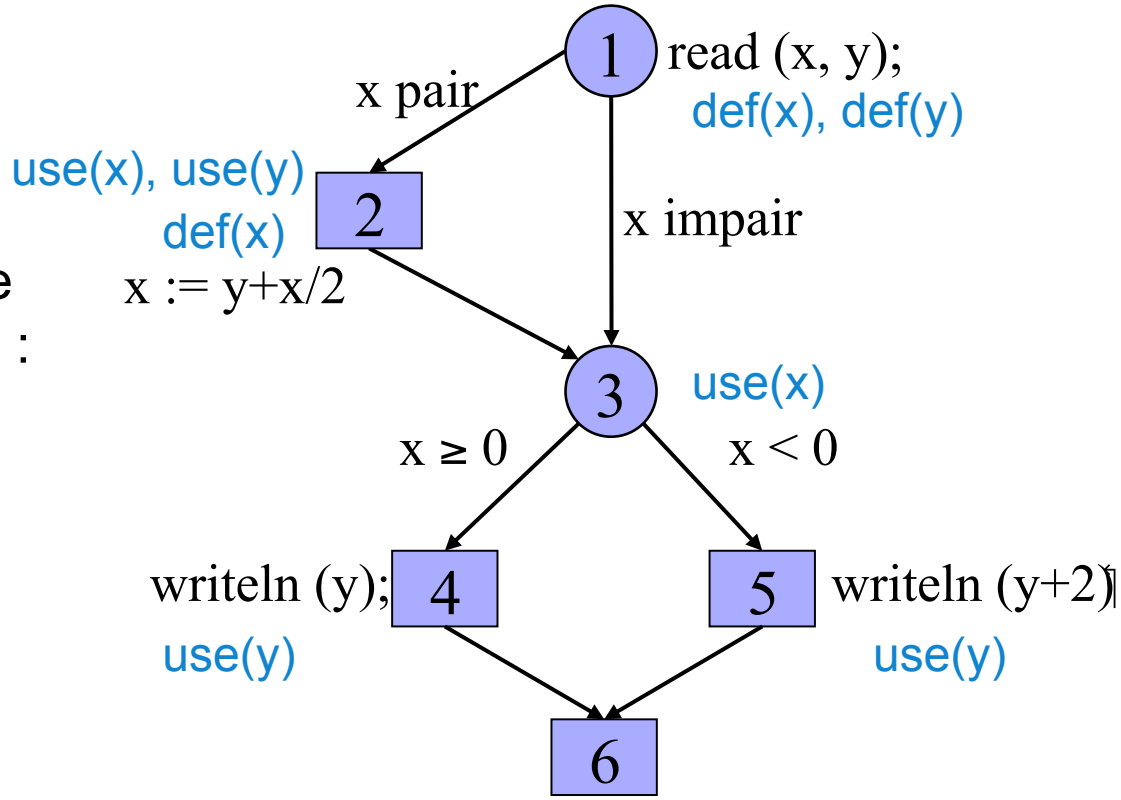
- Toutes les définitions
- Toutes les utilisations
- Tous les DU-chemins

Couverture de toutes les définitions

Pour chaque définition, il existe au moins un chemin qui la couvre dans un test

Couverture du critère toutes-les-définitions :

- [1,2,3,5,6]

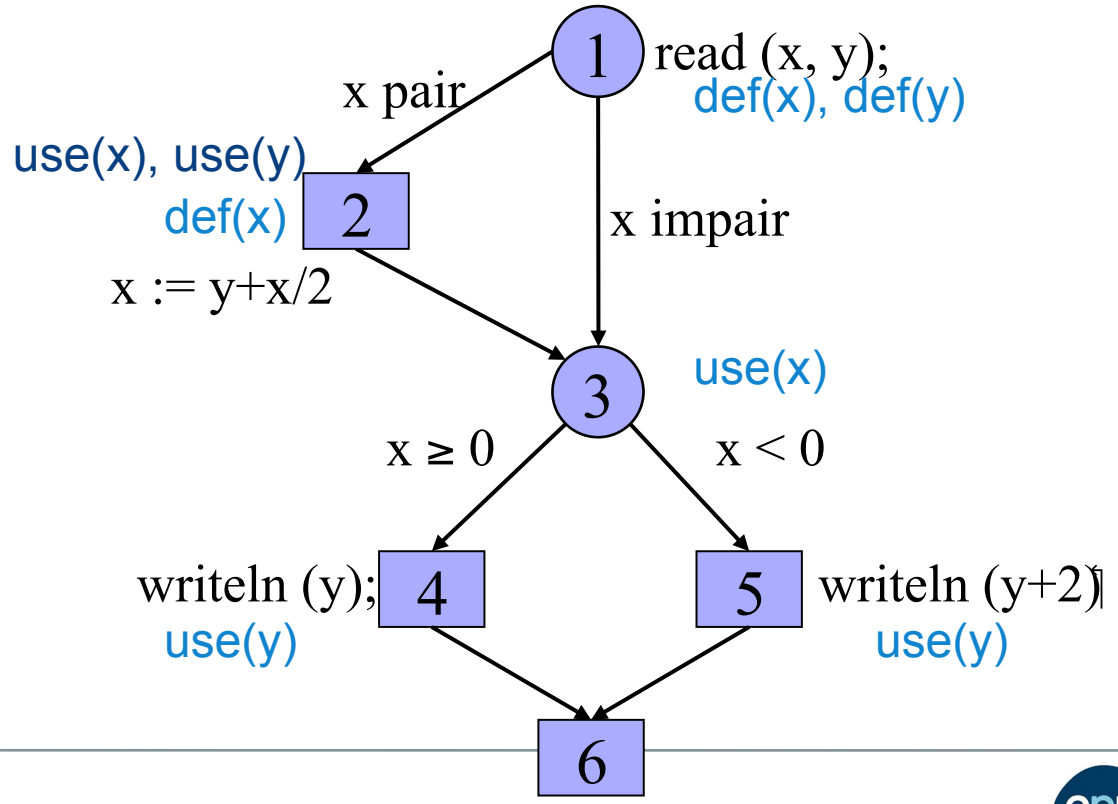


Couverture de toutes les utilisations

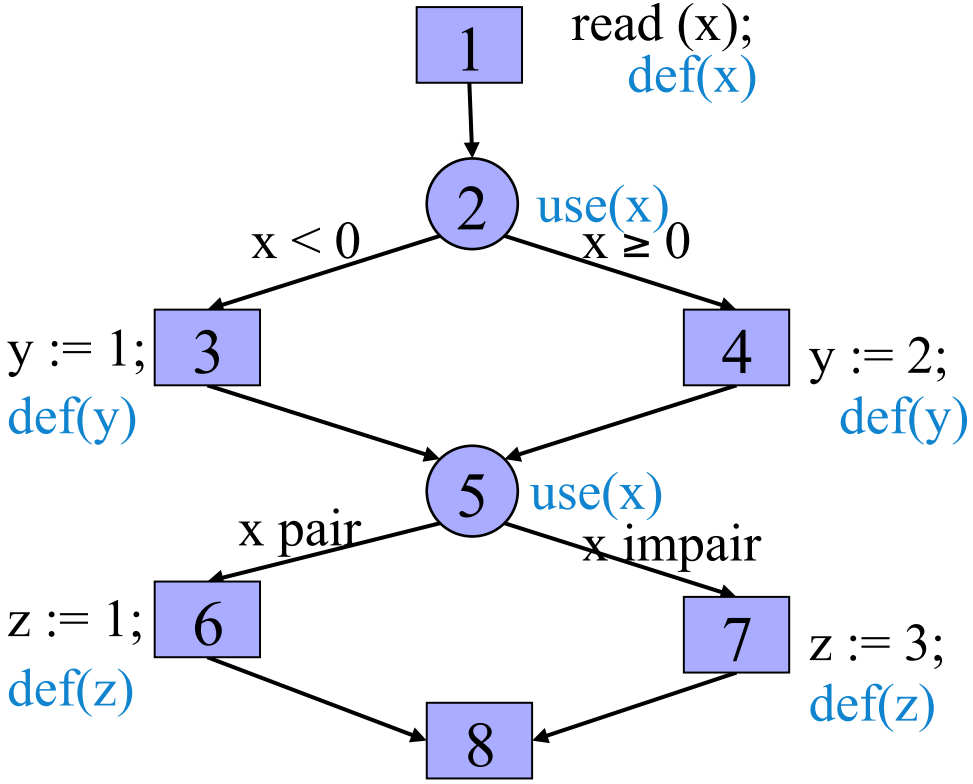
Pour chaque définition et pour chaque utilisation accessible à partir de cette définition, couverture d'un chemin de la définition à l'utilisation

Couverture du critère toutes-les-utilisations :

- [1,3,4,6]
- [1,2,3,4,6]
- [1,3,5,6]
- [1,2,3,5,6]



Limite du critère toutes-les-utilisations



Couverture du critère toutes-les-utilisations :

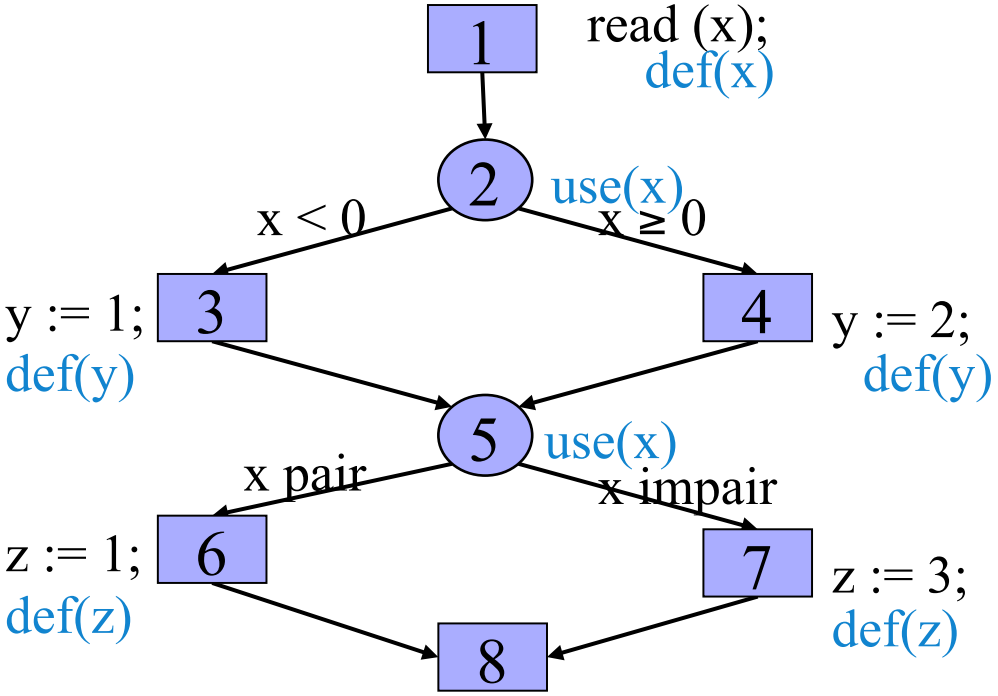
- [1,2,3,5,6,8]
- [1,2,4,5,7,8]

Ces deux tests ne couvrent pas tous les chemins d'utilisation :
(8) peut être utilisatrice de (3) pour la variable y et de (7) pour la variable z

=> critère *tous-les-DU-chemins*

Couverture de tous les DU chemins

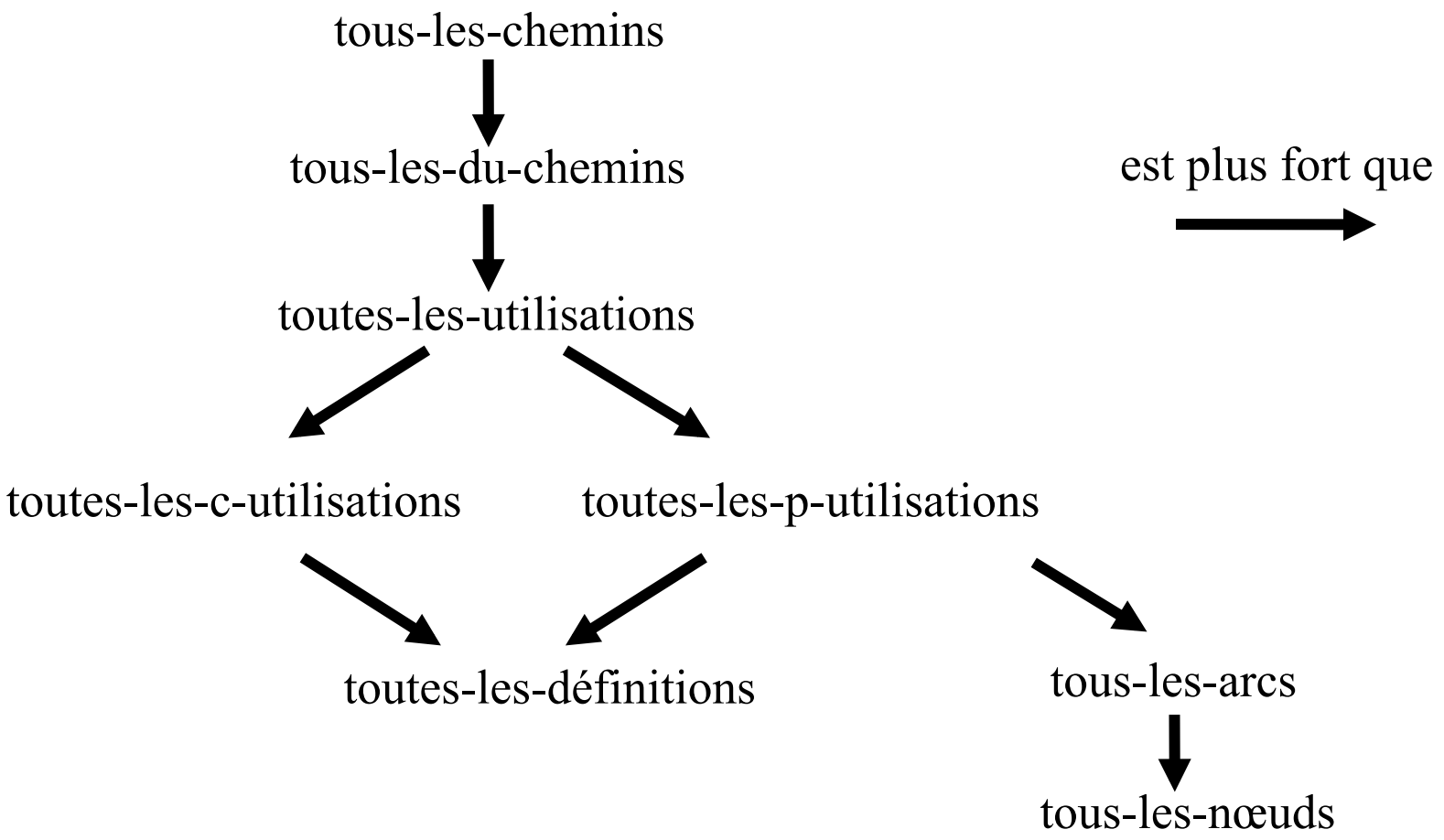
Couvrir tous les chemins possibles entre la définition et la référence, en se limitant aux chemins sans cycle.



