Environnement de test

- Intégration continue, CI/CD

plan

- Tests
  - Tests (unitaires, de régression, fonctionnel, de conformité, de sécurité, de robustesse, etc.)
  - Tests en isolation
  - Patron Given When Then
- Tests et gestion de projet
  - CMMI, TMMI,
  - Cycle de vie, agilité
  - spécifications
- Test et environnements
- Tester avec de l'IA ?

Environnement de test, Test basé modèle

Intégration continue, CI/CD

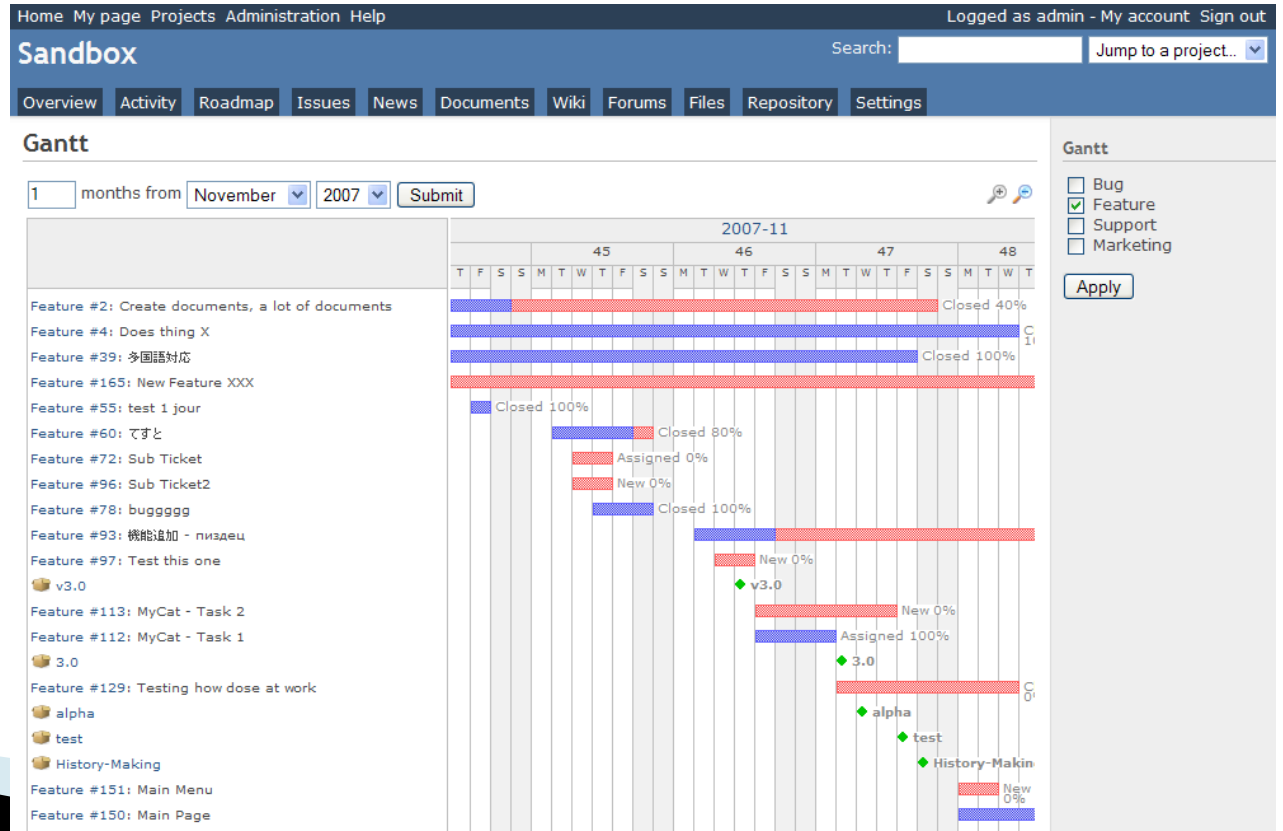
MbT, vérification

Environnement de test

- Besoin de monter tout un environnement pour tests unitaires, d'intégration, etc.

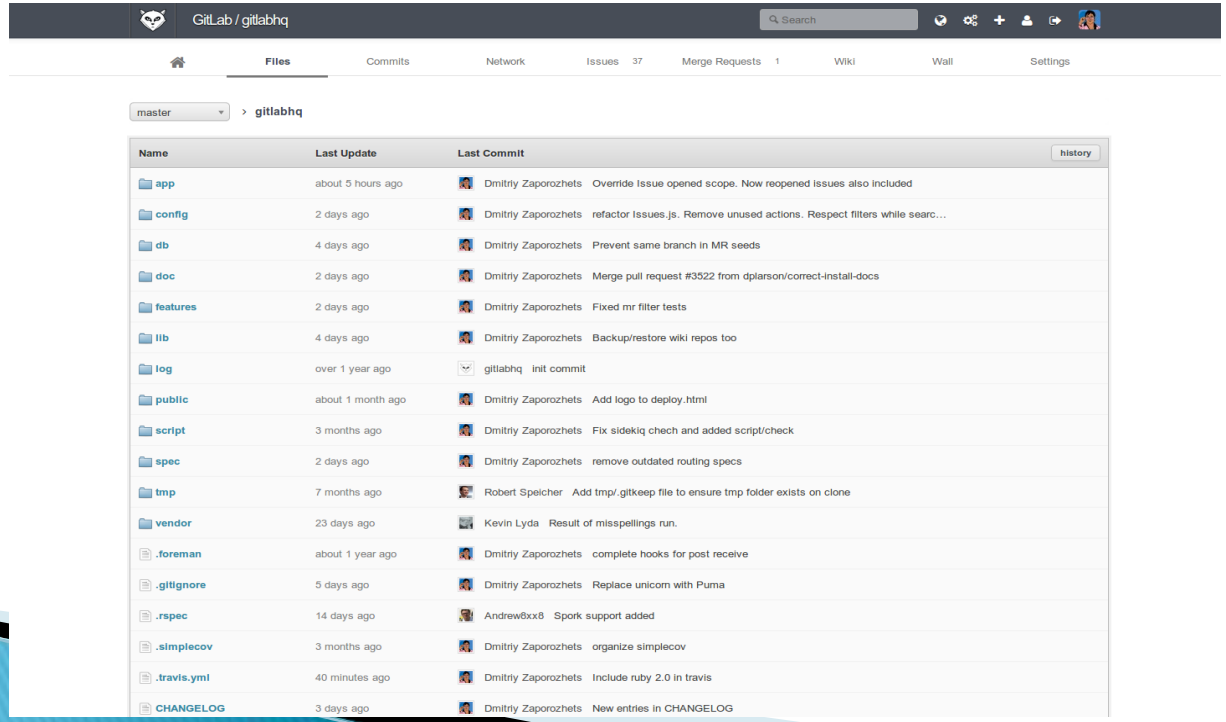
1. Environnement de gestion de projet (redmine ou gitlab mais moins complet)

Redmine : outil (app web) de gestion de projet (gantt, svn, etc.)



