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

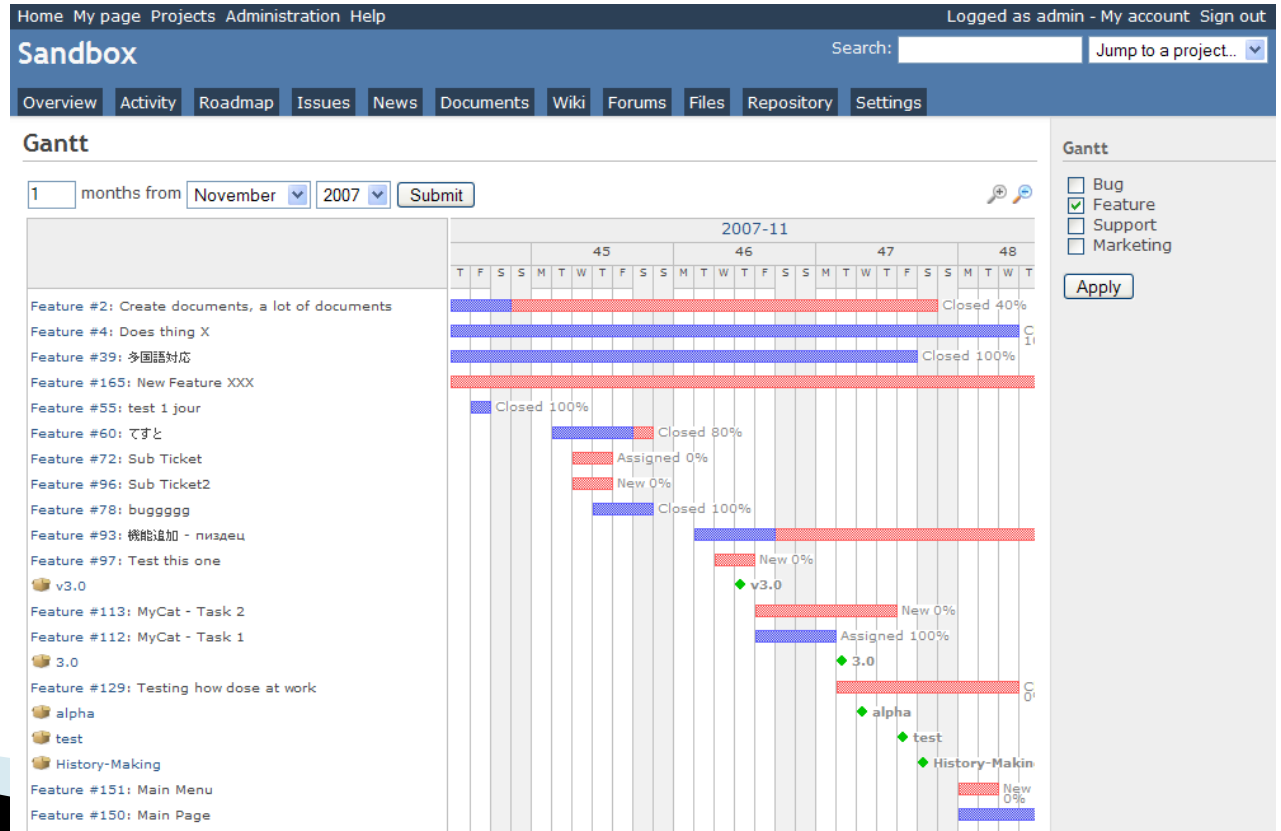
MbT, vérification

Environnement de test

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

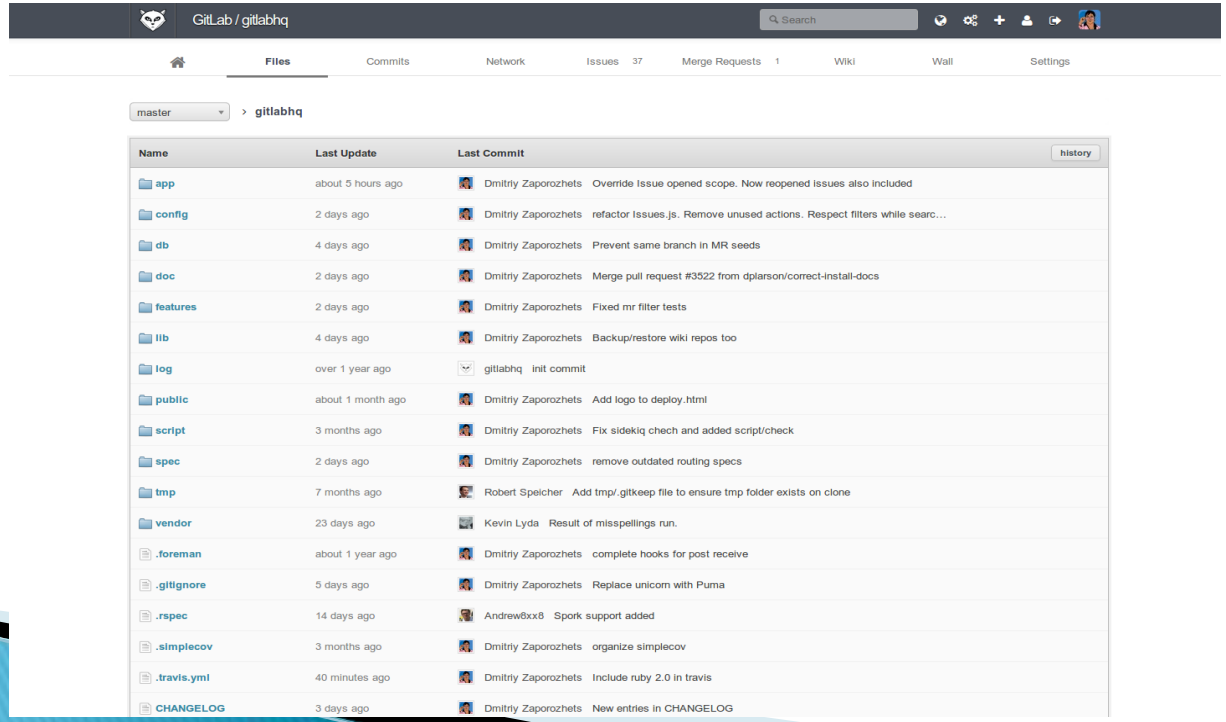
1. Environnement de gestion de projet (redmine ou gitlab mais moins complet)