Couverture du critère tous les DU chemins :

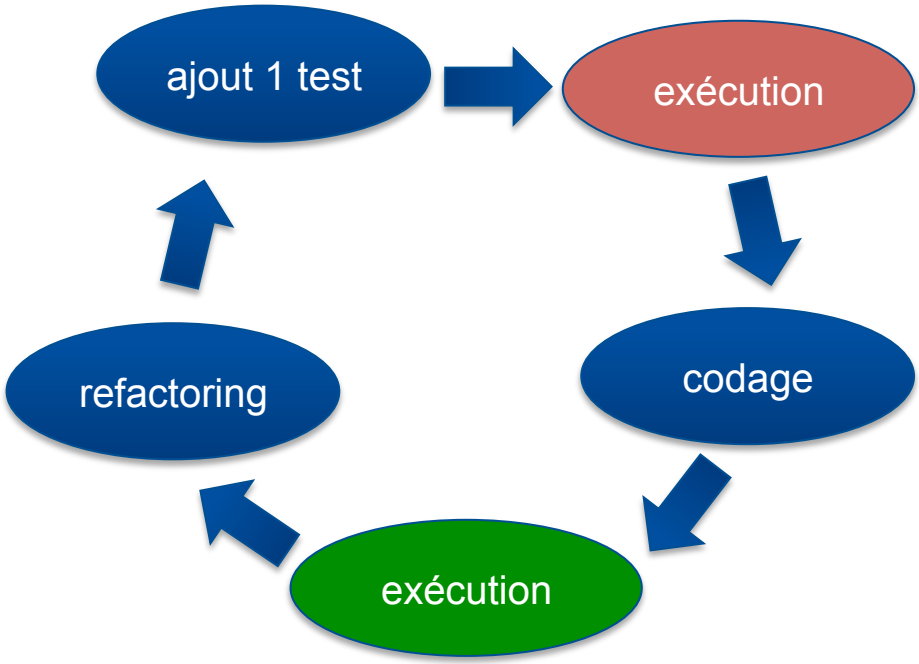
- [1,2,3,5,6,8]
- [1,2,4,5,7,8]
- [1,2,3,5,7,8]
- [1,2,4,5,6,8]

Hiérarchie des critères basés sur le flot de données



TDD: Test Driven Development

Une autre vision des tests



- Préconisé par les méthodes agiles
- Écrire le test avant le code
- Le test spécifie un comportement de la fonctionnalité
- Le cycle doit être court
  - pas de test compliqué

Rejeux des tests

Garantir la non régression

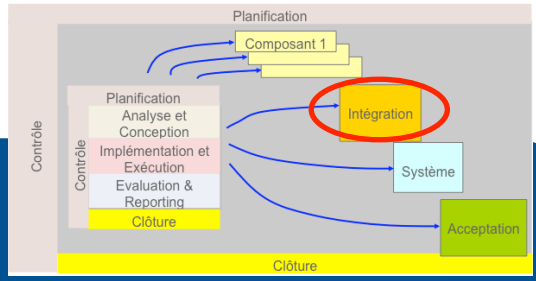
- Une fois écrit, les tests unitaires ont vocation à être rejoués
  - chaque fois que la fonctionnalité est impactée
  - avant chaque commit du développeur
- Il faut donc veiller à ce que :
  - leur nombre ne devienne pas trop important
  - leur temps de passage soit acceptable
- Il faut organiser les tests de sorte à ne pas exécuter systématiquement tous les tests unitaires

3. Automatisation des tests d'intégration

Stratégies propres à l'intégration : big-bang, « au fil de l'eau », par incréments etc…
Intégration ascendante versus descendante.
Objets simulacres : bouchons, mocking pour remplacer un objet
Intégration continue
Focus sur un gestionnaire de configuration logiciel.
Signalement automatique des anomalies.
Exécution automatique et cyclique des tests logiciels.
Focus sur une constructeur de build.
Focus sur un serveur d'intégration continue.

Stratégies pour le test d'intégration

Voir www.istqb.org Syllabus Fondation et Analyste Technique de Test



Niveau Intégration

- Objectifs génériques
  - Tester les interfaces, les interactions avec un système ou environnement (logiciel et matériel)
- Livrables ou documents de référence pour dériver les cas de test (Base de test)
  - Conception du logiciel et du système, Architecture, Flux de données, Cas d'utilisation
- Objets de test (Ce qui est testé)
  - Interfaces, Bases de données, Infrastructure
- Défauts et défaillances typiques à trouver
  - Erreurs de compilation dans l'environnement d'intégration, Contrats d'interface non respectés
- Harnais de test (environnement requis pour l'exécution)
  - Environnement d'intégration disposant de bouchons et drivers
- Outils nécessaires
- Approche spécifique au niveau (stratégie): Big-Bang, Bottom-Up ou Top-Down

Stratégies pour le test d'intégration

Termes, selon le glossaire ISTQB

Intégration

- le processus de combiner des composants ou systèmes en assemblages plus grands

Tests d'intégration

- tests effectués pour montrer des défauts dans les interfaces et interactions de composants ou systèmes intégrés. Voir aussi *tests d'intégration de composants*

Tests d'intégration de composants

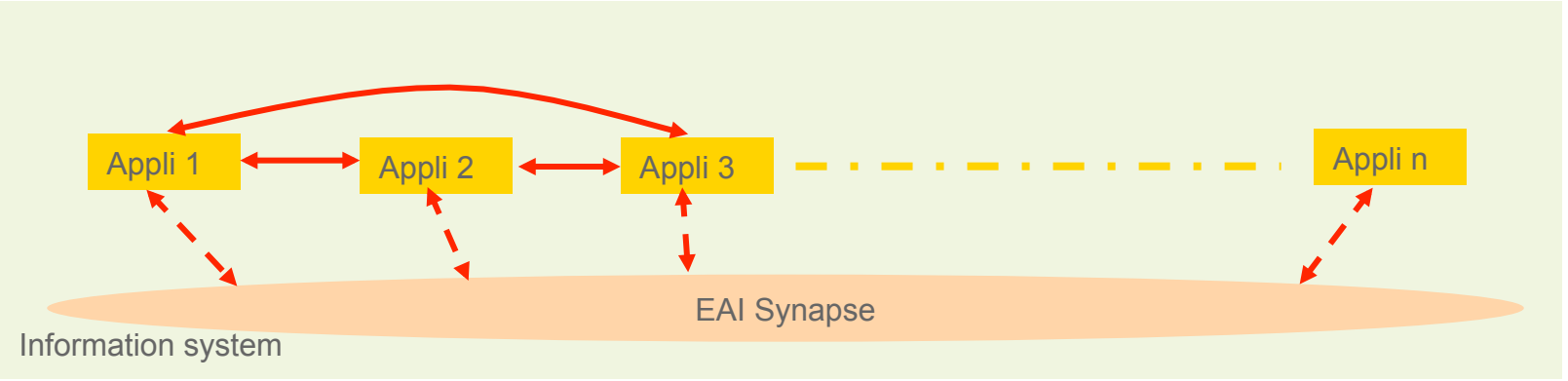
- tests effectués pour découvrir des défauts dans les interfaces et les interactions entre des composants intégrés

Stratégies pour le test d'intégration

Particularités

- Plus précisément, les tests d'intégration testent
  - Les interfaces entre les composants (tests 2 à 2)
  - Les interactions entre différentes parties d'un système (par exemple le système d'exploitation)
  - La communication technique entre applications
  - Le comportement sur incident
  - Quelques éléments d'exploitabilité, de sécurité

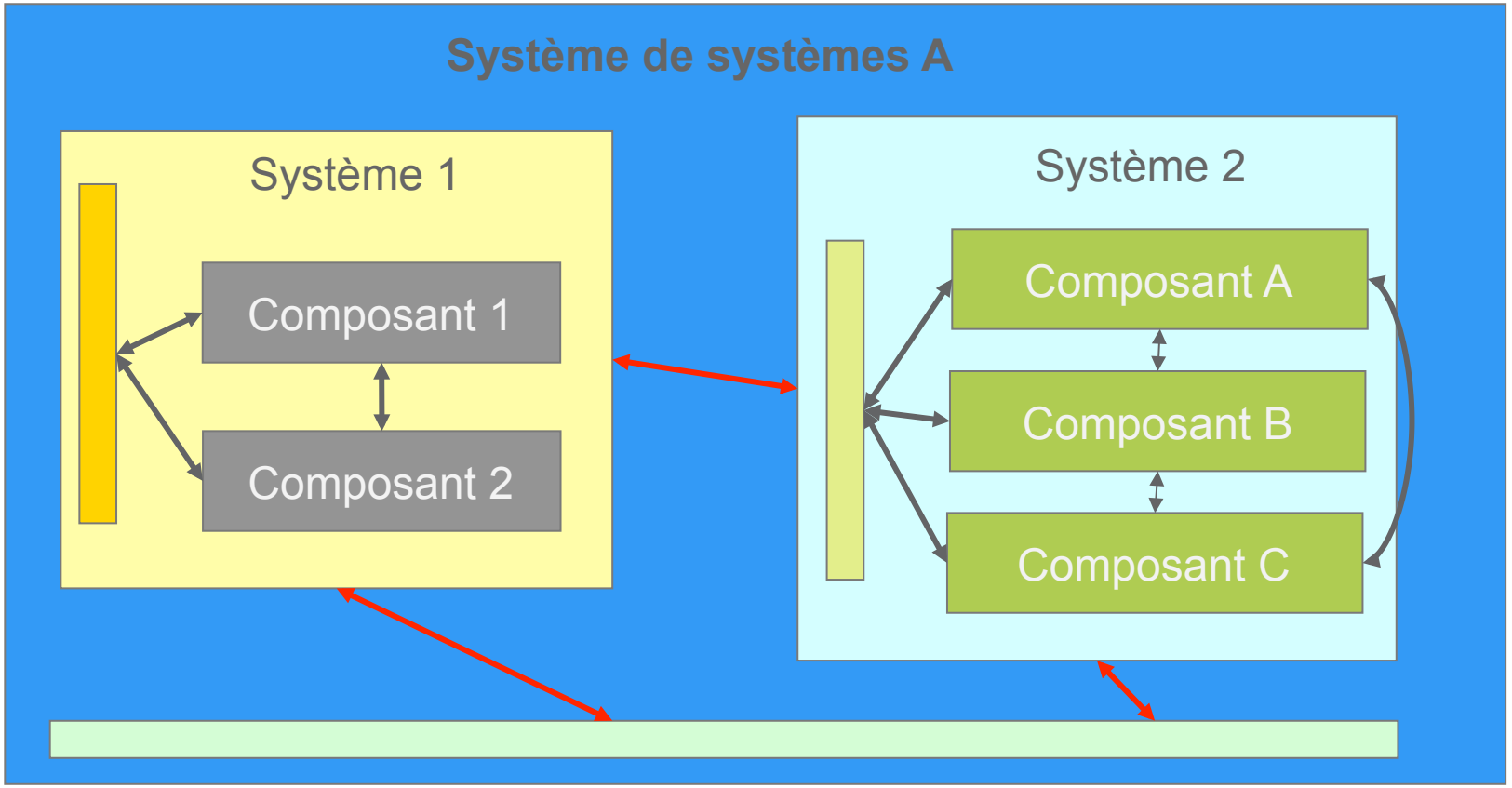
- Exemple: intégration d'applications dans un SI



Stratégies pour le test d'intégration

Particularités: Plusieurs niveaux d'intégration sont possibles

- **Intégration de composants** (formant un système)
- **Intégration de systèmes** (formant un système de systèmes)



Stratégies pour le test d'intégration

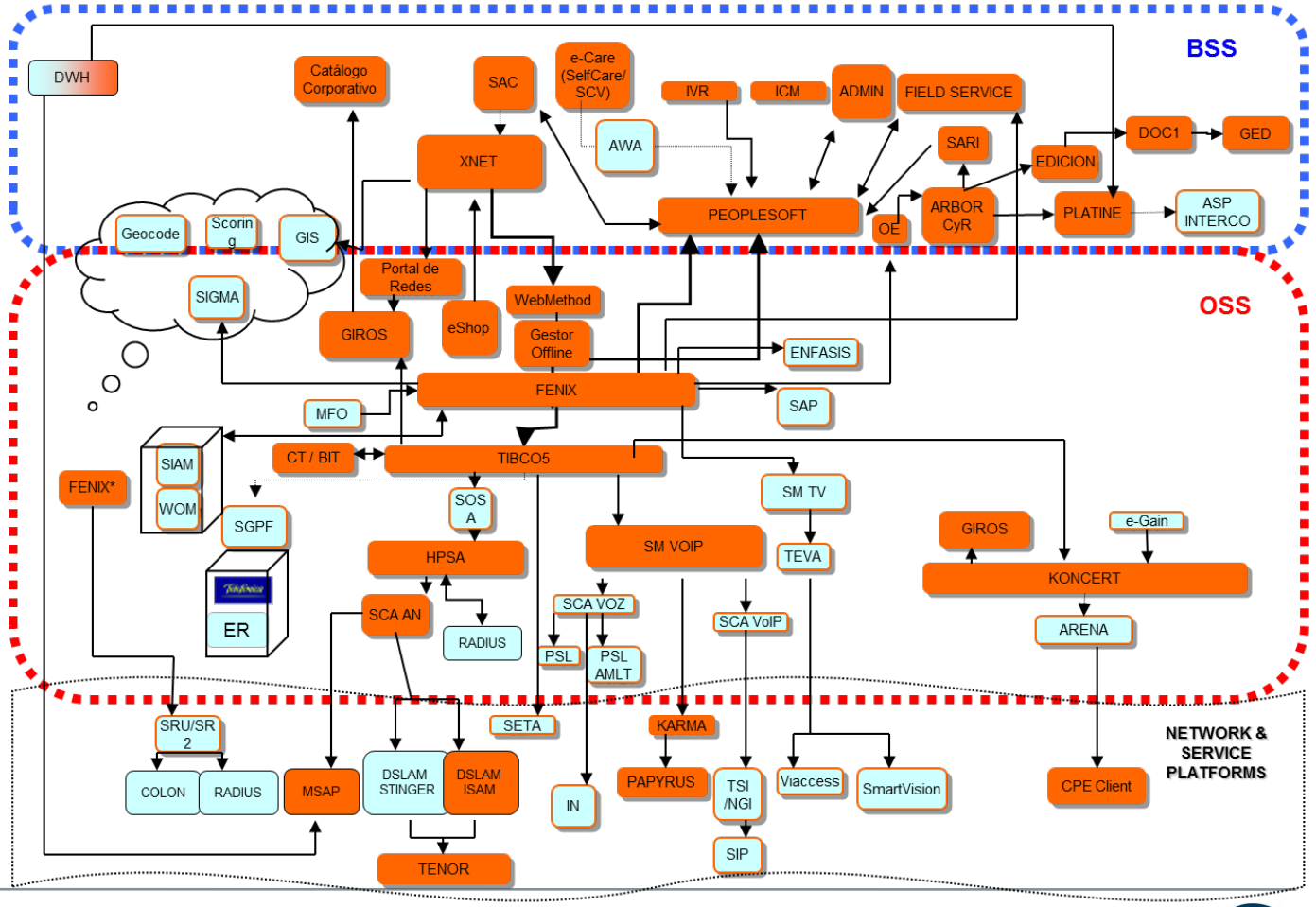
Particularités

- Le test d'intégration est souvent mené après livraison d'un ou plusieurs composants ou systèmes par un sous-traitant ou fournisseur
- La complexité de l'intégration augmente avec le nombre de composants à intégrer
- Plusieurs approches sont possibles en fonction
  - du planning des livraisons
  - des interfaces et interactions entre composants
  - des flux (métiers ou techniques) sollicitant plusieurs composants

Stratégies pour le test d'intégration

Particularités: exemple première partie

- Evolution complexe d'un SI…
  - *Quelles applications simuler?*
  - *Dans quel ordre intégrer?*