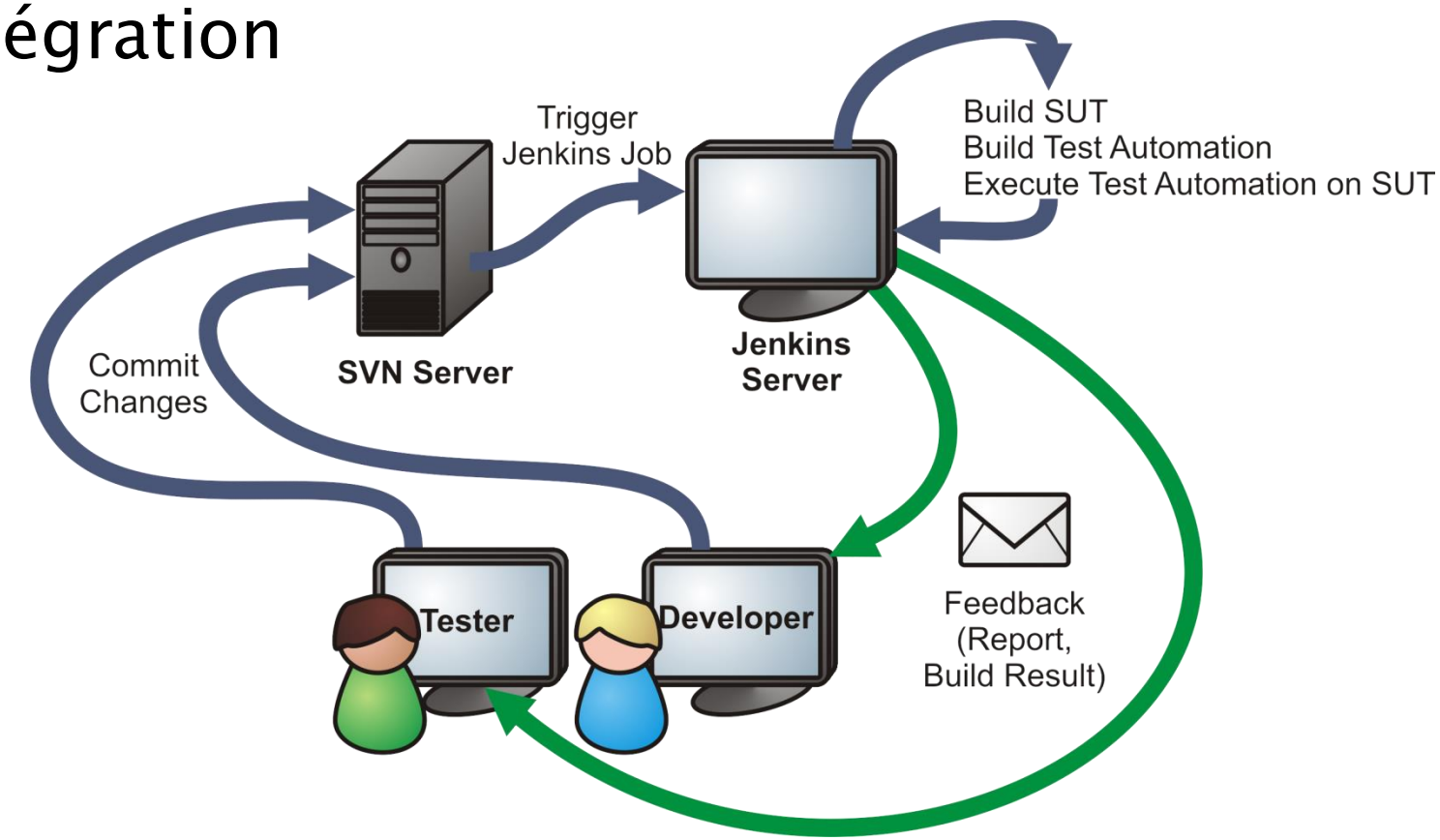
Environnement de test

- Besoin de monter tout un environnement pour tests unitaires, d'intégration, etc.
2. Travail collaboratif, stockage: svn, GitLab, GitHub



Environnement de test

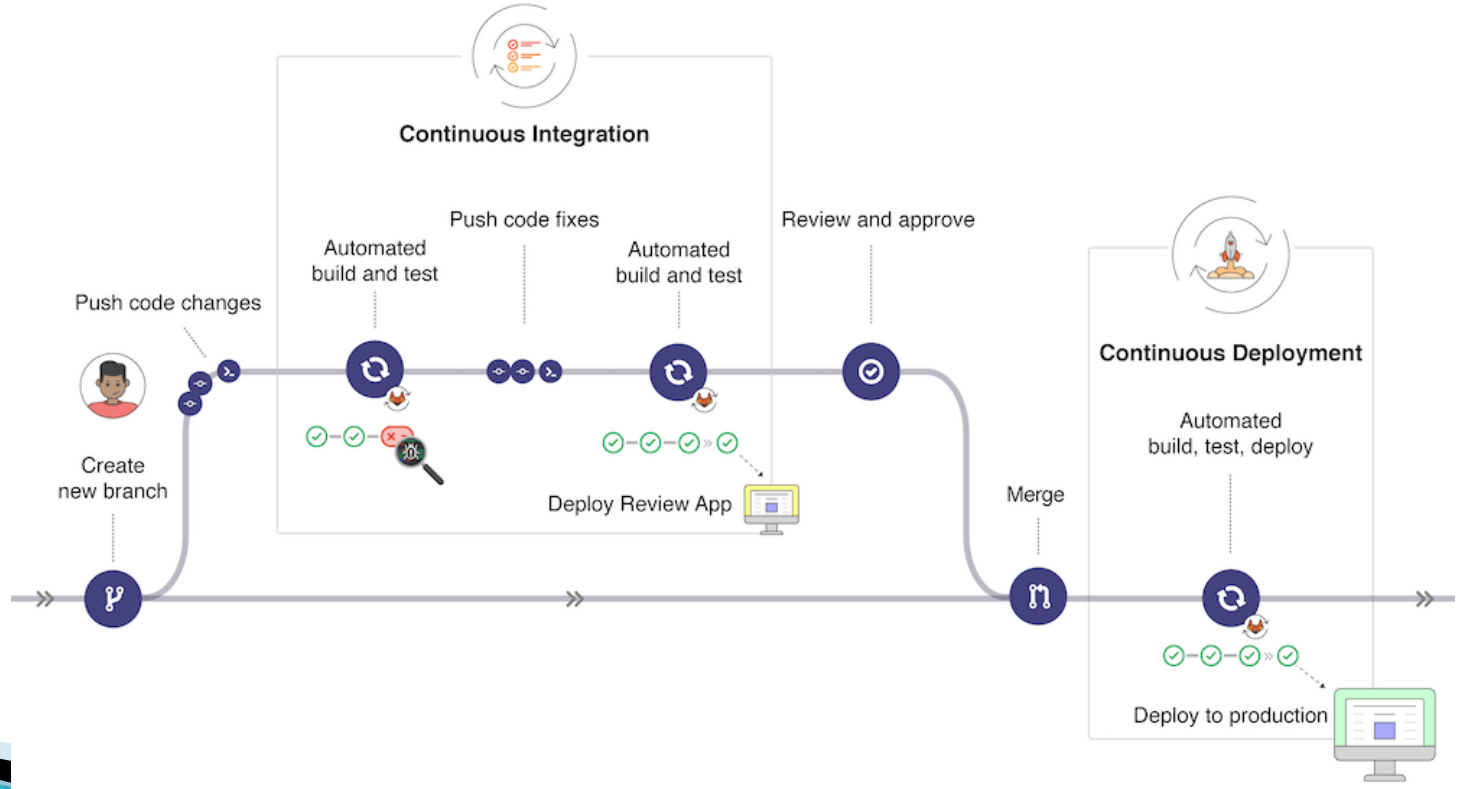
3. Serveur d'intégration
- Jenkins:



+ plugin de test qualité sur le code (redondance, etc.) mais moins puissant que Sonar

Environnement de test

3. Serveur d'intégration
- Gitlab CI/CD



Environnement de test

4. IDE de test (intégré à l'ide de dev ou pas)

   - Infinitest,
   - Eclipse, Netbeans, IntelliJ, Phpstorm, etc.
   - Visual studio,
   - Selenium ide,
   - Archilian,

Mais aussi…

- Qualité du code écrit, sécurité, etc.
  - Ex:Format, boucles, commentaires,…
  - Jdepend, Ndepend (automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to manage package dependencies effectively),
  - Jtest(parasoft, test unitaire, analyse statique de code)
  - Jcrasher (test robustesse automatique en Java)
  - HPQuality Center (test d'intégration, de regression),
  - Profiler (Netbeans fuites mémoire)
  - Testlink (test d'integration, plans de test comme HPQuality c.)