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



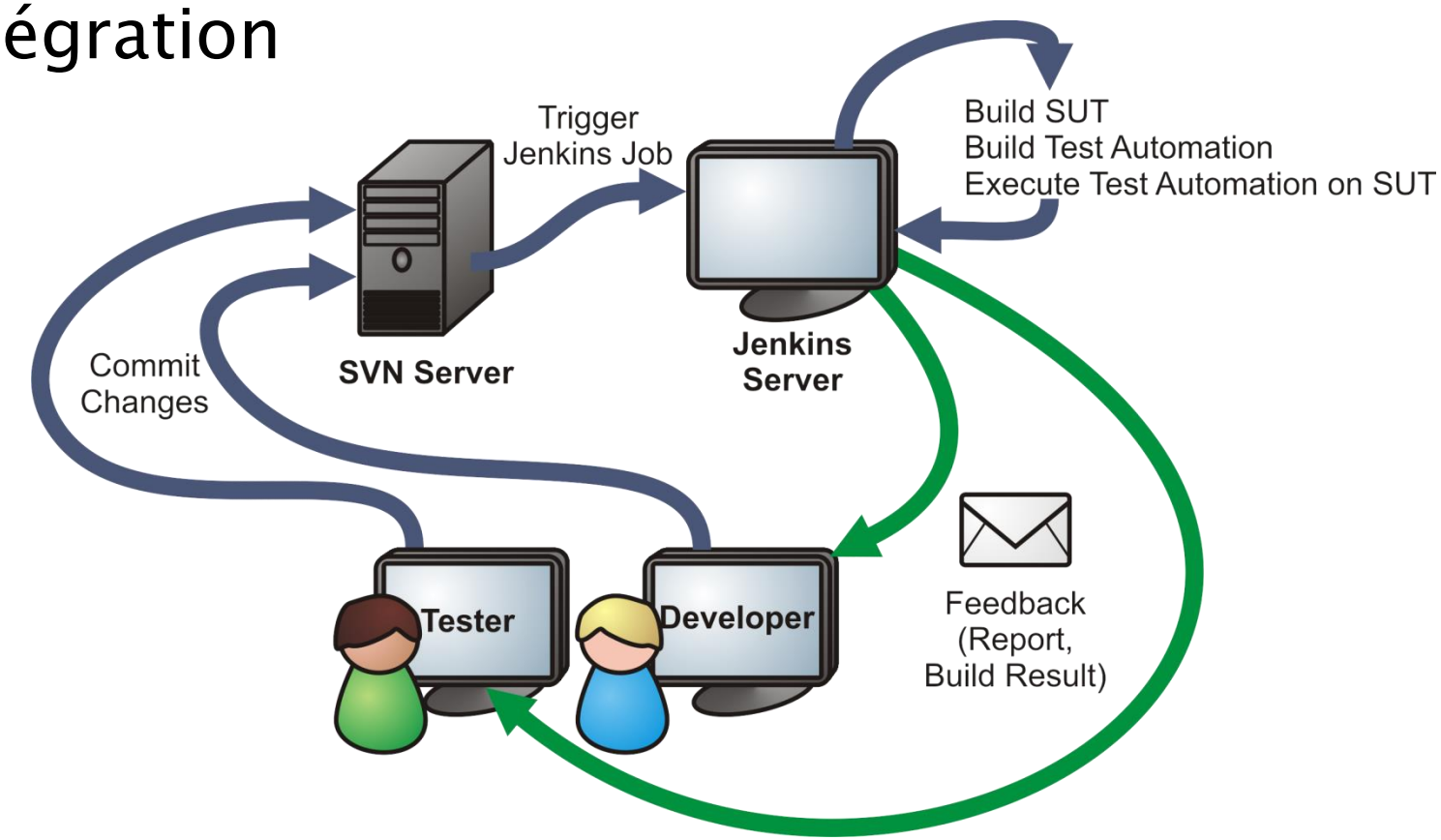
Environnement de test

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



Environnement de test