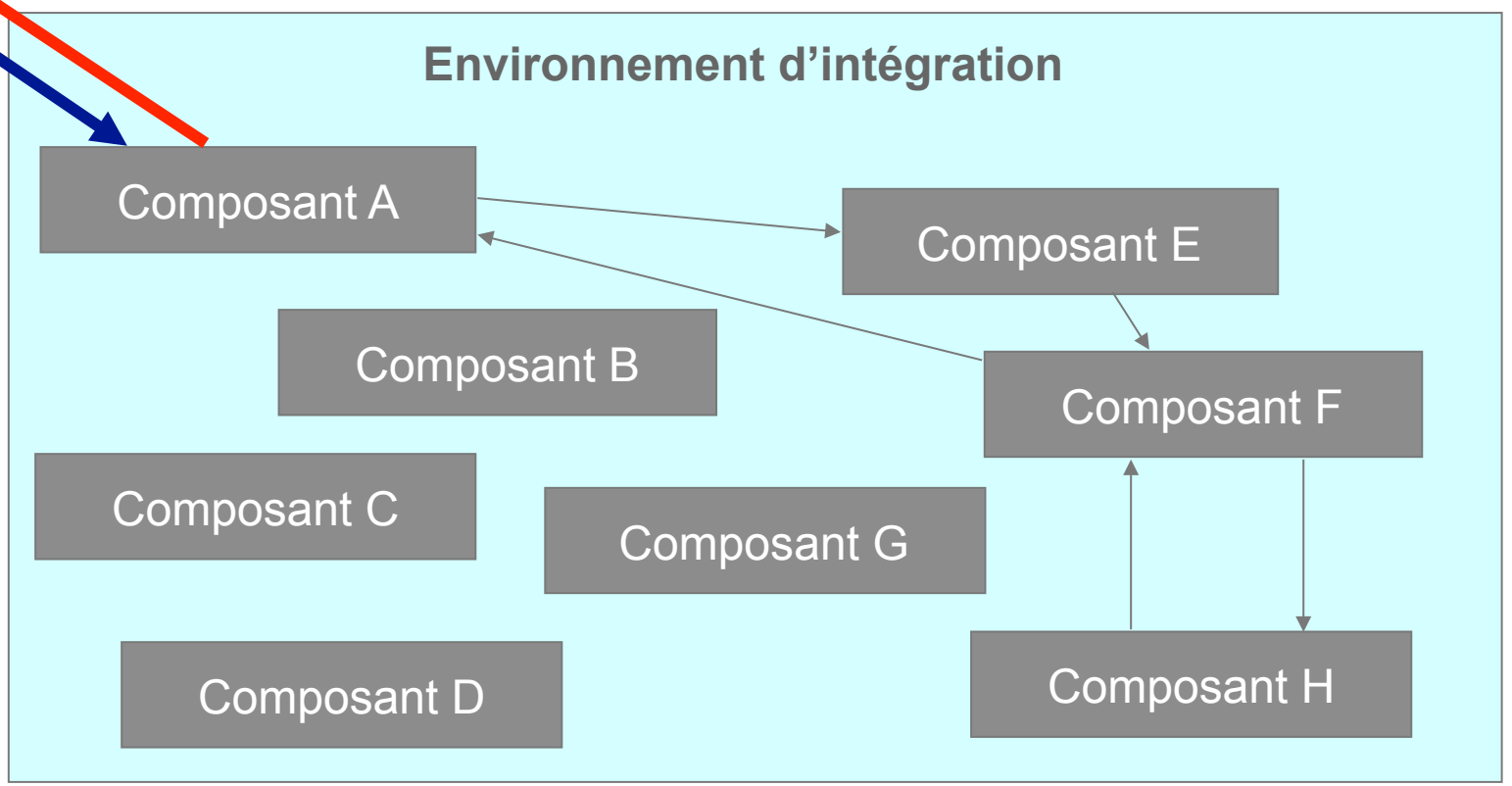
Stratégies pour le test d'intégration

Stratégie d'intégration « Big-Bang »

TEST KO: cherchez l'erreur!

Tous les composants sont intégrés en même temps

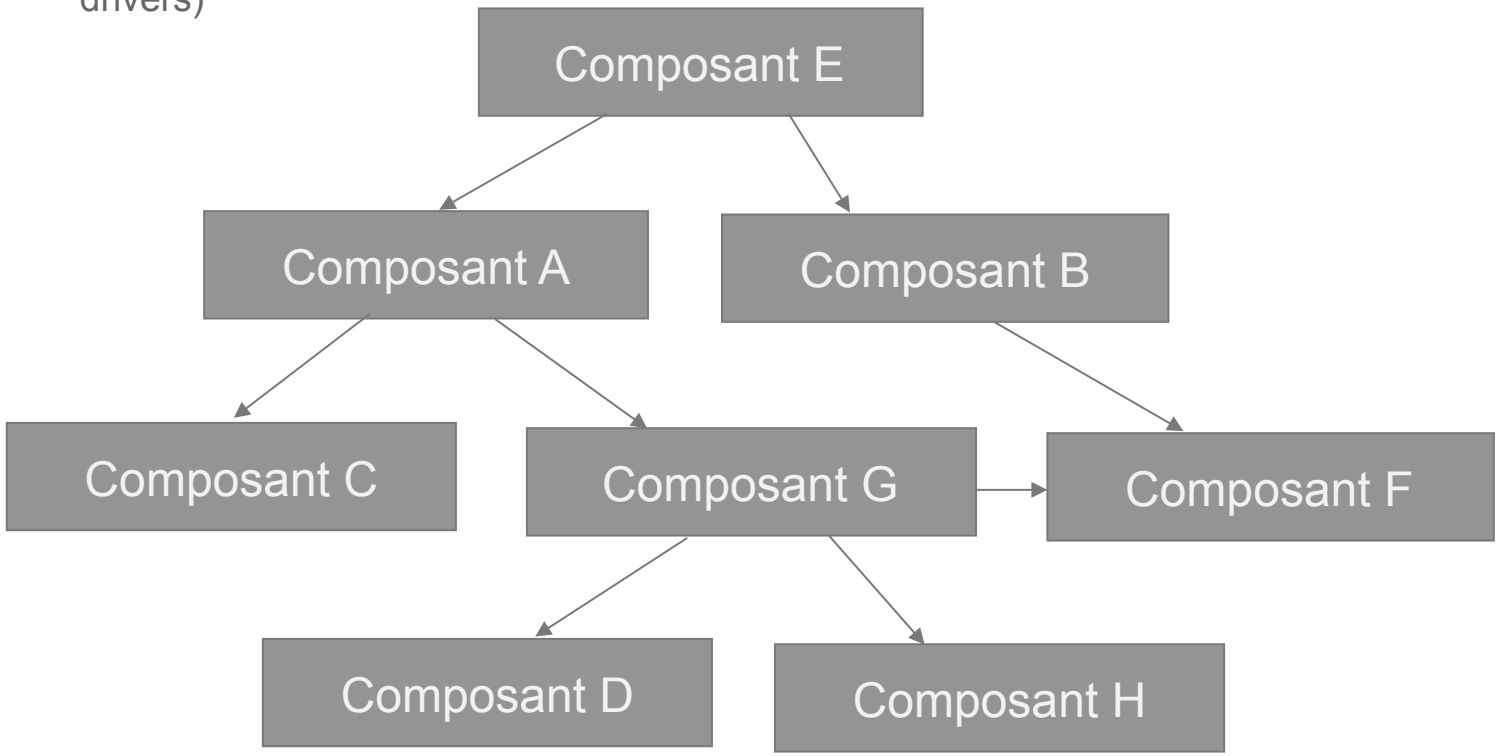
A éviter!



Stratégies pour le test d'intégration

Stratégie d'intégration « Top-Down »

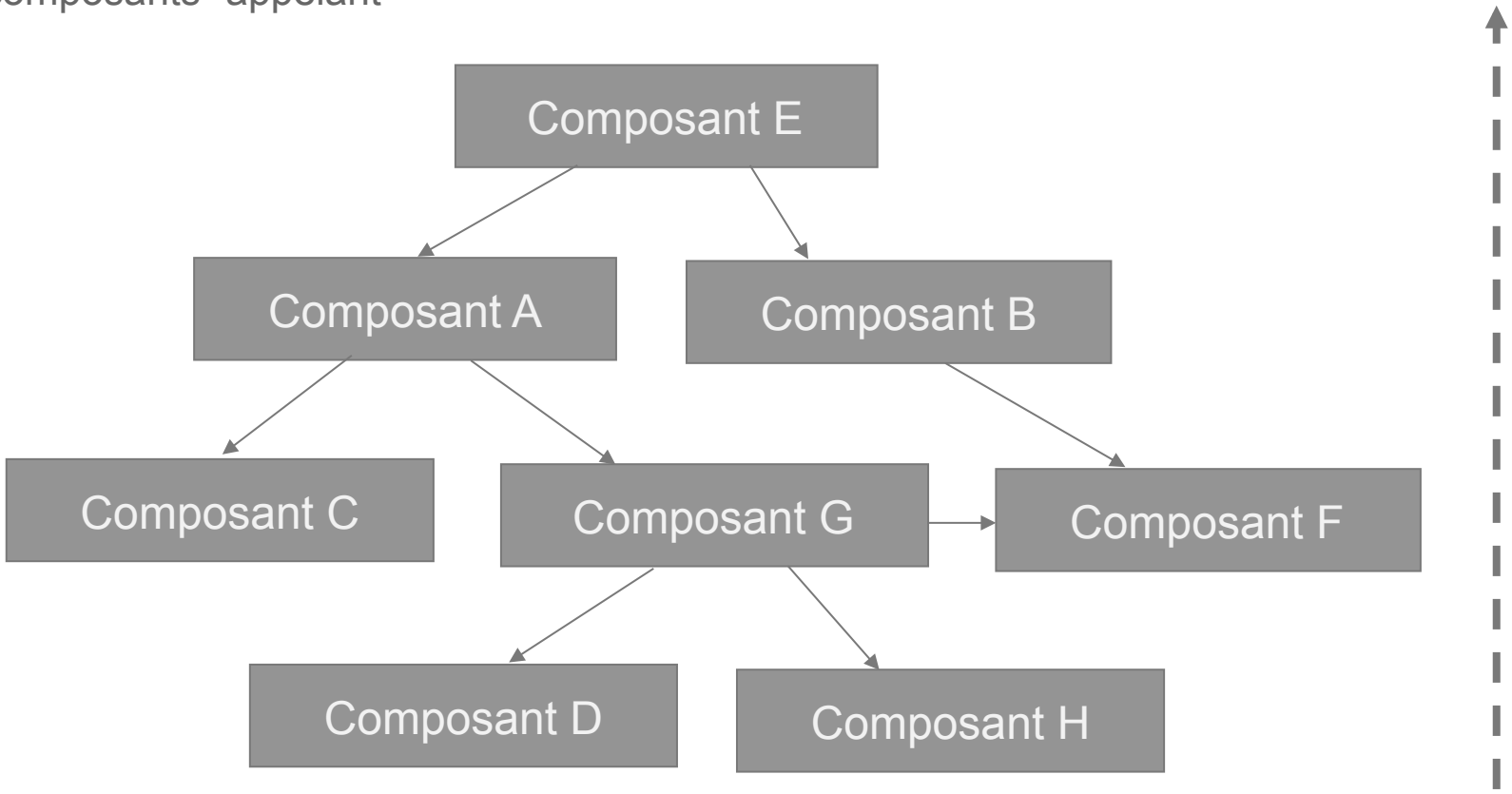
- du composant le moins sollicité vers les composants qu'il sollicite, et ainsi de suite
  - Faire un "graphe d'appel"
  - Se demander par quel composant commencer! (celui qui demande le moins de bouchons/ drivers)



Stratégies pour le test d'intégration

Stratégie d'intégration « Bottom-Up »

- du ou des composants "feuille(s)" (ne sollicitant aucun autre composant) vers les composants "appelant"

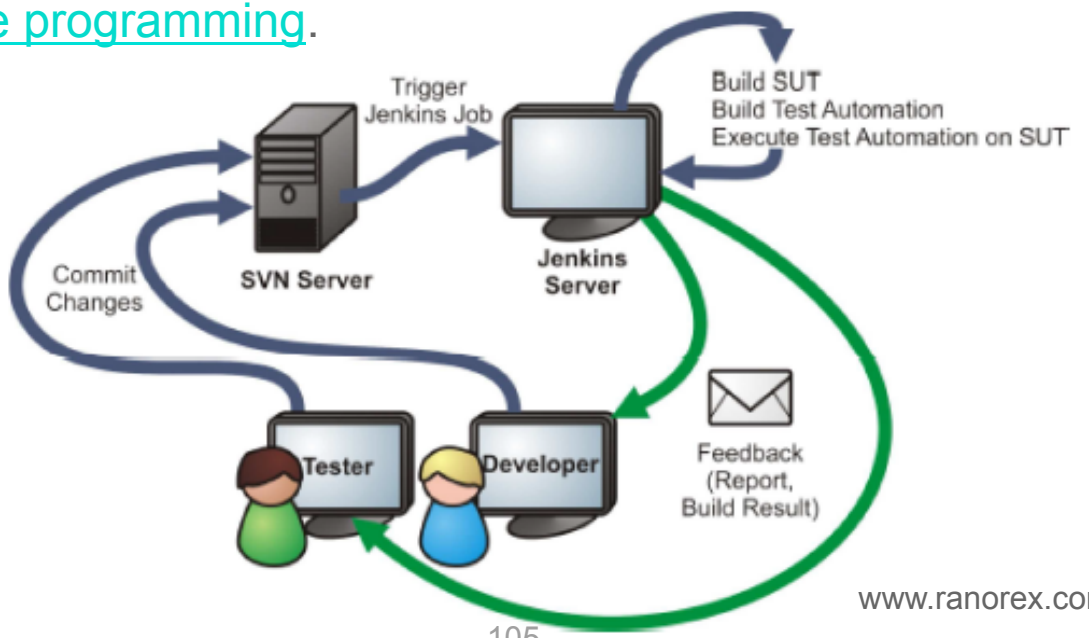


Intégration continue

Définition (source: Wikipedia)

- L'**intégration continue** est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée….

- L'intégration continue se réfère généralement à la pratique de l' extreme programming.



Intégration continue

Prérequis

- Le code source doit être partagé (en utilisant des logiciels de gestion de versions tels que CVS, Subversion, git, Mercurial, etc)
- Les développeurs intègrent (*commit*) quotidiennement (au moins) leurs modifications
- Des tests d'intégration sont développés pour valider l'application (avec JUnit par exemple)
- Il faut un outil d'intégration continue tel que CruiseControl ou Jenkins (anciennement Hudson) pour le langage Java par exemple

Intégration continue

Avantages

- le test immédiat des unités modifiées
- la prévention rapide en cas de code incompatible ou manquant
- les problèmes d'intégration sont détectés et réparés de façon continue, évitant les problèmes de dernière minute
- une version est toujours disponible pour un test, une démonstration ou une distribution

Intégration continue

Pratiques

- Maintenir un dépôt unique de code source versionné
- Automatiser les compilations
- Rendre les compilations auto-testantes
- Tout le monde *commit* tous les jours
- Tout *commit* doit compiler le tronc (trunk) sur une machine d'intégration
- Maintenir une compilation courte
- Tester dans un environnement de production cloné
- Rendre disponible facilement le dernier exécutable
- Tout le monde doit voir ce qui se passe
- Automatiser le déploiement