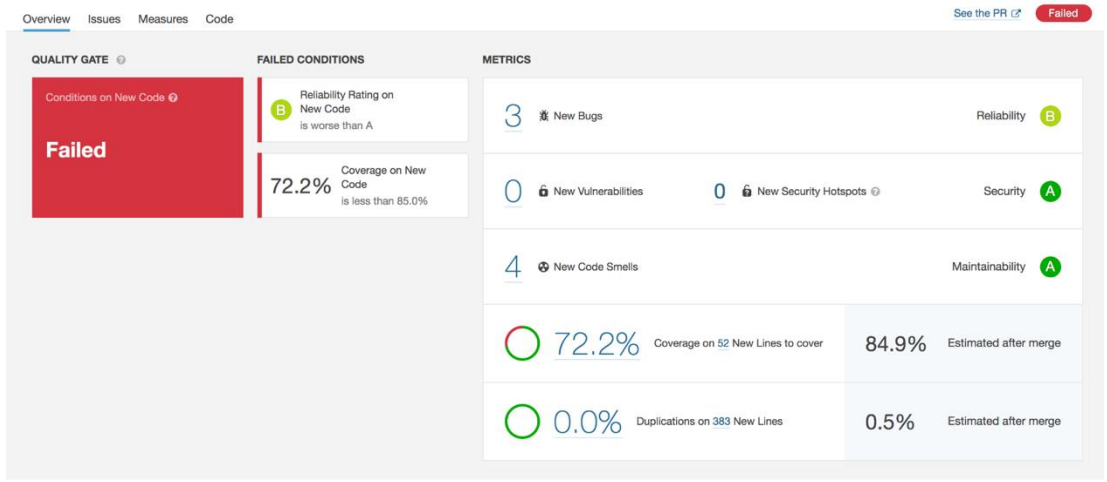
Outils

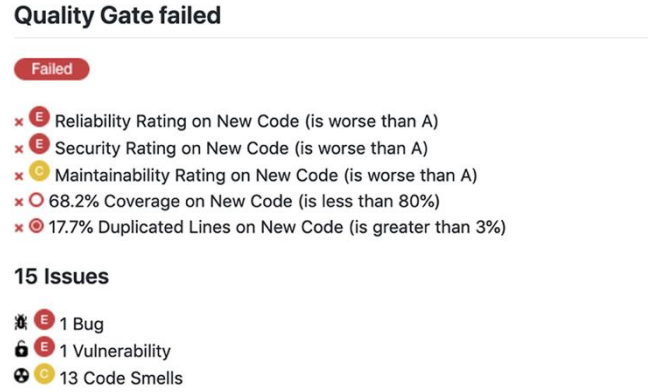
- Outils de contrôle qualité: SonarQube, SonarLint (plugin IDE)
  - SonarQube http://www.sonarqube.org

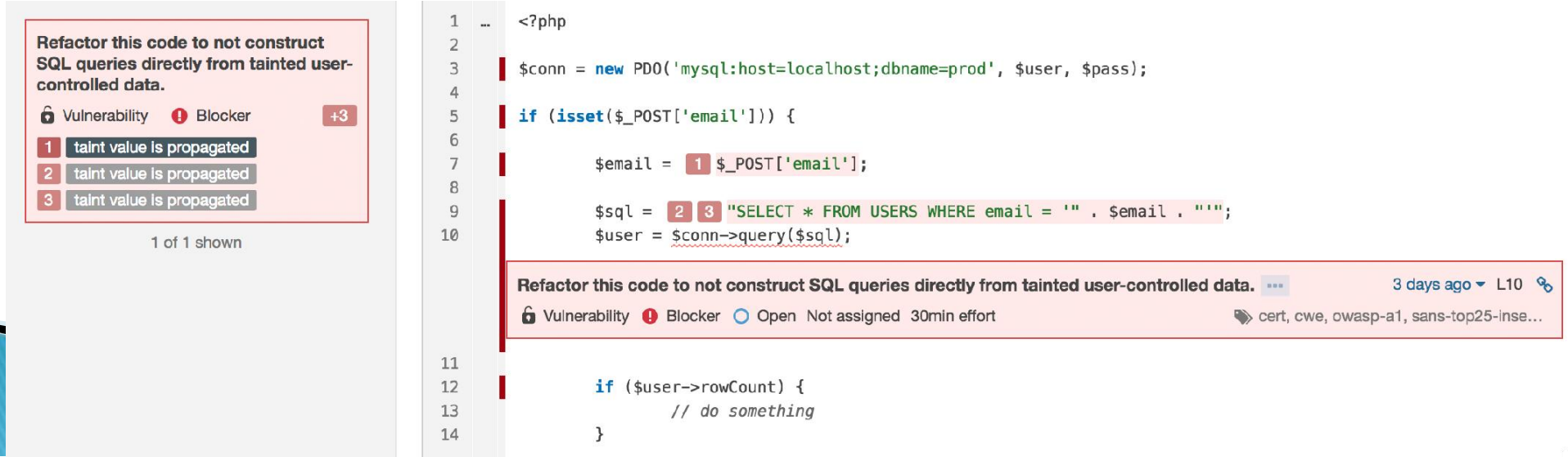
- Tableau de bord complet des différents projets suivis.
- Détection rapide du code à risque.
- Mesures quantitatives : nombre de classes, duplication de code, etc...
- Mesures qualitatives : couverture et taux de réussite des tests, complexité du code, respect des règles de codage...
- Sécurité (détection mdp en dur, sql injections), basique toutefois
- Historiques des statistiques, pour en voir l'évolution au cours du temps.
- Support de plus de 600 règles de qualité (suivant la norme ISO 9126-3).
- Gestion de profils pour les règles de codage.
- Visualisation du code source, surlignant les violations des règles de codage qui s'y trouvent.
- Fonction "Time machine" permettant de comparer plusieurs versions d'une même application.
- Identification des points faibles d'un projet.
- Support des plugins.

Outils

- Sonarqube(http://www.sonarqube.org/)

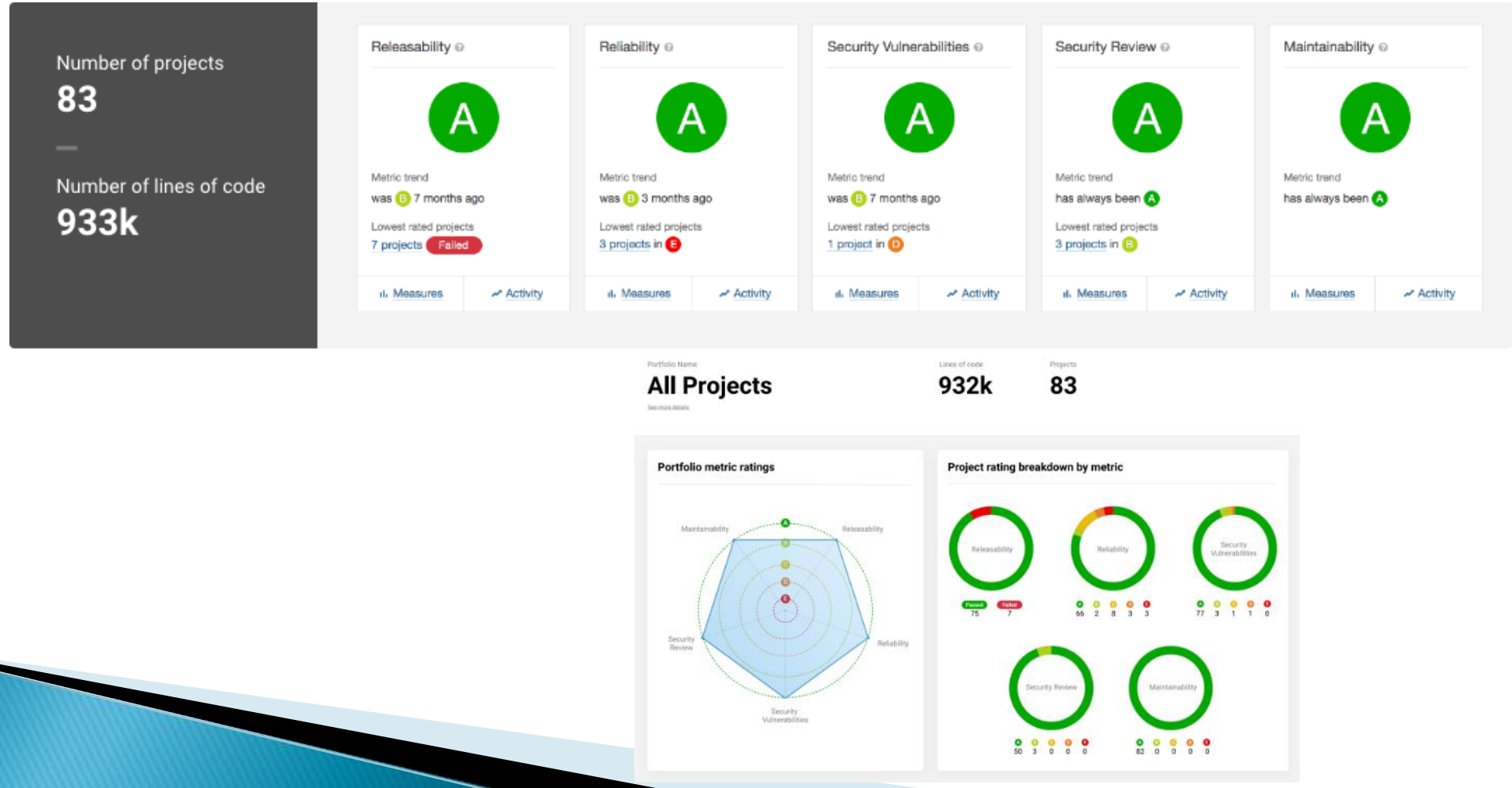






Outils

- Sonarqube(http://www.sonarqube.org/)



Testabilité

Testabilité

- Degré pour lequel un système, une application est testable
- Si système pas ou peu testable,
  - alors inutile de faire des tests ?
  - Ou ?

- La testabilité d'un système : ensemble de critères évaluant le coût du test, l'accessibilité de ses composants, la capacité à relever ses fautes
  - ⇨Degrés (facteurs) pour mesurer la testabilité

Testabilité

La **qualité de test d'un système** est une quantication évaluant :

- le coût du test du système,
- l'accessibilité des composantes ou des parties de comportements du système,
- la capacité du système à relever ses fautes d'une manière non ambiguë, grâce aux stimulations (contrôles) et aux observations,
- la facilité d'établir des critères de test du système et de déterminer si, par l'application du test, ces critères sont satisfaits

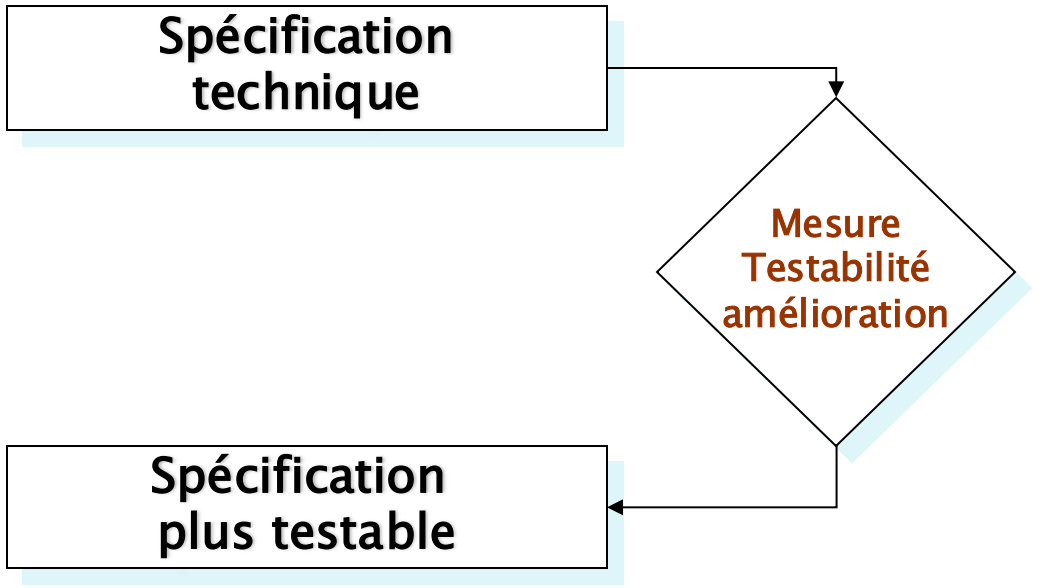
Testabilité

Le système est **testable**, (la qualité de test du système est optimale) si :

- le coût du test du système est le meilleur,
- toutes les fautes du système peuvent être relevées par stimulation et observation, les critères de test peuvent être facilement définis et peuvent être vérifiés.

[ss01]

Testabilité



Calcul des facteurs de testabilité
Transformation en spécification testable

Testabilité

Testabilité :

- Plusieurs degrés :

- Observabilité,
- Contrôlabilité (inclus déterminisme),
- Capacité d'isolation (test en isolation),
  - Moindre mesure plus simple à tester si **séparation des responsabilités, hé**terogeneité

Testabilité

- Observabilité [Fre91] Un système est dit observable si pour toute entrée appliquée au système, une sortie différente est observée

- Contrôlabilité [Fre91] Un système est dit contrôlable si pour 1 sortie observé, il existe une entrée, qui, une fois appliquée au système, force cette observation.

Test basé modèle

introduction

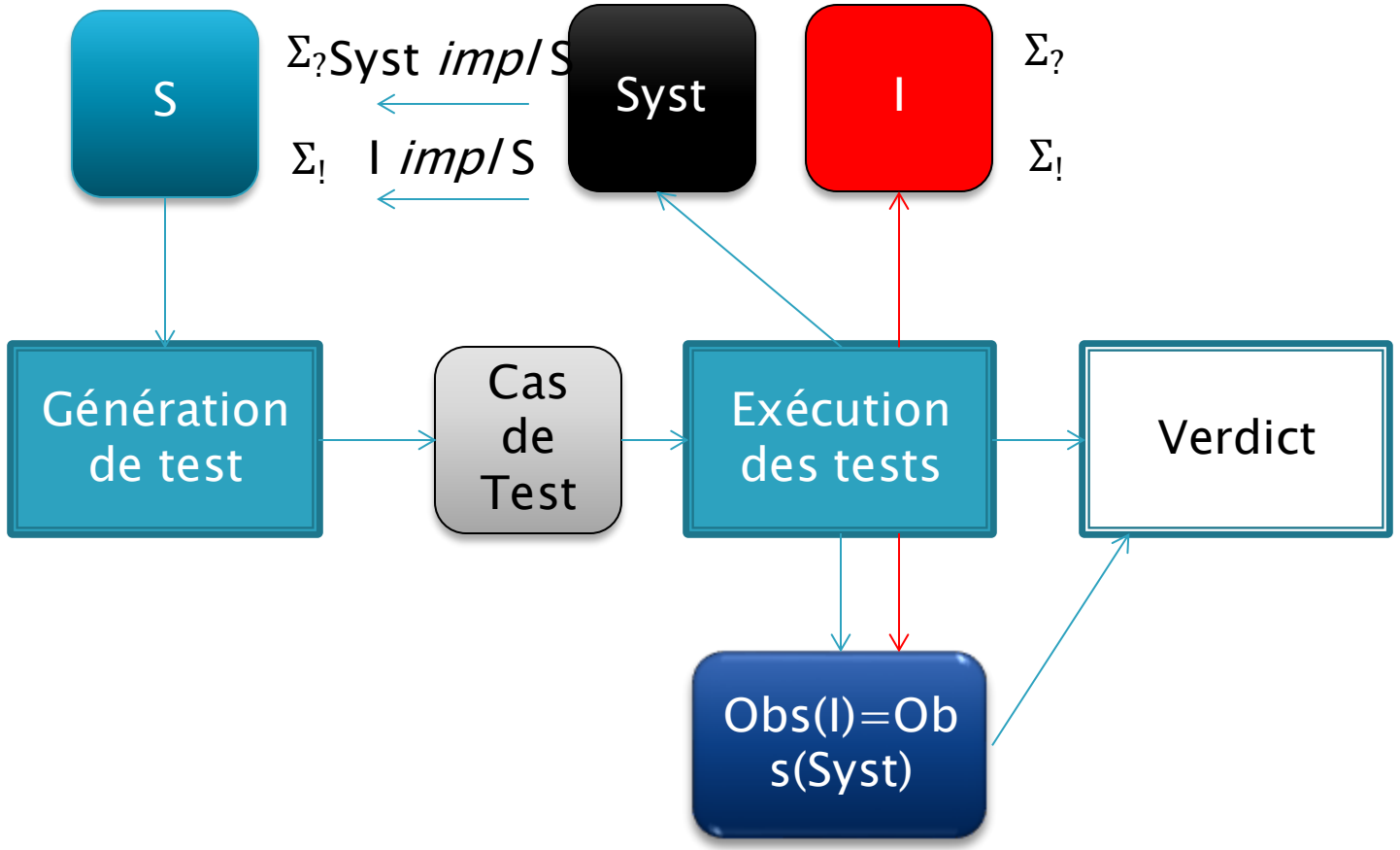
Test basé modèle

- Motivation: automatisation de la synthèse des tests depuis une spécification formelle, formalisation de relations de test, etc.
- Fortement recommandé au moins pour les systèmes critiques (santé, transport, énergie, etc.)
- Ingrédients:
  - spécification, implémantation sous test en boîte noire (IUT),
  - Relation de test *impl* (formalise IUT implante S),
  - Cas de test,
  - testeurs
- Recette ?