Étape 1 : mise à jour des sources

Serveur de déploiement

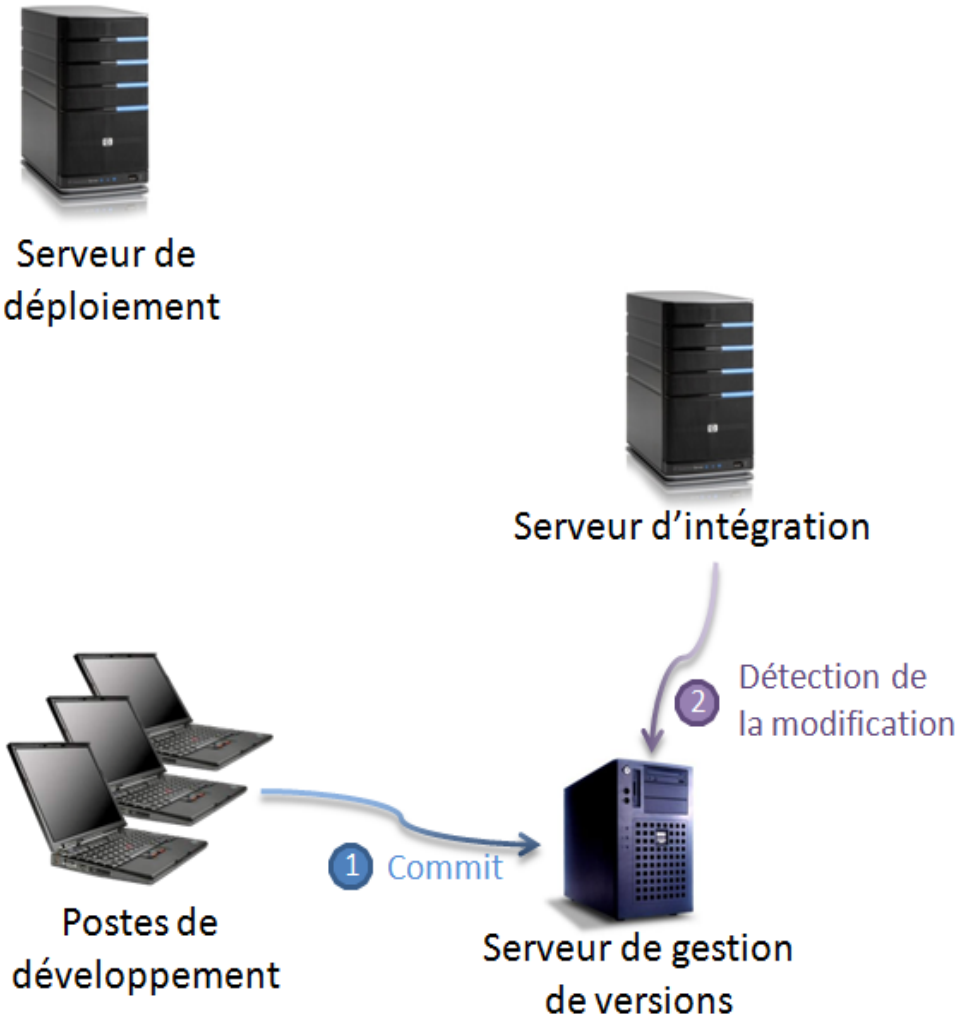
Serveur d'intégration

1 Commit

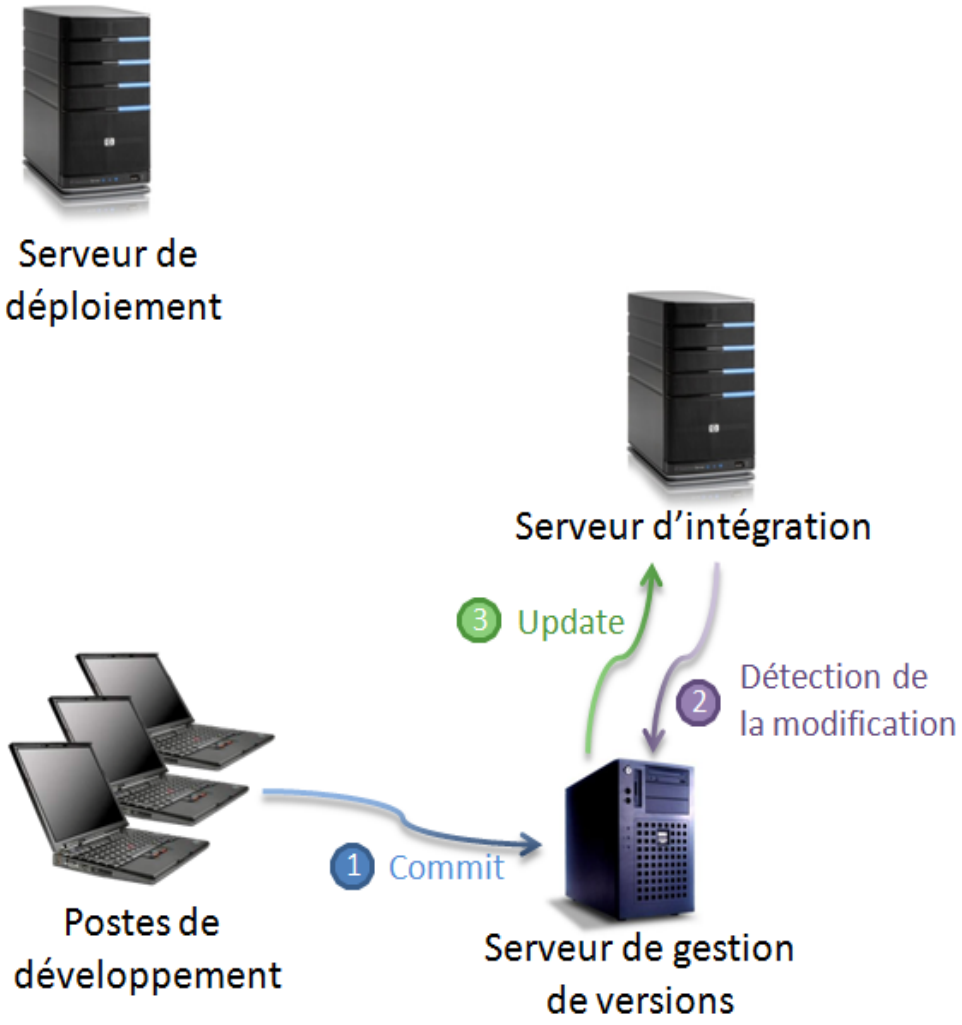
Postes de développement

Serveur de gestion de versions

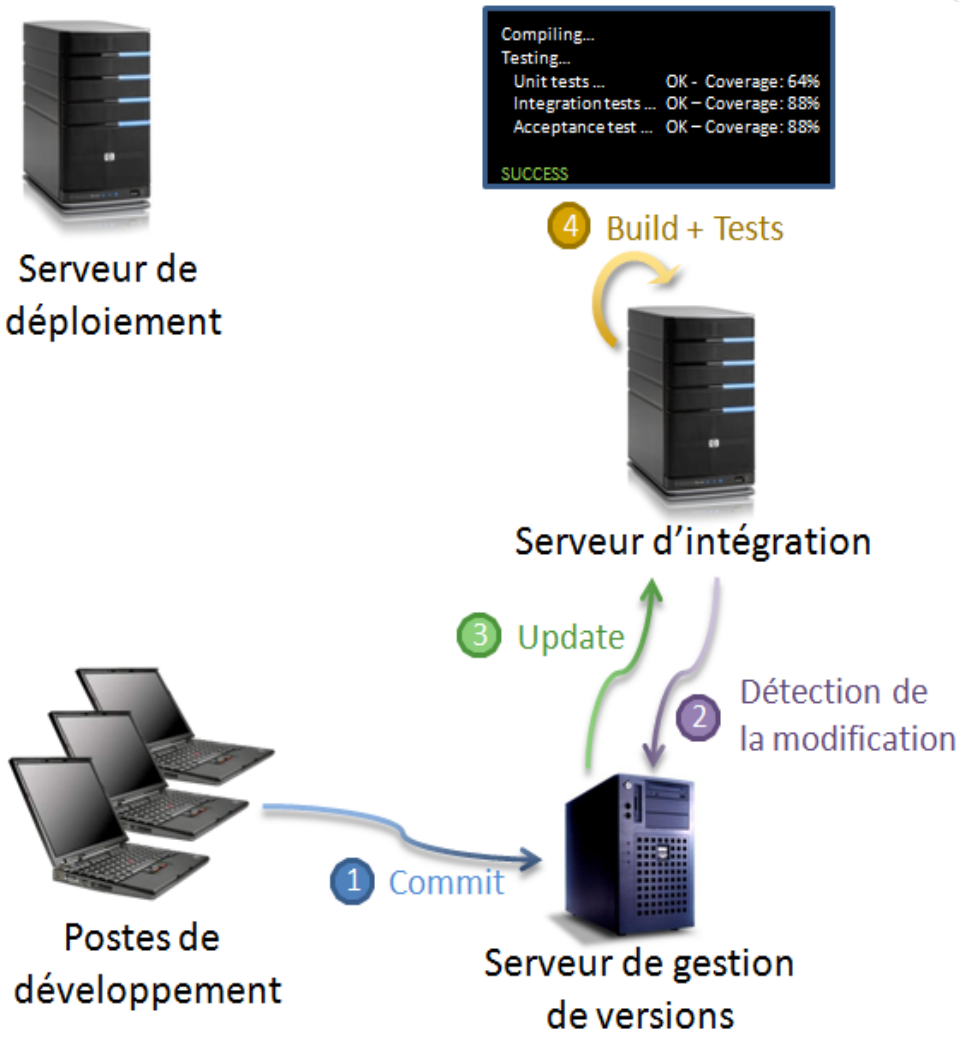
Étape 2 : détection de la mise à jour



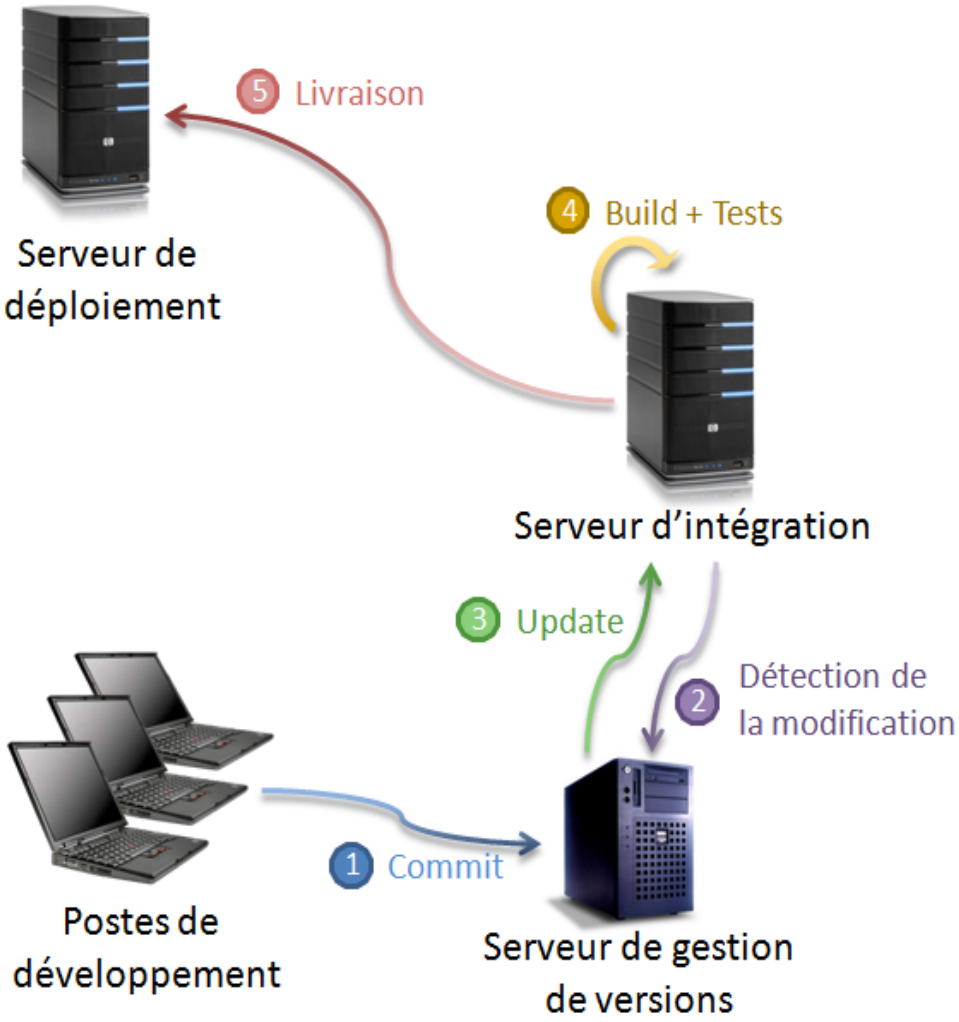
Étape 3 : récupération des sources



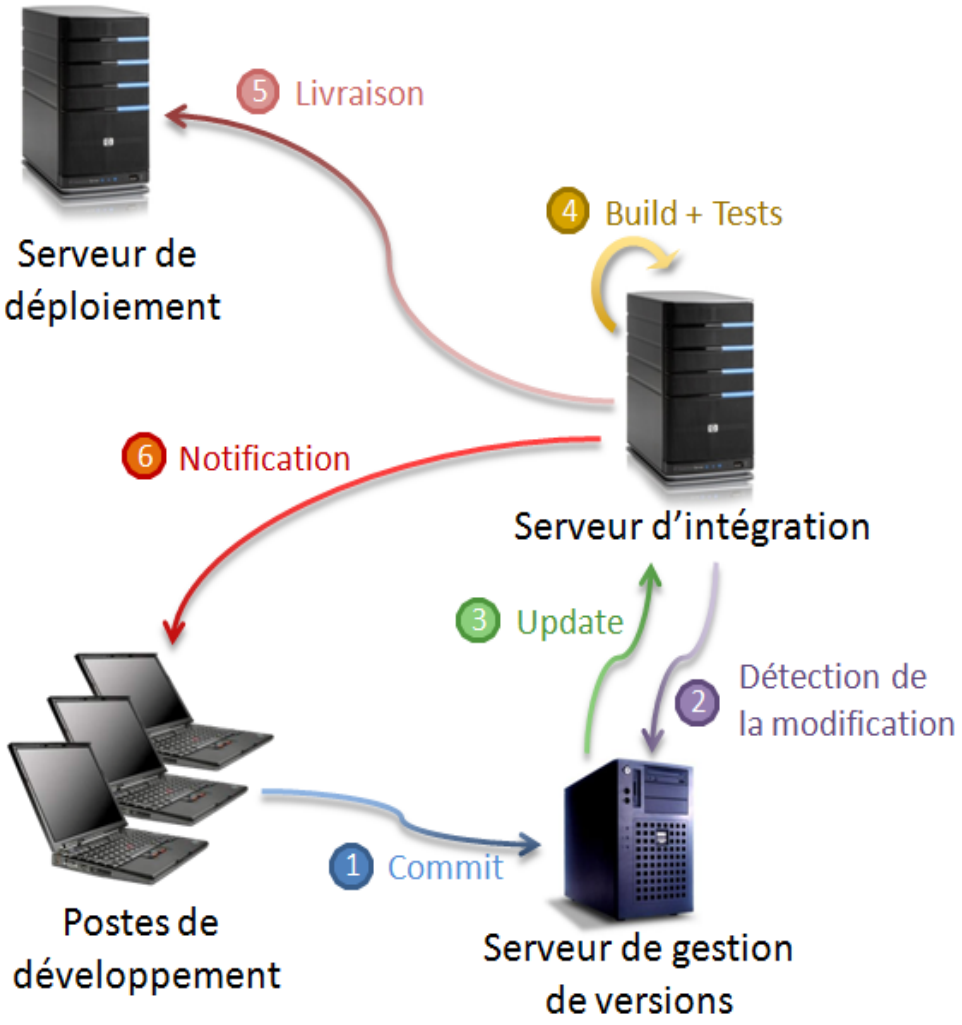
Étape 4 : construction et tests release



Étape 5 : livraison de la release



Étape 6 : publication du rapport



Outils

Jenkins

continuum

Hudson

cruisecontrol

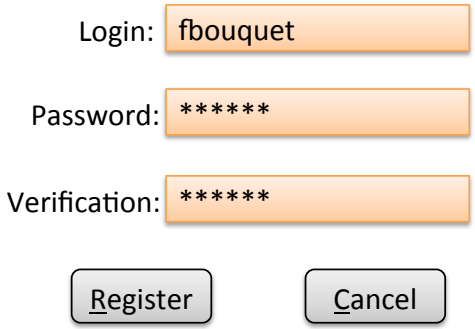
Bamboo

4. Test fonctionnel et son automatisation

Définition du test fonctionnel, non-régression.
Le test simulant l'action des utilisateurs à partir des interfaces utilisateurs (IHM).
Constats sur l'automatisation du test fonctionnel.
Automatisation des tests via l'IHM, via des interfaces de programmation (API).
Chaîne d'outils, robots de test, script (API publiques).
Gestion de l'obsolescence des tests

Test fonctionnel – Exemple (1)

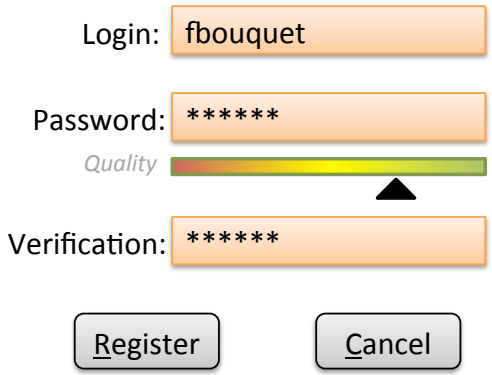
Spécification : "Formulaire d'enregistrement pour un site web."



Quels sont les tests ?

Test fonctionnel – Exemple (2)

Spécification : "Formulaire d'enregistrement pour un site web."