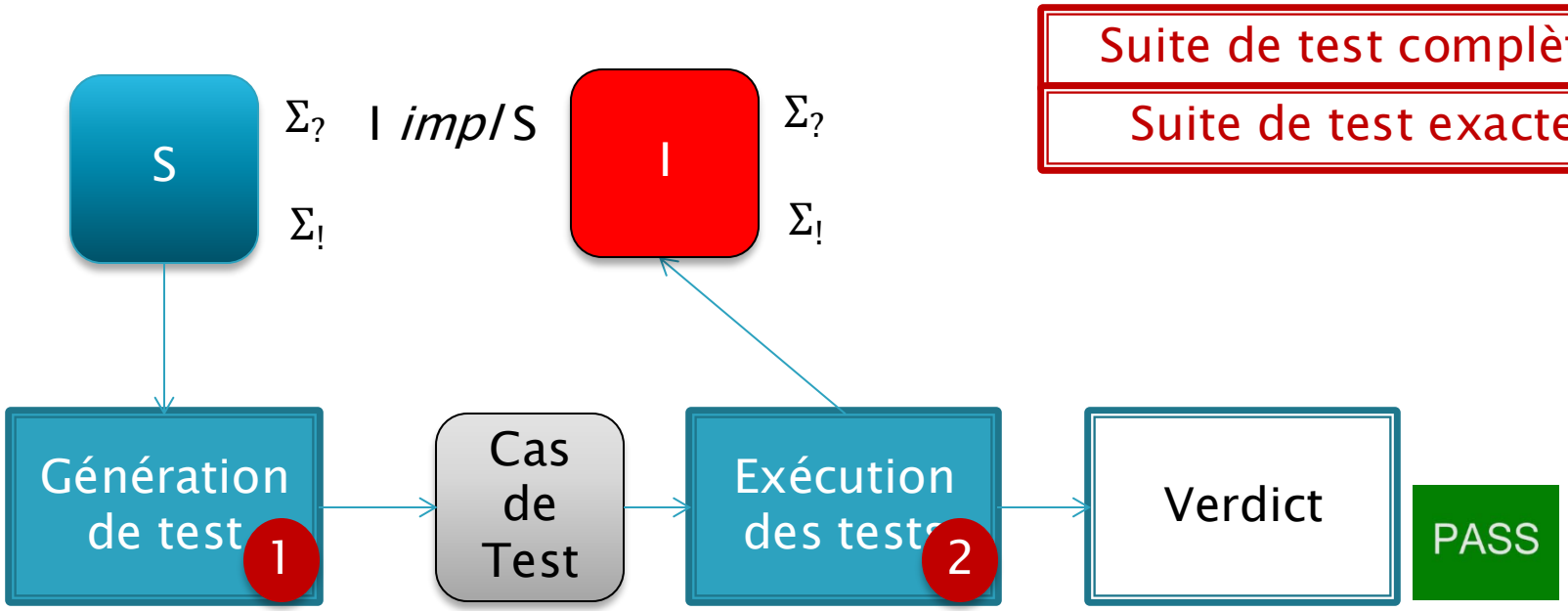
Test basé modèle

- L'implémantation est souvent en boite noire
- L'implémantation est inconnue mais besoin de la décrire par un modèle pour pouvoir définir des relations entre la spécification (modèle) et des implémantations
- => on suppose que l'implémantation peut être modélisée par un modèle I qui n'est pas connu et qui retourne les mêmes observations que l'implémantation.

Test basé modèle



Test basé modèle

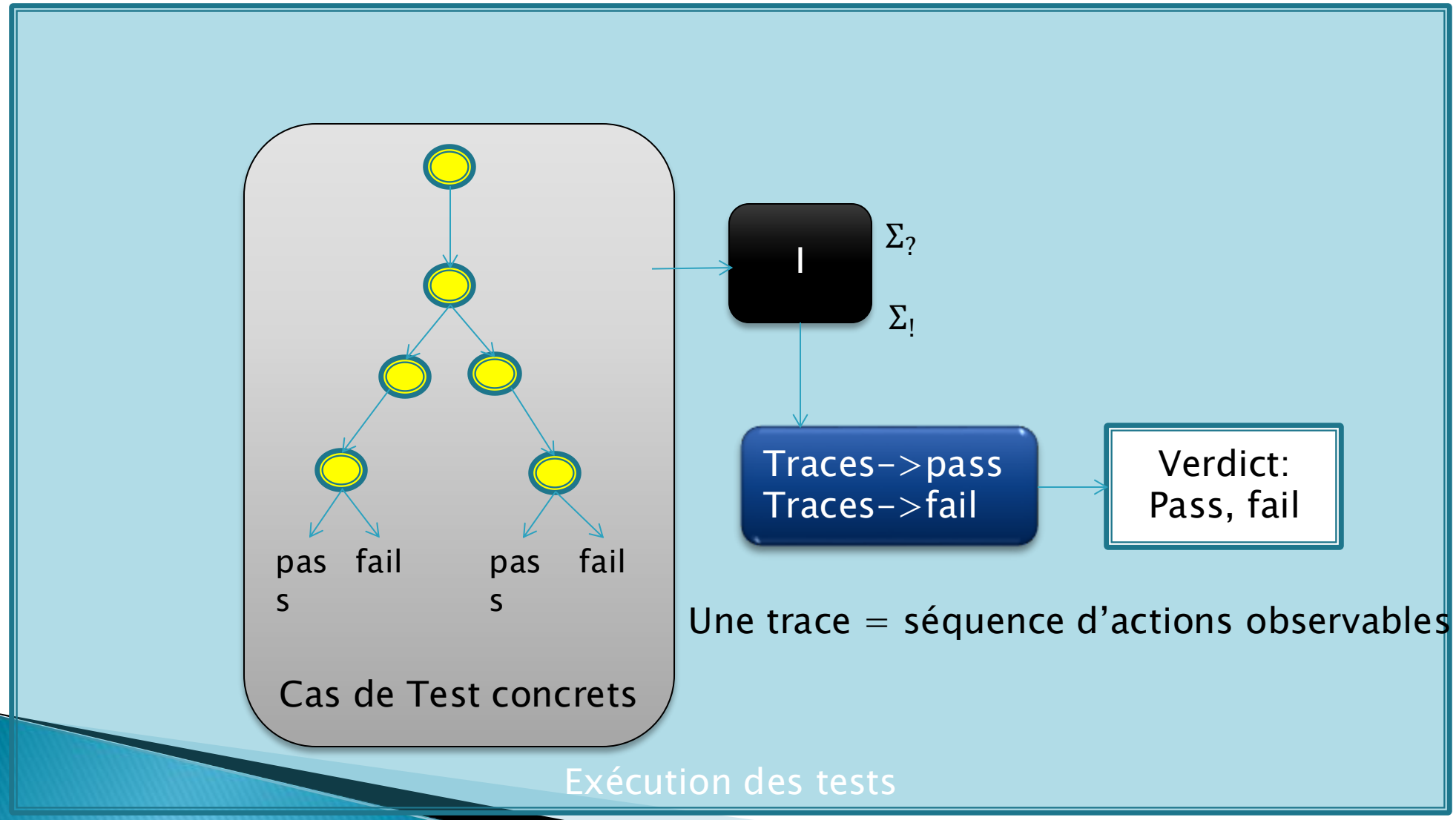


Suite de test complète

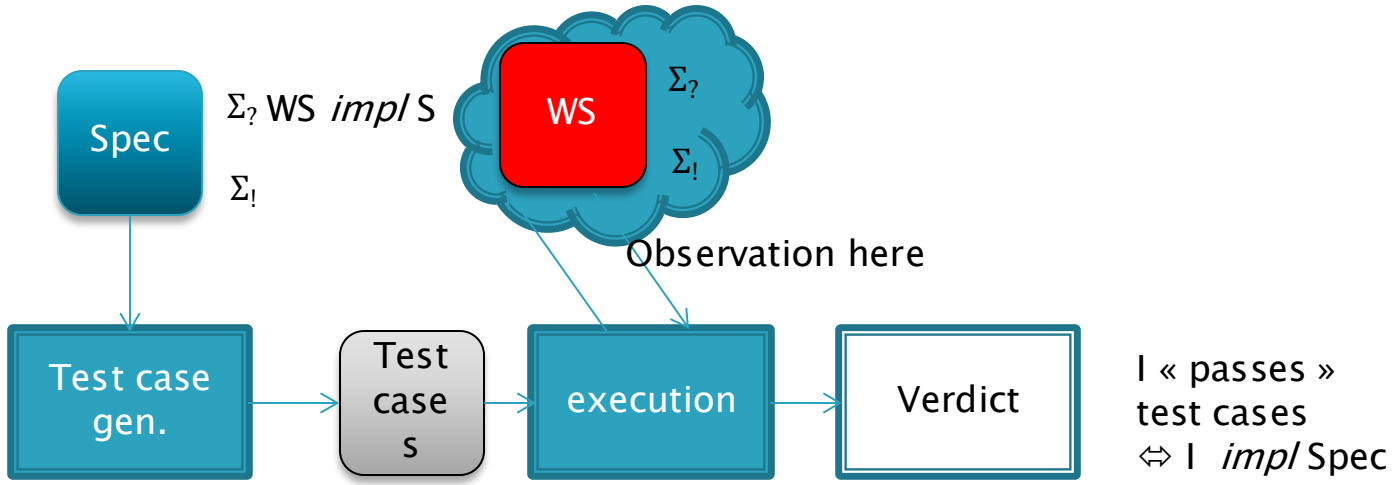
Suite de test exacte

I passe test ⇔
I (Syst) *imp* S ?

Test basé Modèle

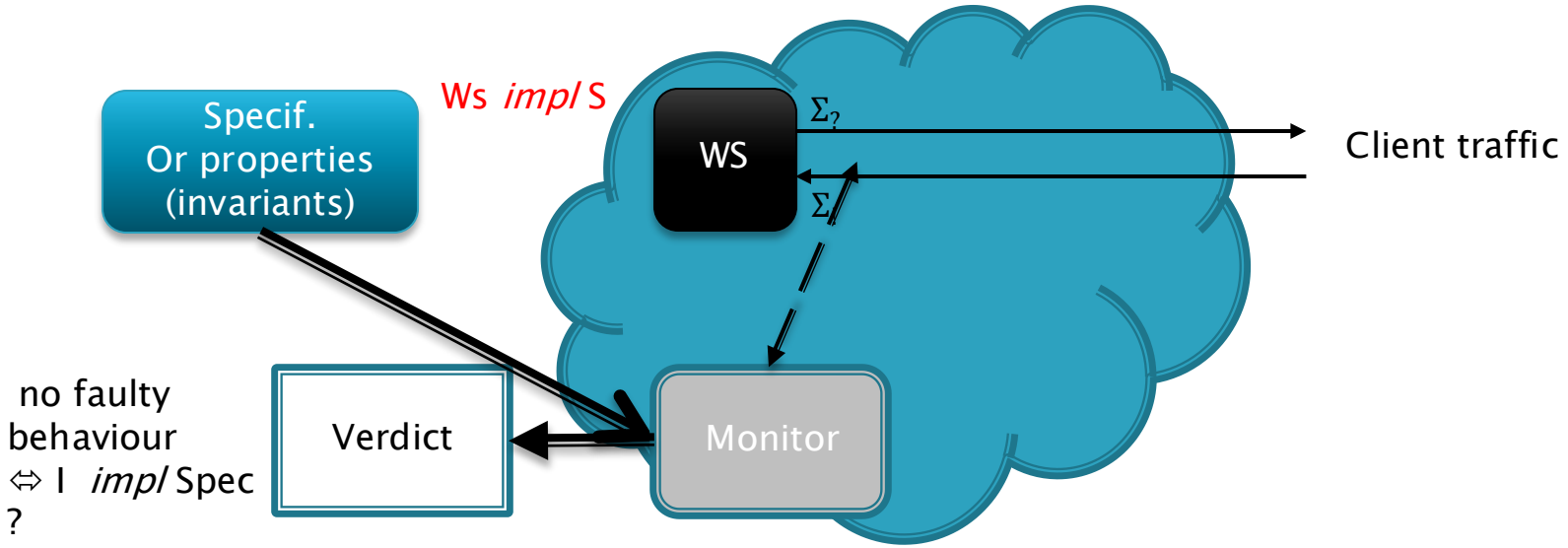


Test actif



From web service composition model -> test case gen. -> test case exec. -> verdict

Test passif



Monitoring de la composition de SW

[ACN10][BDANG7][BP09], etc.

Relation de test
Equivalence de traces

- Une trace = séquence d'actions observables
- Deux LTS sont équivalent en traces (*trace equivalent*) ssi ils possèdent les mêmes traces
- Equivalence de traces = relation de conformité la plus faible

Relation de test ioco

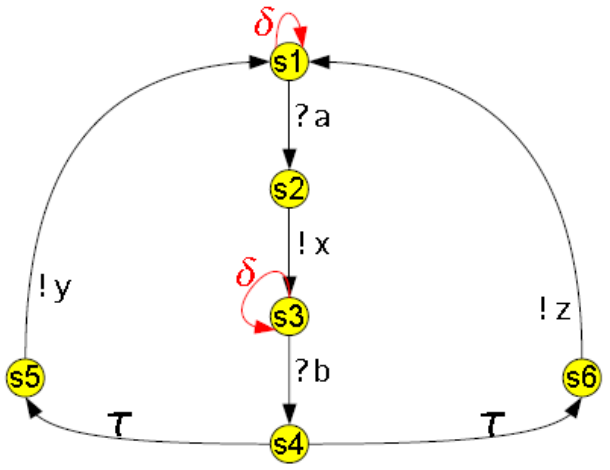
De façon intuitive, une trace d'un LTS représente une séquence d'actions extraites d'un chemin de ce LTS. Cette notion de trace et d'observation est reprise par Tretmans dans [Tre96c] en introduisant la notion d'observateur : un LTS p est équivalent à un LTS q si tout observateur peut faire les mêmes observations avec p et q. Il définit notamment les relations $ioco$, $iocon f$ et $ioco_{\mathcal{F}}$ qui est une généralisation des deux précédentes. Ainsi, la relation $ioco$ entre un LTS S et une implantation I modélisée également par un LTS est définie par :

$$I \text{ ioco } S =_{def} \forall \sigma \in Straces(S) : out(I \text{ after } \sigma) \subseteq out(S \text{ after} \sigma)$$

Relation de test ioco

δ

Definition s after σ = {s' in S \ $s \overset{\sigma}{\Rightarrow} s'$}



s1 after δ = {s1}

s1 after ?a = {s2}

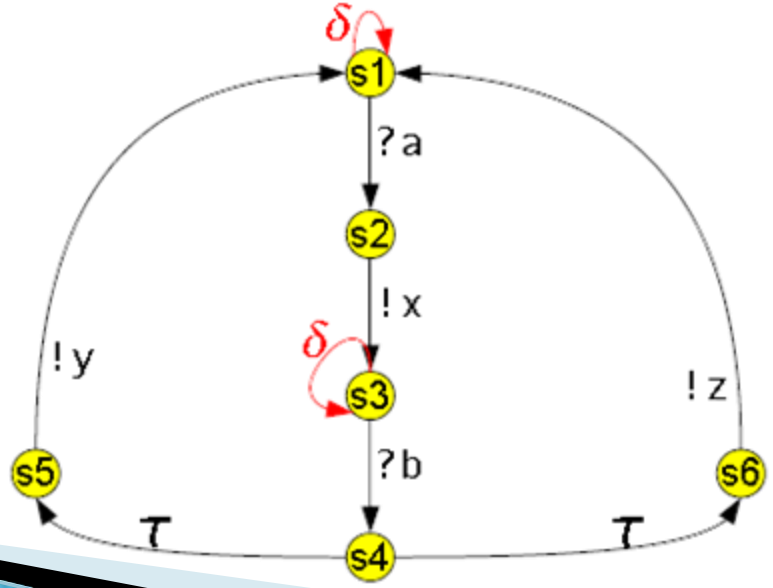
s2 after !x?b = {s4,s5,s6}

s2 after !xδ?b = {s4,s5,s6}

s1 after ?a!x$\delta\delta$?b!zδ = {s1}

Relation de test ioco

- $out(s) =_{def} \begin{cases} \{\delta\} & if\ \delta(s) \\ \{a \in \Sigma_U \mid s \xrightarrow{a}\} & otherwise \end{cases}$
- $out(C) =_{def} \bigcup_{s \in C} out(s)$

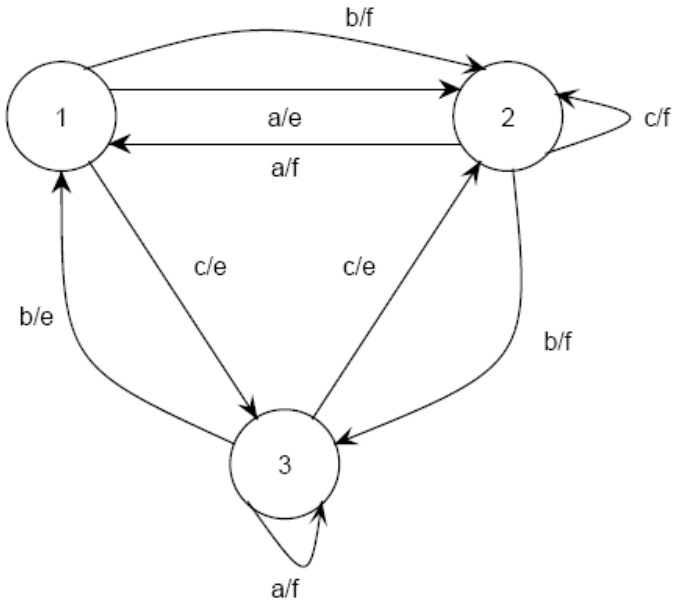


out(s1) = {δ}
out(s2) = {!x}
out(s3) = {δ}
out(s4) = $\emptyset$
out(s5) = {!y}
out({s1,s2}) = {δ,!x}

out(s1 after δ?a) = {!x}
out(s1 after ?a!x?b) = {!y,!z}

Tour de transition

- Test basé modèle de conformité sur automate:
- Tour de transition: consiste à trouver une séquence minimale, parcourant au moins une fois chaque transition de la FSM. On doit obtenir la même séquence sur l