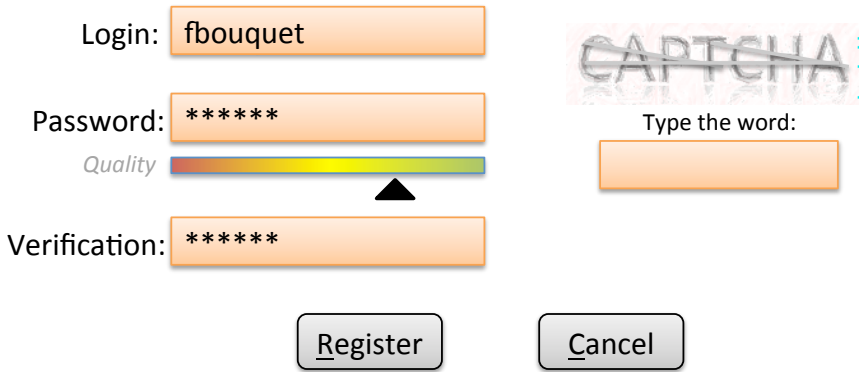


5 Cas : 10 tests logiques

- Login (non) vide (2)
- Login (n') existe (pas) (2)
- Password (non) vide (2)
- Password et Verification (ne) sont (pas) les mêmes (2)
- Protocole http(s) (2)

Test fonctionnel – Exemple (3)

Spécification : "Formulaire d'enregistrement pour un site web."



6 Cas : 13 tests logiques

- Login (non) vide (2)
- Login (n') existe (pas) (2)
- Password (non) vide (2)
- Password et Verification (ne) sont (pas) les mêmes (2)
- Protocole http(s) (2)
- Vérifier qualité du password (1 par niveau) : poor, average, good

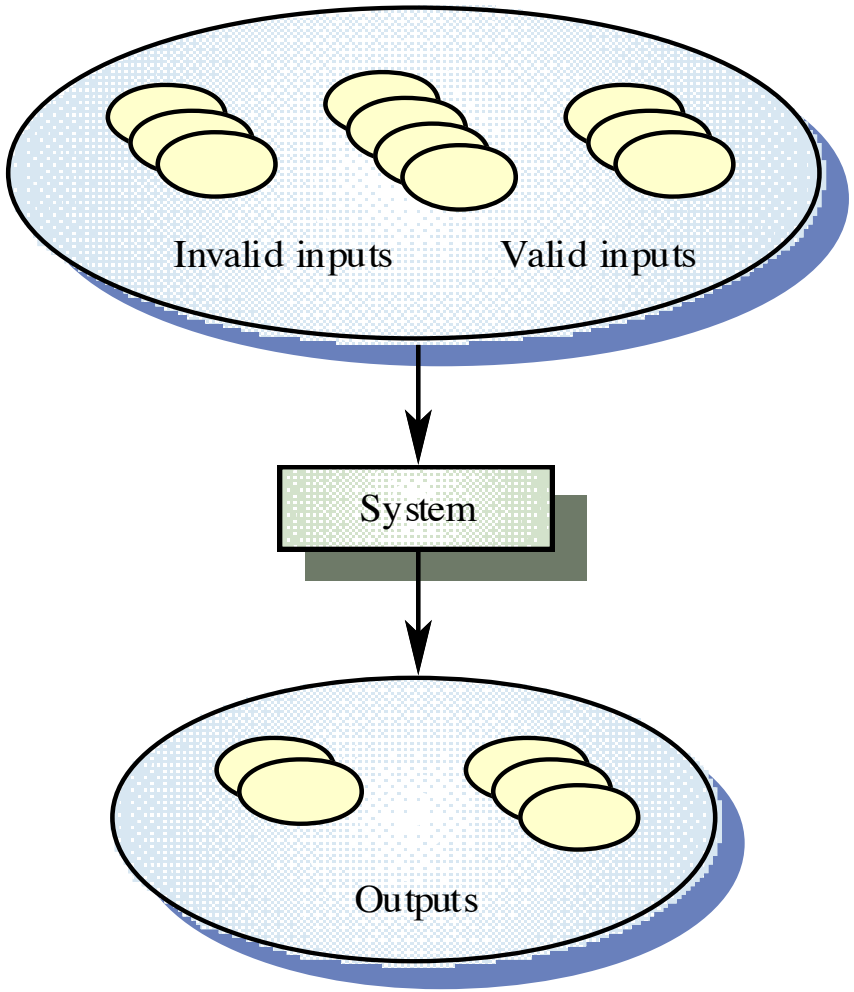
Test fonctionnel – Données des Test

- Le test fonctionnel vise à examiner le comportement fonctionnel du logiciel et sa conformité avec la spécification du logiciel

  ⇨Sélection des Données de Tests (DT)

- Méthodes de sélection :
  - Analyse partitionnelle des domaines des données d'entrée
  - Test aux limites
  - Test combinatoire – Algorithmes Pairwise
  - Test aléatoire
  - Génération de tests à partir d'une spécification

Analyse partitionnelle



Une *classe d'équivalence* correspond à un ensemble de données de tests supposé tester le même comportement,
c'est-à-dire activer le même défaut.

Analyse partitionnelle - Méthode

- Trois phases :
  - Pour chaque donnée d'entrée, calcul de classes d'équivalence sur les domaines de valeurs,
  - Choix d'un représentant de chaque classe d'équivalence,
  - Composition par produit cartésien sur l'ensemble des données d'entrée pour établir les données de test.

Règles de partitionnement

- Si la valeur appartient à un intervalle, construire :
  - une classe pour les valeurs inférieures,
  - une classe pour les valeurs supérieures,
  - n classes valides.

- Si la donnée est un ensemble de valeurs, construire :
  - une classe avec l'ensemble vide,
  - une classe avec trop de valeurs,
  - n classes valides.

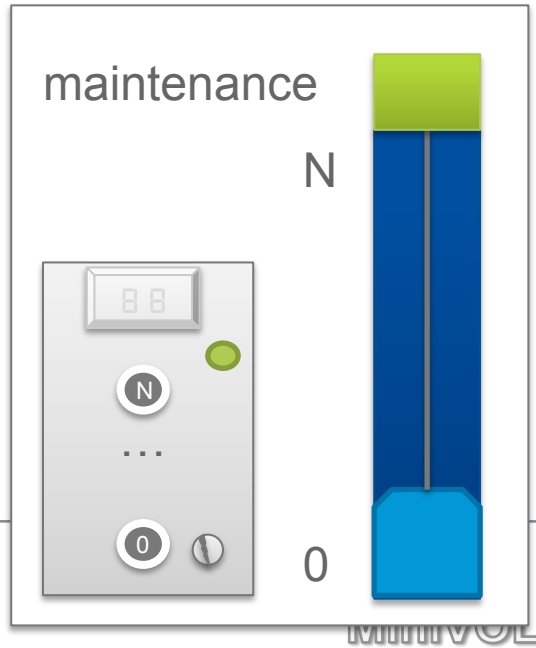
- Si la donnée est une obligation ou une contrainte (forme, sens, syntaxe), construire :
  - une classe avec la contrainte respectée,
  - une classe avec la contrainte non-respectée

Exercice – Classe d'équivalence

On souhaite trouver les classes d'équivalence d'un ascenseur.

Les fonctionnalités sont les suivantes :

- Il est borné par un étage minimum : le rez-de-chaussée, il ne descend donc pas plus bas
- Il est borné par un étage maximum : N, il ne monte donc pas plus haut
- Il a une position de maintenance

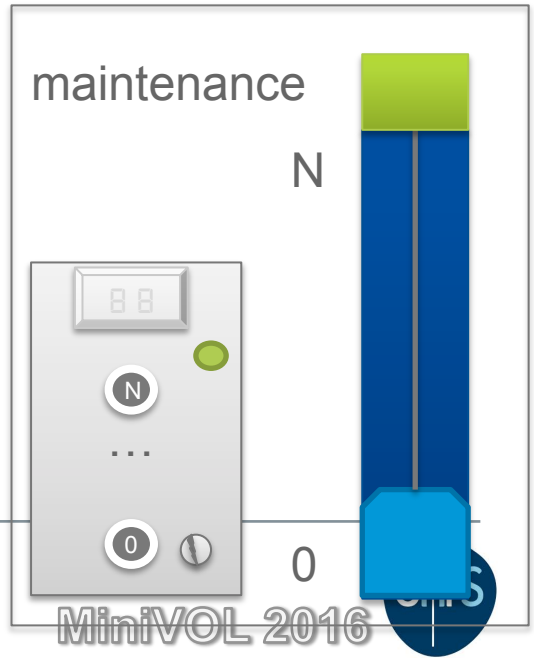


Correction – Classe d'équivalence

On souhaite trouver les classes d'équivalence d'un ascenseur.

Les fonctionnalités sont les suivantes :

- Il est borné par un étage minimum : le rez-de-chaussée, il ne descend donc pas plus bas
- Il est borné par un étage maximum : N, il ne monte donc pas plus haut
- Il a une position de maintenance



Test aux limites

- Principe : on s'intéresse aux bornes des intervalles partitionnant les domaines des variables d'entrées :
  - pour chaque intervalle, on garde les 2 valeurs correspondant aux 2 limites, et les n valeurs correspondant aux valeurs des limites ± le plus petit delta possible :

    $n \in 3 .. 15 \Rightarrow v1 = 3, v2 = 15, v3 = 2, v4 = 4, v5 = 14, v6 = 16$
  - si la variable appartient à un ensemble ordonné de valeurs, on choisit le premier, le second, l'avant dernier et le dernier

    $n \in \{-7, 2, 3, 157, 200\} \Rightarrow v1 = -7, v2 = 2, v3 = 157, v4 = 200$
  - si une *condition d'entrée* spécifie un *nombre de valeurs*, définir les cas de test à partir du nombre *minimum* et *maximum* de valeurs, et des tests pour des nombres de valeurs hors limites invalides.

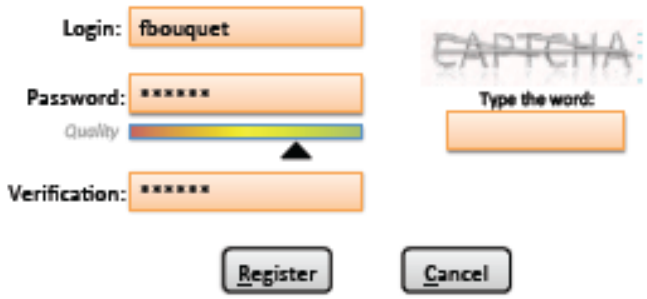
    Un fichier d'entrée contient 1-255 records, produire un cas de test pour 0, 1, 255 et 256.

Test aux limites – Types de données

- Les données d'entrée ne sont pas seulement des valeurs numériques : booléen, image, son, …
- Ces catégories peuvent, en général, se prêter à une analyse partitionnelle et à l'examen des conditions aux limites :
  - True / False
  - Fichier plein / Fichier vide
  - Trame pleine / Trame vide
  - Nuances de couleur
  - Plus grand / plus petit
  - ….

Exemple – Valeur aux limites

Spécification : "Formulaire d'enregistrement pour un site web."



7 Cas : 15 tests logiques

- Login (non) vide (2)
- Login (n') existe (pas) (2)
- Password (non) vide (2)
- Password et Verification (ne) sont (pas) les mêmes (2)
- Protocole http(s) (2)
- Vérifier qualité du password (1 par niveau) : poor, average, good
- Vérifier si enregistrement (non) humain (2)

4 variables :

- Login : vide, court, normal, très longue chaine (+256c), login existant, 'invalide' login
- Password : vide, très longue chaine, même login, poor, average, good
- Password verification : différent du Password, identique
- Captcha : la bonne chaine, pas la bonne

Analyse partitionnelle et test aux limites – synthèse

- L' analyse partitionnelle est une méthode qui vise à diminuer le nombre de cas de test par calcul de classes d'équivalence
  - importance du choix de classes d'équivalence : risque de ne pas révéler un défaut

- Le choix de conditions d'entrée aux limites est une heuristique solide de choix de données d'entrée au sein des classes d' équivalence
  - cette heuristique n'est utilisable qu'en présence de relation d'ordre sur la donnée d'entrée considérée.

- Le test aux limites produit à la fois des cas de test nominaux (dans l'intervalle) et de robustesse (hors intervalle)

Exercice – Test aux limites

Supposons que nous élaborions un compilateur pour un langage. Un extrait des spécifications précise :

≪ L'instruction `FOR` *n'accepte qu'un seul paramètre en tant que variable auxiliaire. Son nom ne doit pas dépasser deux caractères non blancs ; Après le signe = est précisée aussi une borne supérieure et une borne inférieure. Les bornes sont des entiers positifs et on place entre eux le mot-clé* `TO`*.≫*

Par exemple : `I=1 TO 10 ...`

Question : Déterminer par analyse partitionnelle des domaines des données d'entrée les cas de test à produire pour l'instruction FOR.

Exercice – Test au limite correction

Exemples des 19 cas de tests :

Test aux limites - Evaluation

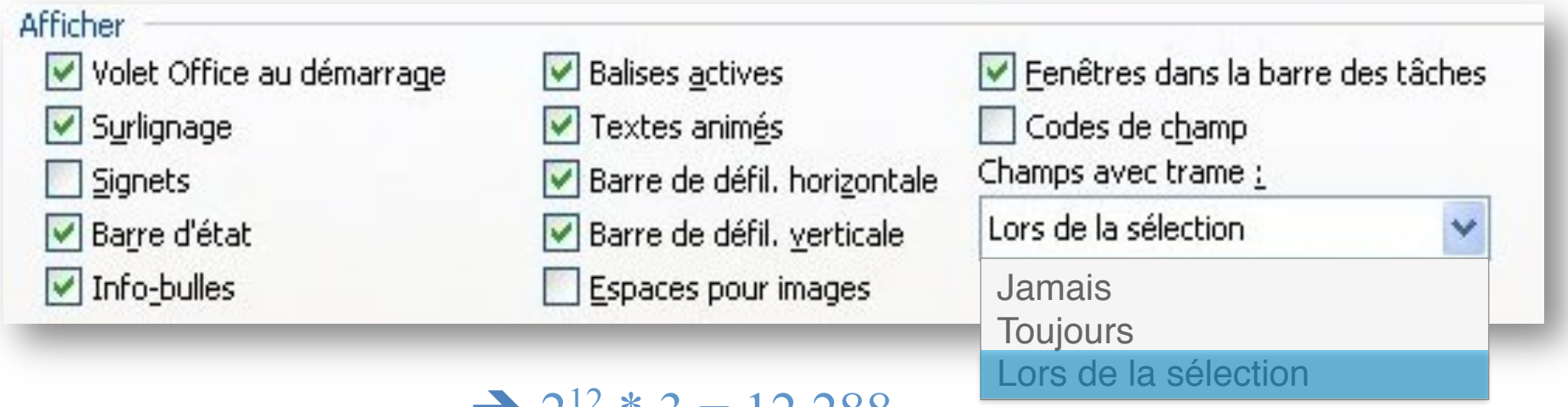
- Méthode de test fonctionnel très productive :
  - le comportement du programme aux valeurs limites est souvent pas ou insuffisamment examiné
- Couvre l'ensemble des types de test
- Inconvénient : caractère parfois intuitif ou subjectif de la notion de limite

  ⇒ Difficulté pour caractériser la couverture de test.

Méthode pour le test combinatoire

- Les combinaisons de valeurs de domaines d'entrée donne lieu à explosion combinatoire
- Exemple : Options d'une boite de dialogue MS Word :

  On a 12 cases à cocher et un menu déroulant pouvant prendre 3 valeurs.