{ c/e a/f b/e a/e b/f c/e c/f a/f b/f }

Problématique du test

- Besoin d'une suite de test complète et exacte pour que la relation de test soit vraie
  - Obtenir une suite de test complète et exacte est souvent impossible (suite très souvent infinie)
  - Besoin de définir des hypothèses sur les systèmes
    - S Déterminisme, Minimal, fortement connexe
    - I doit avoir les mêmes entrées / sorties que S
    - I et S peuvent être réinitialisés, ont un état initial, etc.

Problématique du test

- Test: processus coûteux
- Formaliser tout en évitant des explosions combinatoires
- Gen. d'une suite de test complète pas toujours possible
  - Choix des hypothèses pour limiter ?
  - Sélection des tests?

Problématique du test

Qualité du test

- Couverture de code suffisante ?

Mutation testing

Mutation testing

Mutation testing : évaluer la qualité des tests automatisés en introduisant intentionnellement des modifications (ou “mutations”) dans le code source pour simuler des erreurs ou des bugs potentiels.

1. **Gen. De Mutants** : Des versions modifiées du code sont générées en changeant légèrement certaines parties du programme. Ex. Outils Pitest

2. **Exécution des tests** : Les tests automatisés existants sont exécutés sur ces mutants.

3. Calcul du ratio killed mutants :
   - Si un test échoue pour un mutant, celui-ci est dit **“tué”**.
   - Si un mutant passe les tests, il est dit **“survivant”**, ce qui indique que les tests peuvent manquer de couverture ou ne sont pas suffisamment robustes.

⇒Plus le ratio de killed mutants est élevé, plus la suite de test est de qualité
⇒Fonctionne aussi pour les réseaux de neurones !

Vérification

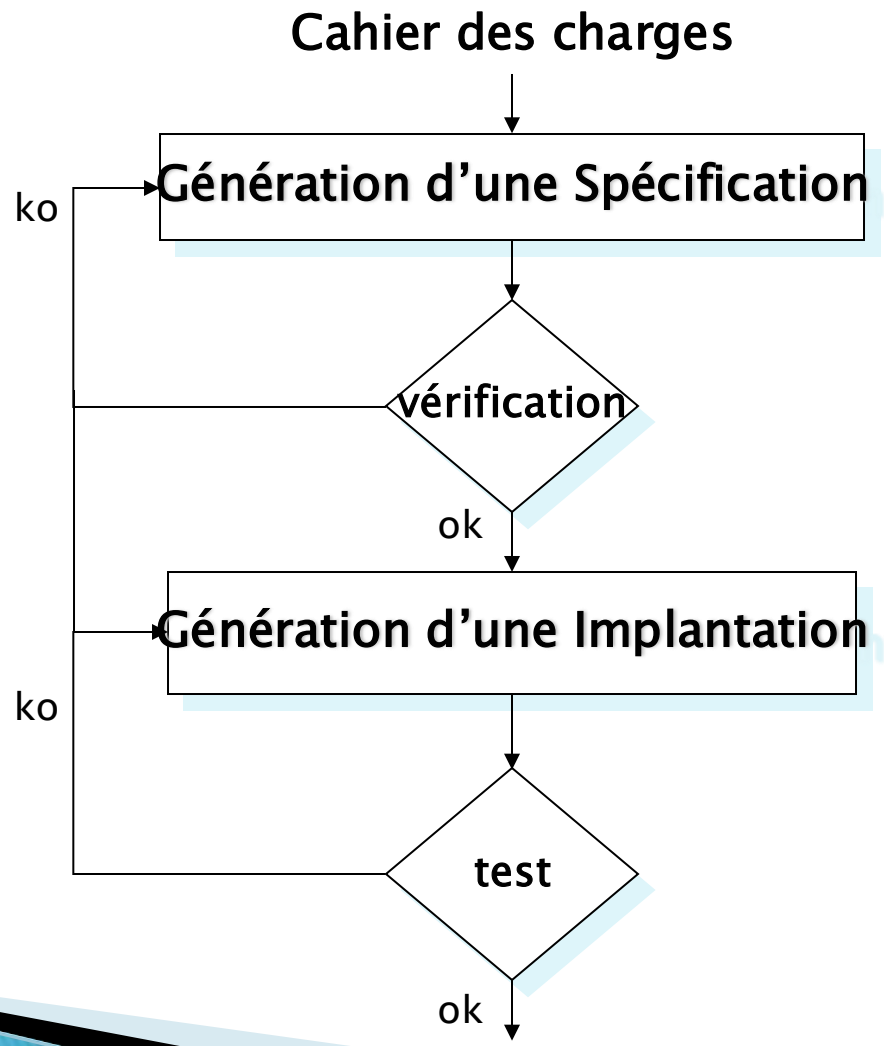
Preuve, model checking

La vérification

- a pour but de vérifier la cohérence de la spécification.

- « La vérification formelle est la production d'une preuve démontrant que le produit respecte effectivement la spécification formelle dont il "prétend" être la réalisation. »

La vérification



La vérification

- Plusieurs types:
  - Vérification de propriétés (preuve): analyse automatique de la spécification pour rechercher si des propriétés sont satisfaites
    - Interblocage (Deadlock)
    - Bouclage (Livelock)
    - Réception non spécifiée
    - Code mort (état inaccessible,
    - transition infranchissable…)

La vérification

- Plusieurs types:
  - Model checking: vérification exhaustive
    - Exprimé plus formellement par:

étant donné une formule logique p et un modèle M ayant un état initial s, le model checking vérifie si $M,s \models p$ (satisfait).

La vérification

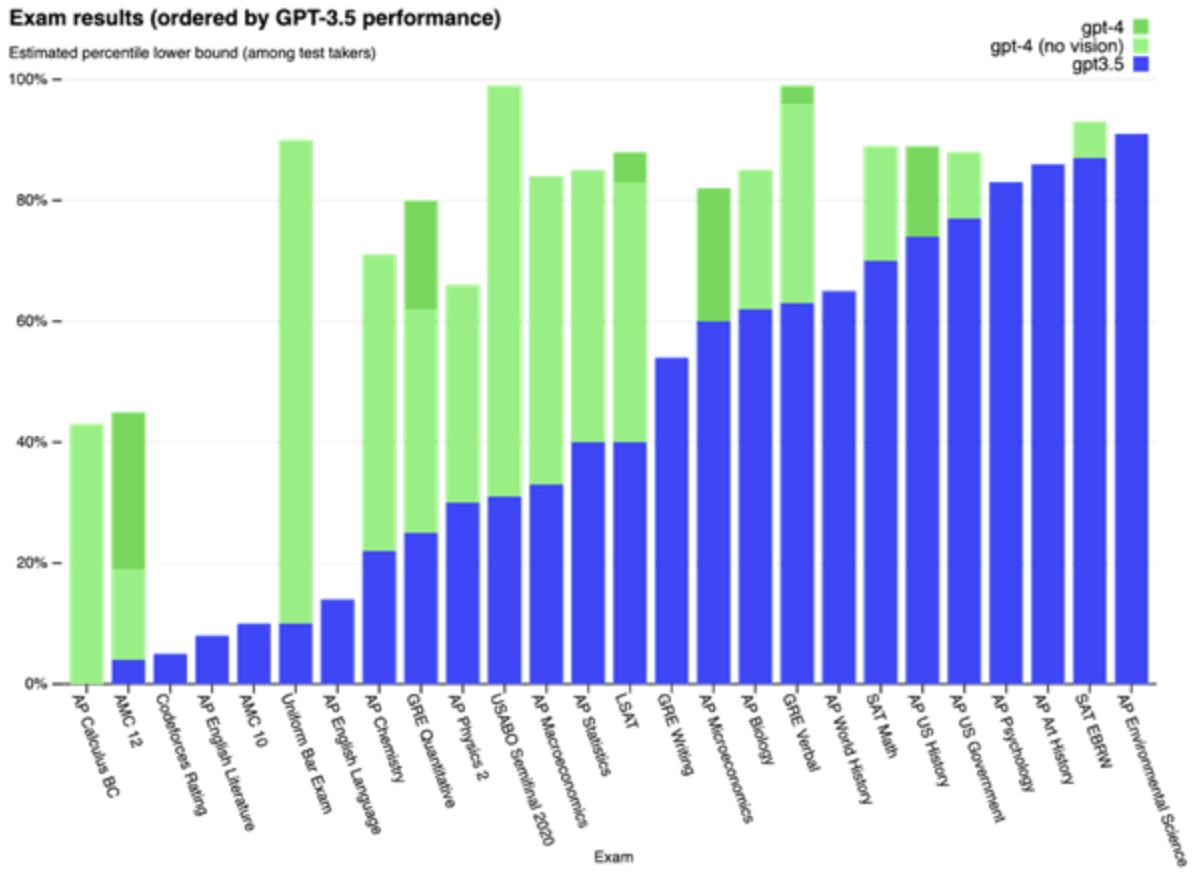
- Outils
- Spin basé sur langage promela (C++ + mots clés)
- Kronos (automates)