➔ $2^{12} * 3 = 12\,288$

Test combinatoire : Pair-wise

- Tester un fragment des combinaisons de valeurs qui garantissent que chaque combinaison de 2 variables est testé
- Exemple : 4 variables avec 3 valeurs possibles chacune

OS	Réseau	Imprimante	Format
Windows	Cable	Laser	Texte
Linux	Wifi	Encre Liquide	Image
Mac Os	Bluetooth	Encre Solide	Mixe

Toutes les combinaisons : 81

Toutes les paires : 9

Pairwise – Exemple

- 9 cas de test : chaque combinaison de 2 valeurs est testée

N°	OS	Réseau	Imprimante	Application
Cas 1	Windows	Bluetooth	Laser	Texte
Cas 2	Windows	Cable	Jet d'encre	Image
Cas 3	Windows	Wifi	Ruban	Mixe
Cas 4	Mac OS	Bluetooth	Jet d'encre	Mixe
Cas 5	Mac OS	Cable	Ruban	Texte
Cas 6	Mac OS	Wifi	Laser	Image
Cas 7	Linux	Bluetooth	Ruban	Image
Cas 8	Linux	Cable	Laser	Mixe
Cas 9	Linux	Wifi	Jet d'encre	Texte

L'idée sous-jacente : la majorité des fautes sont détectées par des combinaisons de 2 valeurs de variables

Exercice – n-wise

On désire tester la procédure de prise de commande d'un restaurant. Pour cela, la procédure prend en paramètres 4 variables représentant respectivement une entrée, un plat, un dessert et un café. Chacune possède les valeurs possibles suivantes :

Entrée	Plat	Dessert	Café
Pâté	Grillade	Fromage blanc	Non
Salade	Poisson	Fruit	Oui
Soupe	Volaille	Gâteau	
		Glace	

Calculer un ensemble de données de tests avec un approche pairwise ?

Exercice – n-wise correction

N° Test	Entrée	Dessert	Plat	Café
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Exercice – n-wise correction

Exercice – n-wise correction

Exercice – n-wise correction

Exercice – n-wise correction

Exercice – n-wise correction

Test combinatoire

- L'approche Pair-wise se décline avec des triplets, des quadruplets, …. mais le nombre de tests augmente très vite

- Différents outils permettent de calculer les combinaisons en Pairwise (ou n-valeurs) :
  - http://www.pairwise.org/default.html
  - Prise en charge des exclusions entre valeurs des domaines et des combinaison déjà testée

- Problème du Pair-wise :
  - le choix de la combinaison de valeurs n'est peut-être pas celle qui détecte le bug …
  - Le résultat attendu de chaque test doit être fourni manuellement

Test aléatoire ou statistique

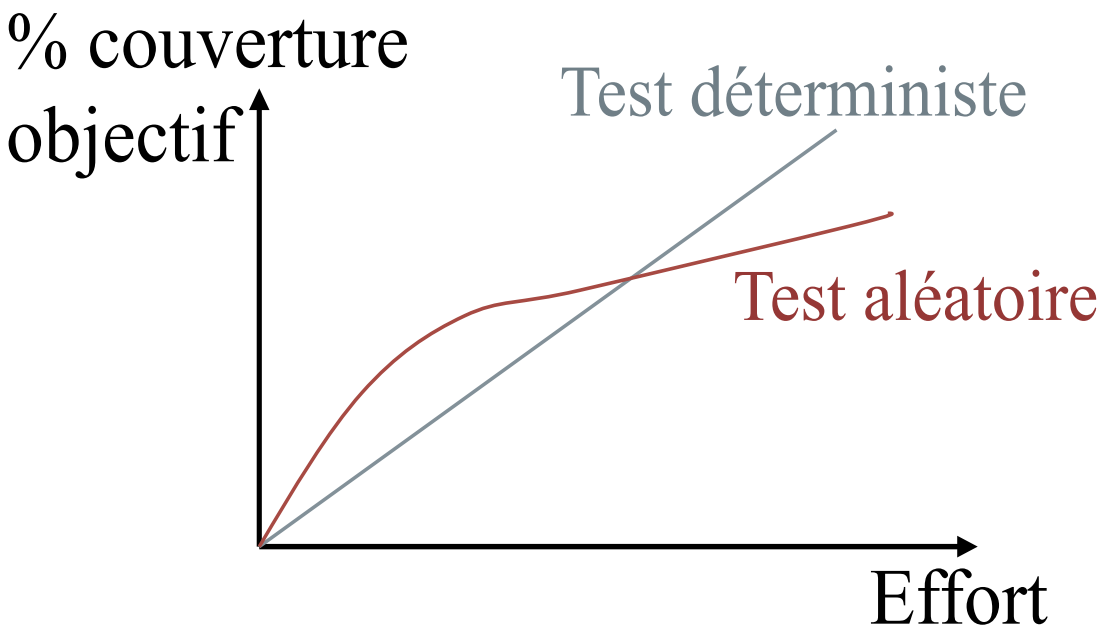
- Principe : utilisation d'une fonction de calcul pour sélectionner les données de test :
  - fonction aléatoire : choix aléatoire dans le domaine de la donnée d'entrée,
  - utilisation d'une loi statistique sur le domaine.
- Exemples :
  - Echantillonnage de 5 en 5 pour une donnée d'entrée représentant une distance,
  - Utilisation d'une loi de Gauss pour une donnée représentant la taille des individus,
  - ...

Evaluation du test aléatoire

- Intérêts de l'approche statistique :
  - facilement automatisable pour la sélection des cas de test, (plus difficile pour le résultat attendu)
  - objectivité des données de test.
- Inconvénients :
  - fonctionnement en aveugle,
  - difficultés de produire des comportements très spécifique

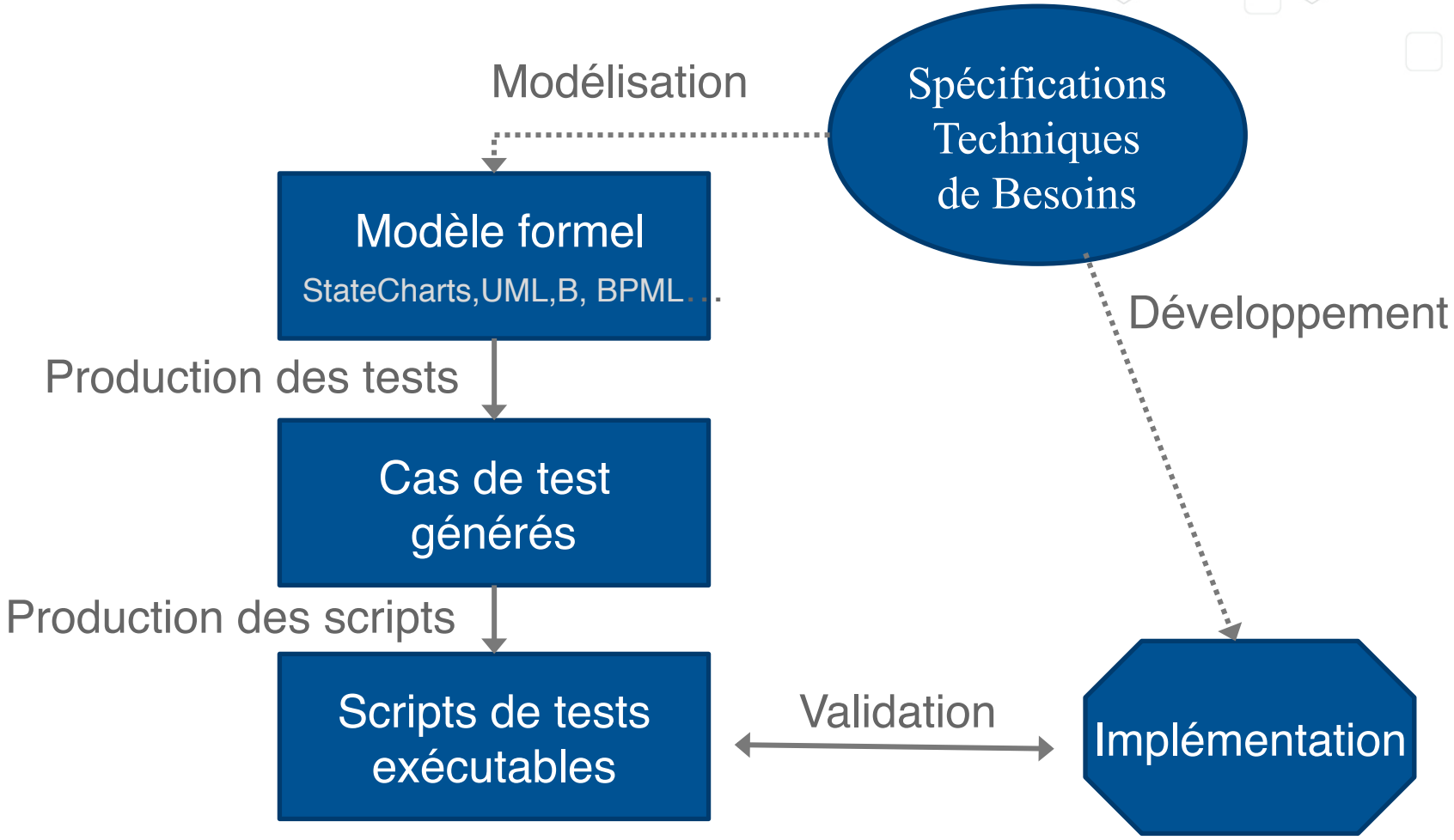
```
public int methodeImprobable(int x, int y) {
        if (x == 600 && y == 500) then
                thrown new Exception("Bonne chance pour me trouver");
        return (x+y);
}
```

Productivité du test aléatoire ou statistique



Les études montrent que le test statistique permet d'atteindre rapidement 50 % de l'objectif de test mais qu'il a tendance à plafonner ensuite.

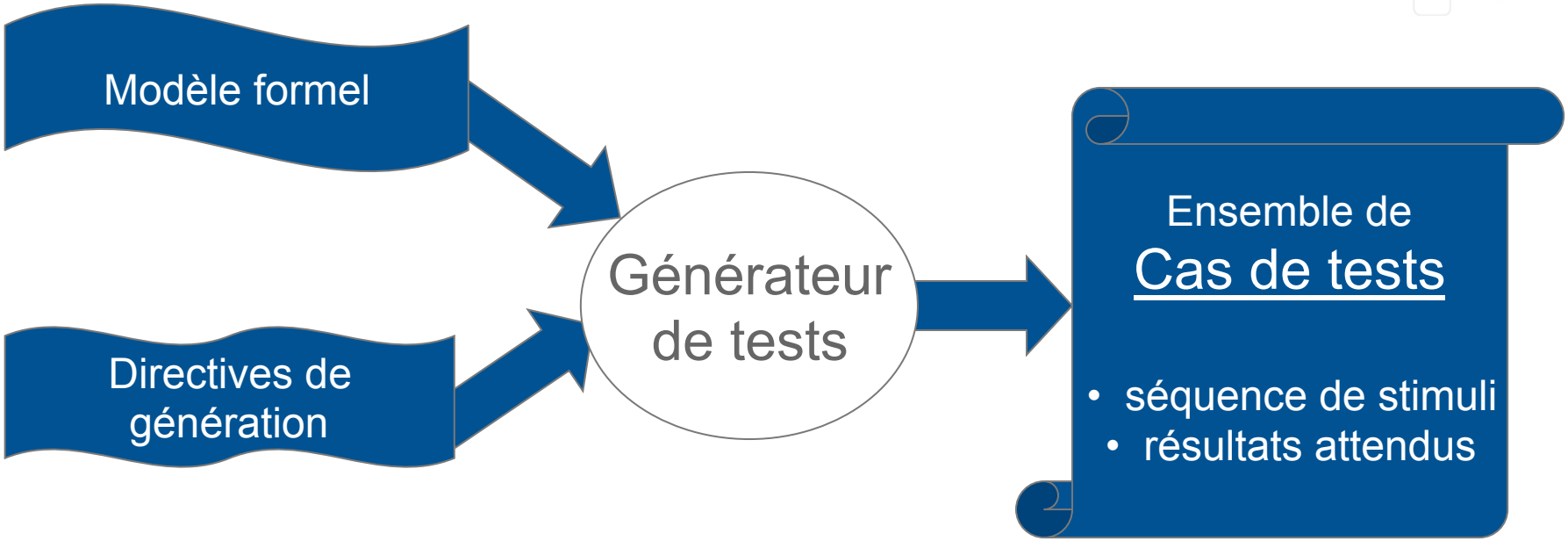
Test à partir de modèles



Langages de modélisation

- De nombreux paradigmes
  - Systèmes de transitions (Etats / Transitions / Evénements)
    - Flow Charts
    - Data Flow Diagrams
    - Diagramme d'état
  - Diagrammes objet & association (Entité-Relation, héritage, …)
    - Diagramme de classe (UML)
    - Entité-Relation
  - Représentation Pré-Post conditions
    - OCL (UML)
    - Machine Abstraite B
- Représentation :
    - Graphique (plus « intuitive »)
    - Textuelle (plus précise)

Model Based Test Generation



- Directives de génération (définition de scénarii de test) :
  - Critères de couverture du modèle
  - Critères de sélection sur le modèle

Synthèse

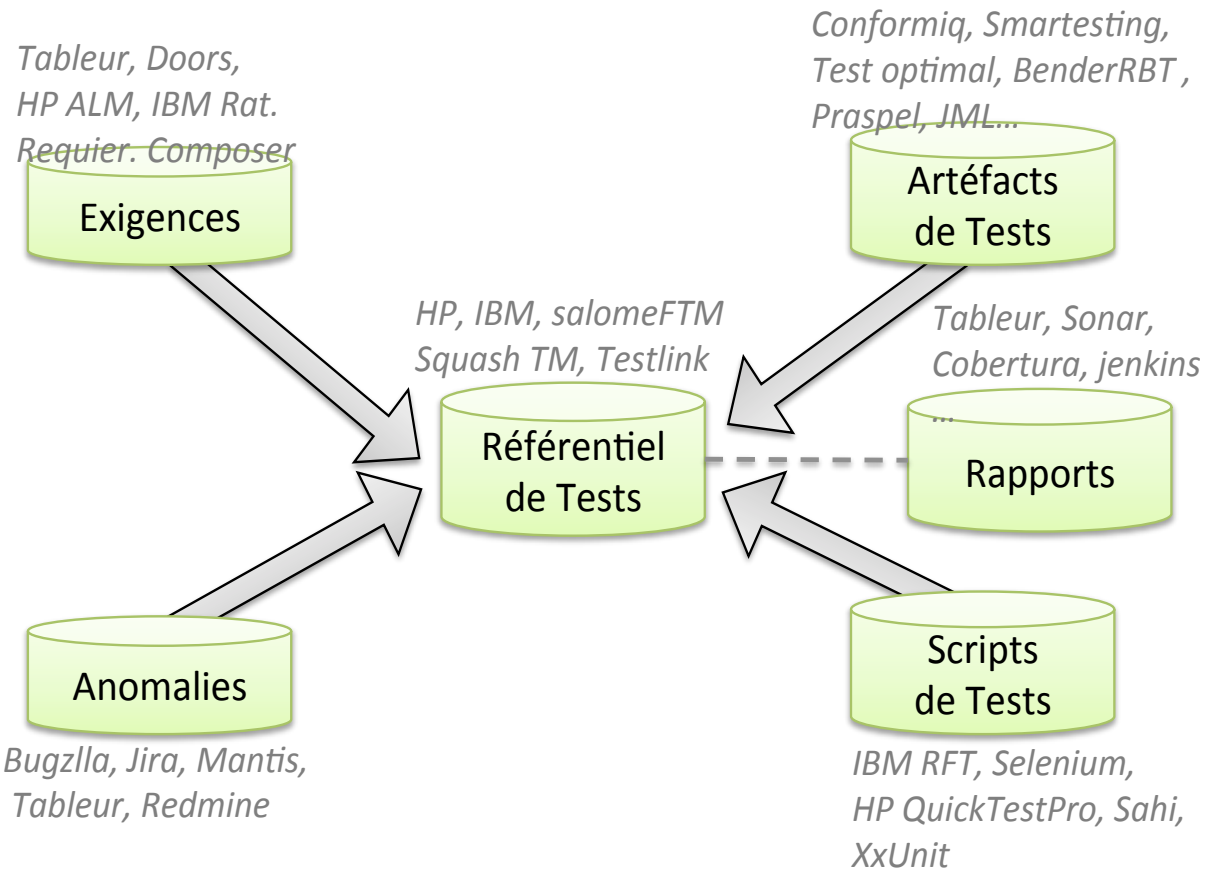
- La production de test s'appuie (généralement) sur une analyse du programme (test structurel) ou de sa spécification (test fonctionnel)

- Différentes stratégies permettent de sélectionner des données de test pertinentes.

- Ces stratégies ne sont que des heuristiques !

Elles ne fournissent aucune garantie de sélectionner la valeur qui révélera les erreurs du programme.

Univers outillé



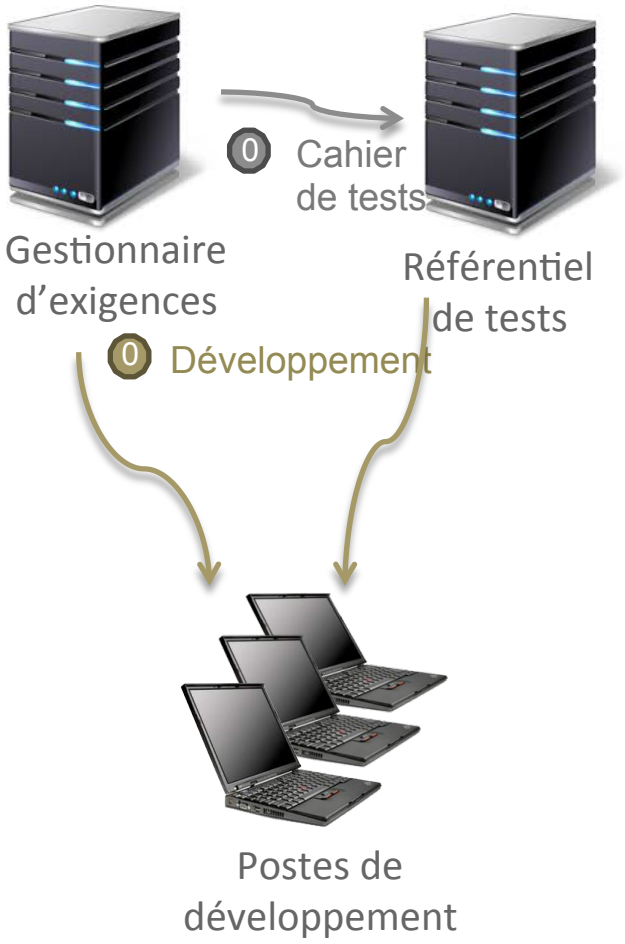
5. Automatisation de la gestion des tests

Gestion de la couverture des exigences par les tests.
Démarche de mise au point : organisation des suites de tests et création des cas.
Préparation à l'automatisation.
Construction de la population de test.
Mise au point et vérification des tests (Revue)
Exécution, enregistrement des anomalies. Notion de rapport d'incident d'après l'IEEE.
Gestionnaires d'anomalies. Automatisation de la création des anomalies.
Analyse de résultats d'exécution de tests. Consolidation des tests.