IA et test

GAN

IA et code gen.

- LLMs have been more and more used for programming.
- Kanade et al. proposed CuBERT, a specialised transformer model for code understanding (Kanade et al., 2020).
- Other papers showed that LLMs such as ChatGPT can solve easy and medium programming problems (Austin et al., 2021; Helander et al., 2024).
- They can also correctly audit applications to detect vulnerabilities and explain the results (Ma et al., 2024).



https://openai.com/research/gpt-4

IA et Test

- Code generation

- https://github.com/features/copilot
- **Llama 2, OpenAI Codex , code T5**

- Research 2023, in software engineering

https://github.com/reddy-lab-code-research/StructCoder/

Code generator from natural language or from another code source

IA et Test

- UN LLM ne pense pas (reasonning tel que o1, o3, QwQ etc.) decomposent un pb et utilisent des techniques de few shots)
- Un LLM hallucine
  - Peut vous donner un code qui lui semble correct mais qui ne n'est pas

- Seule façon de contrôler la fiabilité du code généré c'est le test (*test passing correctness + mutation testing*)

- Si du code de test est généré alors il faut le revoir attentivement
  - Flaky test ? Bonne isolation de certains composants ?

IA et Test

- Utiliser l'IA dans les tests peut s'avérer extrêmement pratique , exemple
- Test fonctionnels de GUI (web mobile)

- Problématique : l'interface change, le test est à réécrire

- Solution : utiliser l'IA pour naviguer et rechercher les éléments dynamiquement
  - Besoin de mettre des gardes-fous pour limiter, contrôler les hallucinations !

IA et Test

- Exemple (TP3)
- V1 tous les tests unitaires

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class MoneyTest {

    @Test
    public void testAdd() {
        Money money = new Money();
        money.add(50.0, "USD");
        assertEquals(50.0, money.getCurr(), 0.01);
        assertEquals("USD", money.getDevise());
    }

    @Test
    public void testAddMultipleTimes() {
        Money money = new Money();
        money.add(30.0, "EUR");
        money.add(20.0, "EUR");
        assertEquals(50.0, money.getCurr(), 0.01);
        assertEquals("EUR", money.getDevise());
    }

    @Test
    public void testGetCurr() {
        Money money = new Money();
        assertEquals(0.0, money.getCurr(), 0.01);
    }

@Test
    public void testGetDevise() {
        Money money = new Money();
        assertEquals("", money.getDevise());
    }

    @Test
    public void testToString() {
        Money money = new Money();
        money.add(75.5, "GBP");
        assertEquals("75.5 GBP", money.toString());
    }

    // Ajoutez d'autres tests au besoin pour couvrir toutes les méthodes

}
```

AVIS ??????

IA et Test

- Exemple:

V2 : tests de robustesse

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class MoneyTest {

    @Test
    public void testAddWithNegativeAmount() {
        Money money = new Money();
        money.add(-20.0, "USD");
        assertEquals(0.0, money.getCurr(), 0.01);
        assertEquals("", money.getDevise());
    }

    @Test
    public void testAddWithNullCurrency() {
        Money money = new Money();
        money.add(30.0, null);
        assertEquals(0.0, money.getCurr(), 0.01);
        assertEquals("", money.getDevise());
    }

    @Test
    public void testAddWithEmptyCurrency() {
        Money money = new Money();
        money.add(30.0, "");
        assertEquals(0.0, money.getCurr(), 0.01);
        assertEquals("", money.getDevise());
    }

    @Test
    public void testAddWithInvalidCurrencyConversion() {
        Money money = new Money();
        money.add(30.0, "XYZ"); // Assuming "XYZ" is an invalid currency
        assertEquals(0.0, money.getCurr(), 0.01);
        assertEquals("", money.getDevise());
    }

    @Test
    public void testToStringWithNullCurrency() {
        Money money = new Money();
        money.add(25.0, null);
        assertEquals("25.0 ", money.toString());
    }

    // Ajoutez d'autres tests de robustesse au besoin

}
```

AVIS ??????

IA et Test

- Exemple:

V2 : tous les tests de sécurité

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class MoneySecurityTest {

    @Test
    public void testAddWithMaliciousInput() {
        Money money = new Money();
        money.add(Double.POSITIVE_INFINITY, "USD");
        assertEquals(0.0, money.getCurr(), 0.01);
        assertEquals("", money.getDevise());
    }

    @Test
    public void testAddWithSQLInjection() {
        Money money = new Money();
        money.add(50.0, "'; DROP TABLE Money; --");
        assertEquals(50.0, money.getCurr(), 0.01);
        assertEquals("'; DROP TABLE Money; --", money.getDevise());
    }

    @Test
    public void testAddWithXSSAttack() {
        Money money = new Money();
        money.add(30.0, "<script>alert('XSS');</script>");
        assertEquals(30.0, money.getCurr(), 0.01);
        assertEquals("<script>alert('XSS');</script>", money.getDevise());
    }

    @Test
    public void testAddWithOverflow() {
        Money money = new Money();
        money.add(Double.MAX_VALUE, "USD");
        assertEquals(Double.MAX_VALUE, money.getCurr(), 0.01);
        assertEquals("USD", money.getDevise());
    }

    // Ajoutez d'autres tests de sécurité au besoin

}
```

AVIS ?????

IA et Test

- Exemple:
- test unitaire en Java qui permet de tester la méthode add avec des currency différentes

```
@Test public void testAddWithDifferentCurrencies() {
Money money = new Money(); money.add(50.0, "USD");
Money anotherMoney = new Money();
anotherMoney.add(30.0, "EUR"); // Supposez que la conversion de 30 EUR en USD donne 35 USD
money.add(anotherMoney.getCurr(), anotherMoney.getDevise());
assertEquals(85.0, money.getCurr(), 0.01); // 50 USD + 35 USD = 85 USD
assertEquals("USD", money.getDevise()); }
```

Réécrire le test en utilisant un mock pour la classe Convertion

```
@Test public void testAddWithDifferentCurrencies() { // Crée un mock de la classe Convertion
Convertion conversionMock = Mockito.mock(Convertion.class);

// Définir le comportement du mock (supposons que 30 EUR en USD donne 35 USD) Mockito.when(conversionMock.convert(30.0, "EUR", "USD")).thenReturn(35.0);
Money money = new Money();
money.add(50.0, "USD");
Money anotherMoney = new Money();
 anotherMoney.add(30.0, "EUR");

// Utilisez le mock pour la conversion
double convertedAmount = conversionMock.convert(anotherMoney.getCurr(), anotherMoney.getDevise(), "USD");
money.add(convertedAmount, "USD");

// Vérifiez le résultat
assertEquals(85.0, money.getCurr(), 0.01); // 50 USD + 35 USD = 85 USD
assertEquals("USD", money.getDevise()); }
```

IA et Test

- Tests incomplets
- Aspects métiers ne peuvent être générés (seront recopiés d'autres projets)
- Besoin de structurer, d'architecturer les tests
- Besoin de définir des stratégies de test
- IA est un guide pour générer le code et c'est tout pour le moment

- Attention au code généré farfelu !

- Perspective ? : des modèles d'IA avec amélioration continue du code et des tests à partir de besoins fonctionnels
- Vous devez donc comprendre et être capable de faire mieux que l'IA (de la critiquer notamment)