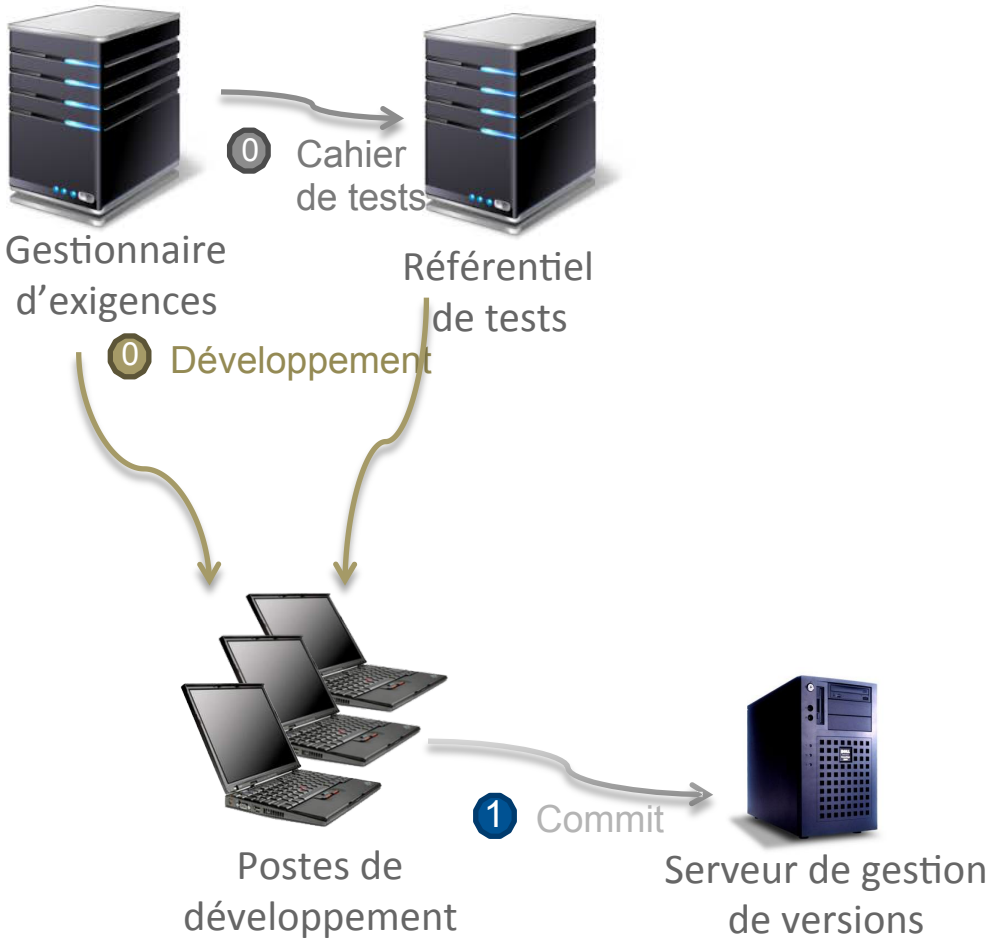
Etape -1 : Recueil des exigences

Gestionnaire d'exigences

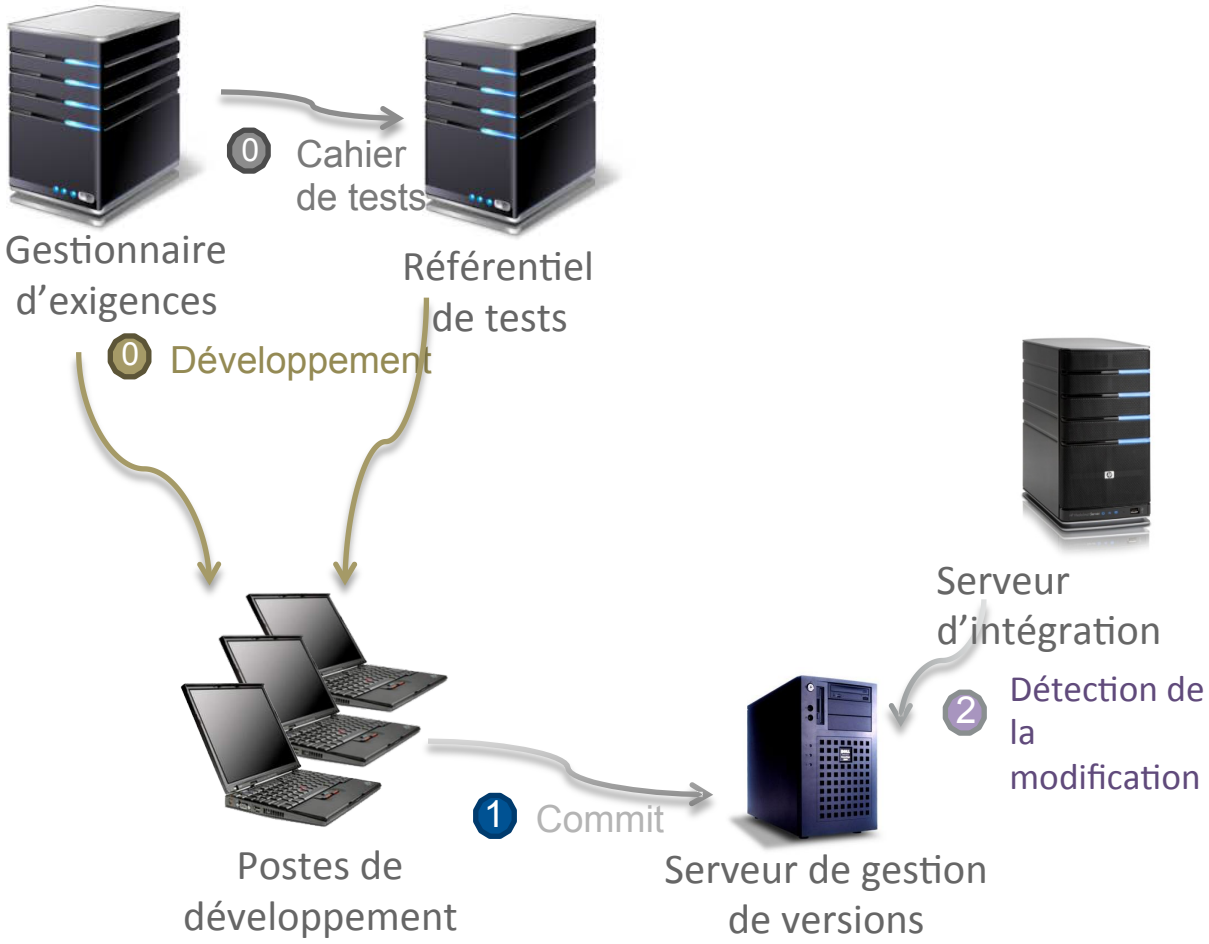
Etape 0 : utilisation des exigences



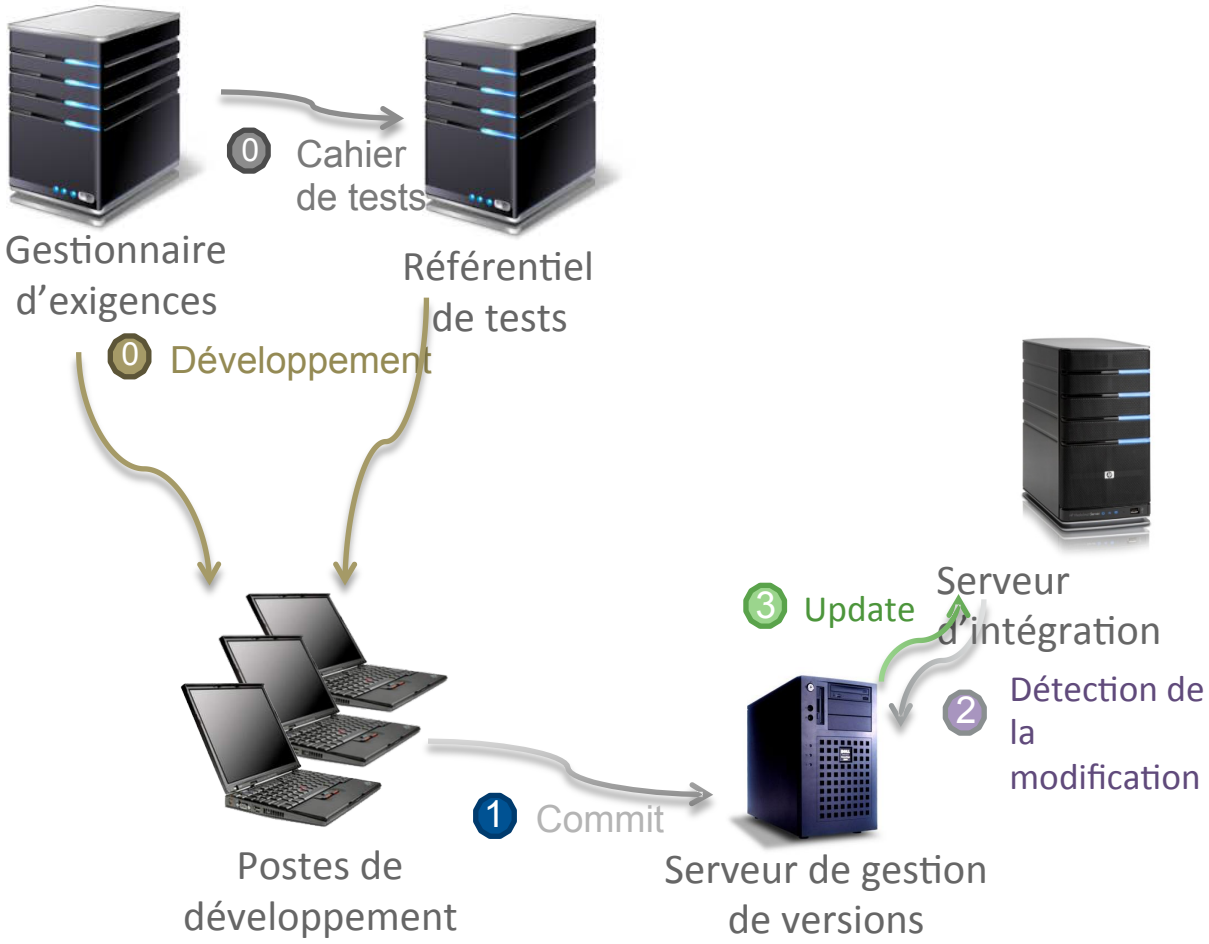
Etape 1 : mise à jour des sources



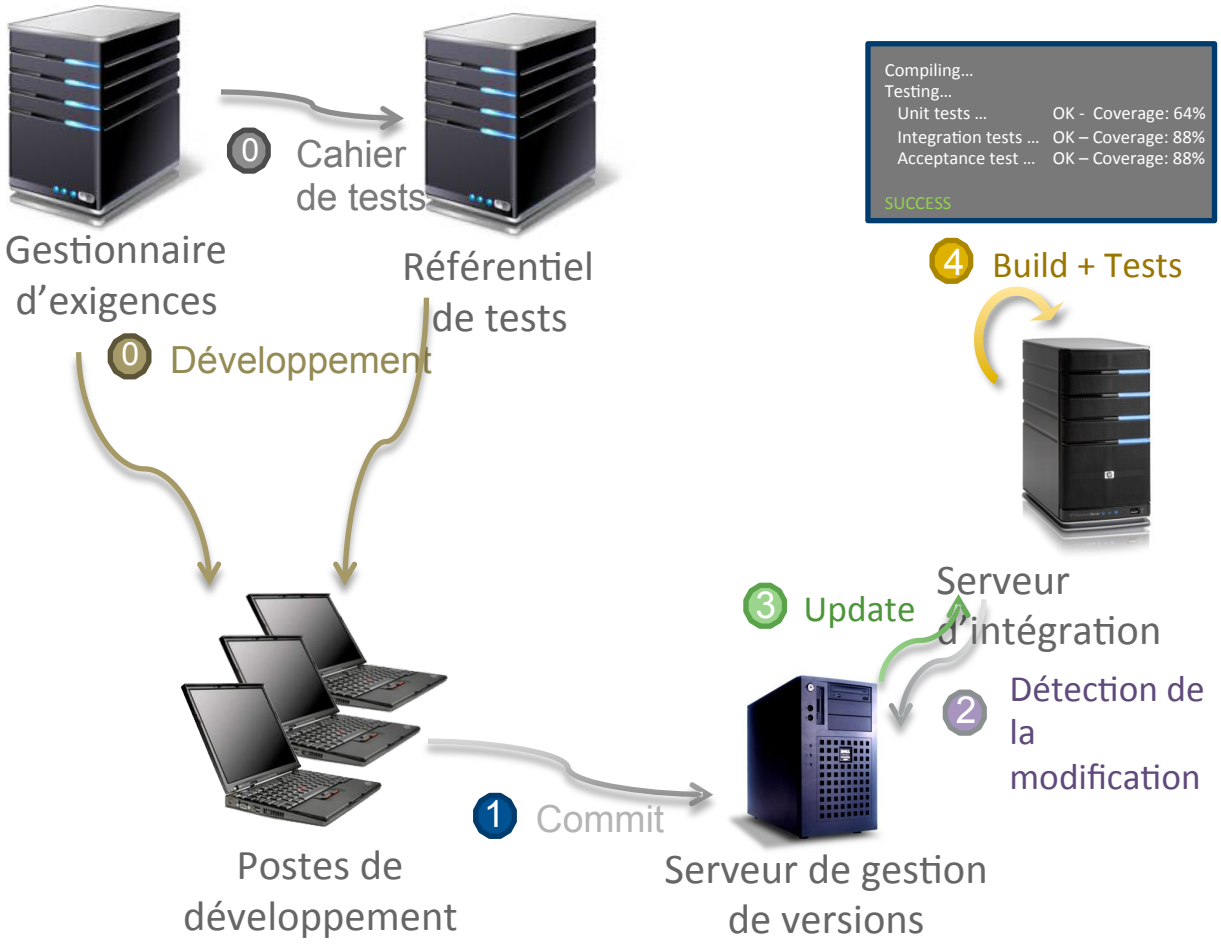
Etape 2 : détection de la mise à jour



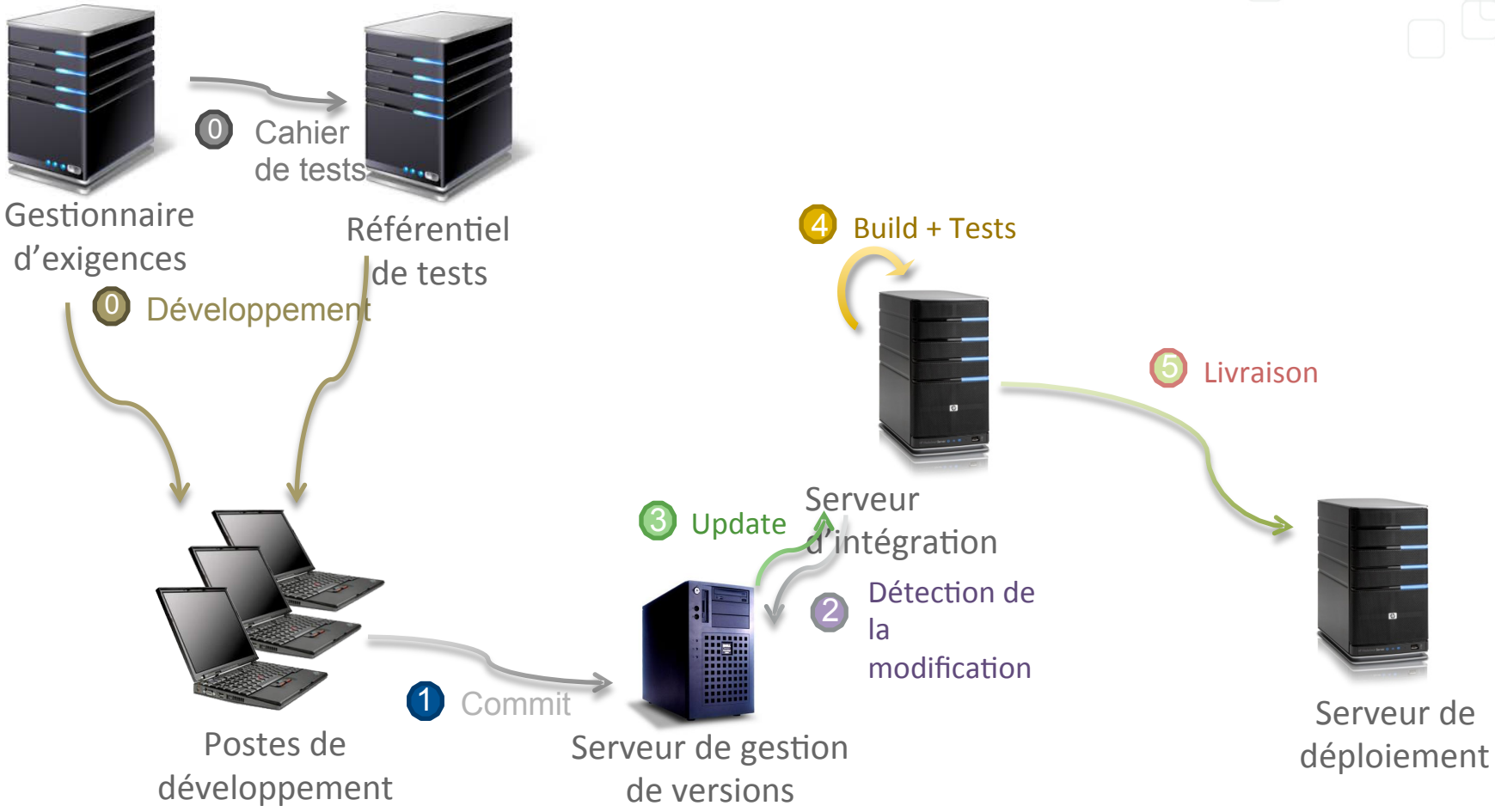
Etape 3 : récupération des sources



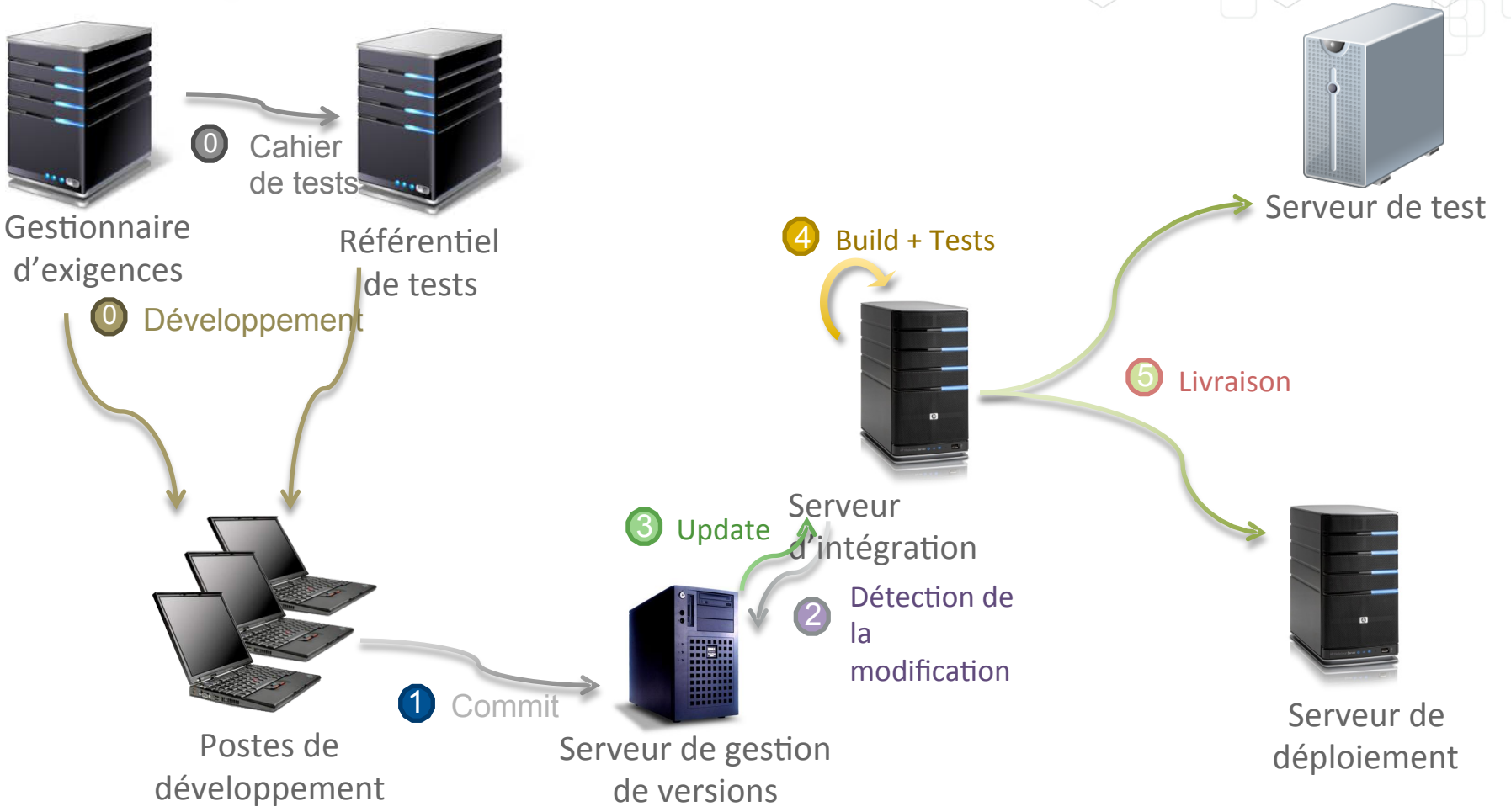
Etape 4 : construction et tests realease



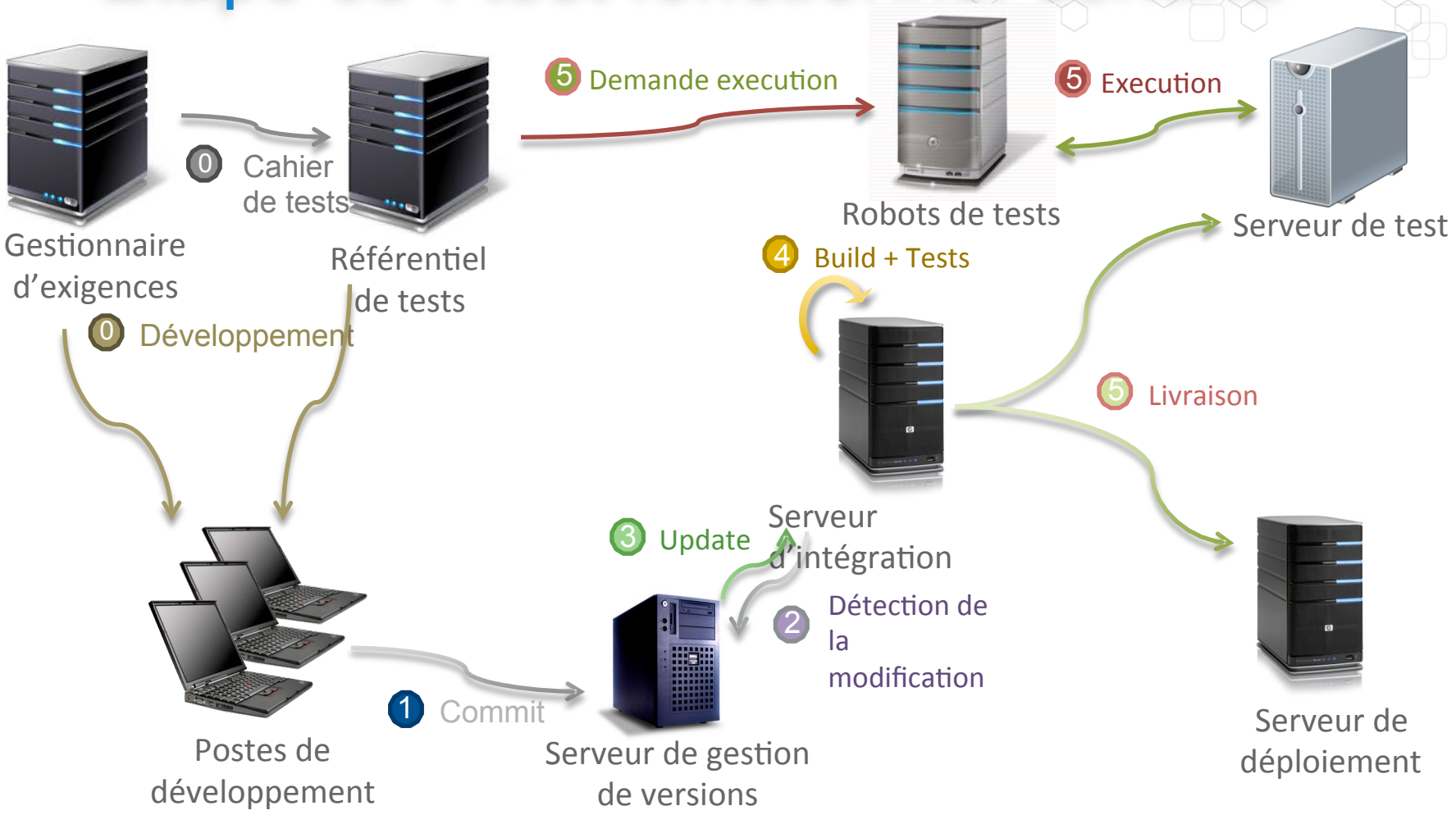
Etape 5 : livraison release



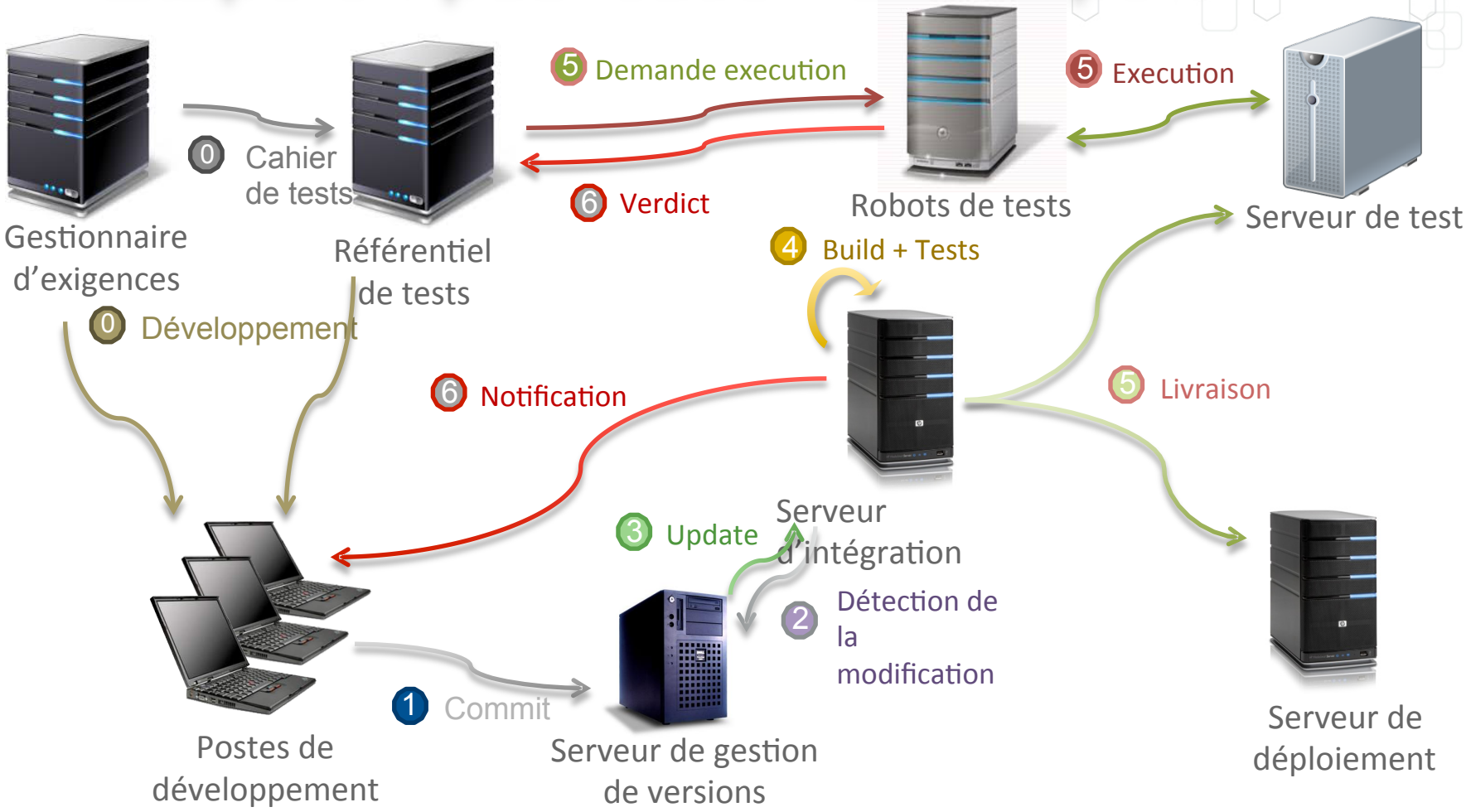
Etape 5a : livraison continue



Etape 5b : test fonctionnel continu



Etape 6 : publication du rapport



Outils

- Gestions des exigences
- Référentiel de tests
- Gestionnaire des scripts de tests
- Robot d'exécutions
- Rapport d'exécutions

Rational Quality Manager

DOORS, Jazz

MICRO FOCUS

Caliber – Silk Test

squash TM

squash TA

Visual Studio

Test Professional 2015

Test Link

jama

hp

QTP

Quality Center

Fin de la partie Cours

Merci pour votre participation et bonne continuation