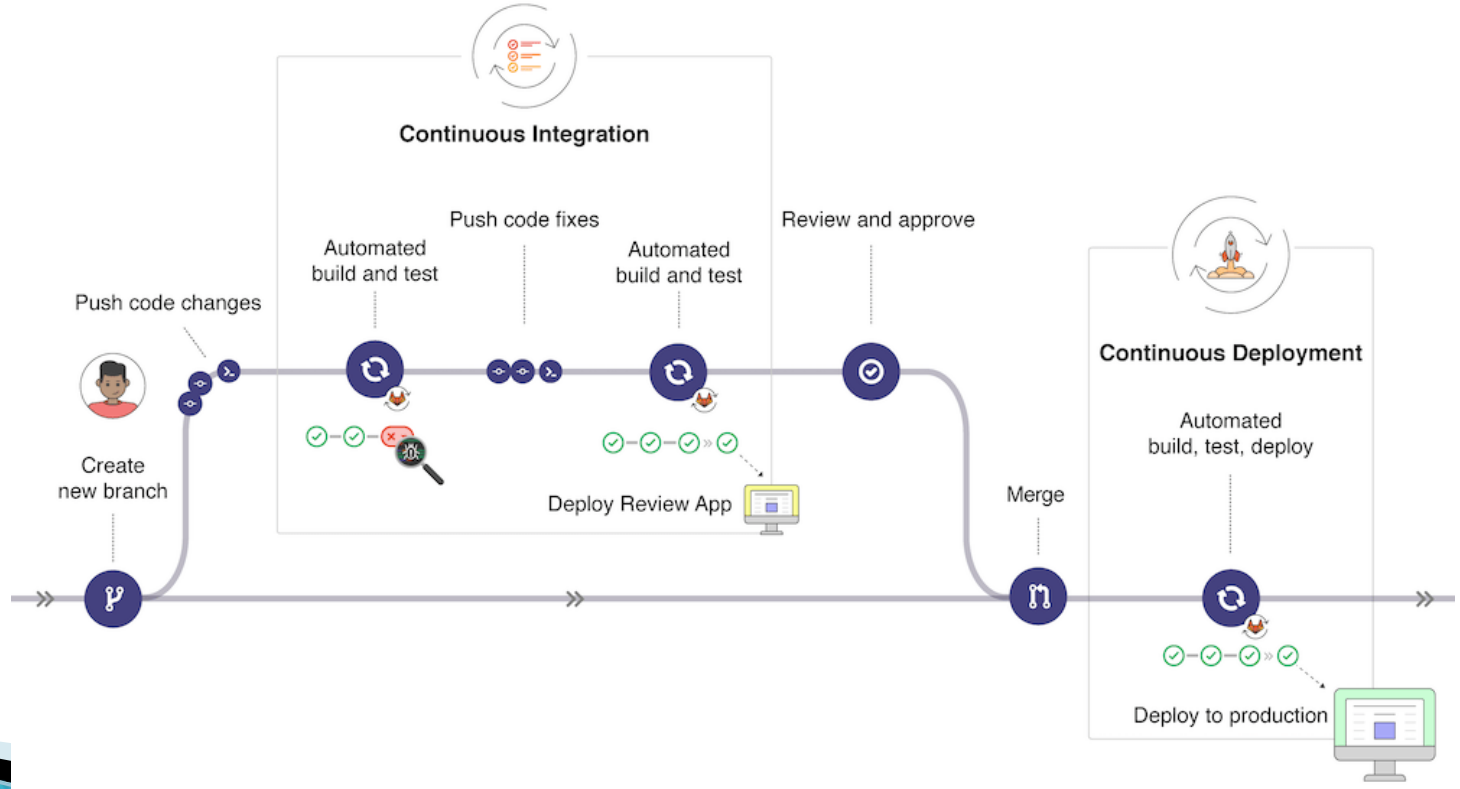
3. Serveur d'intégration
- Jenkins:



+ 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 unitiaire, 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.)

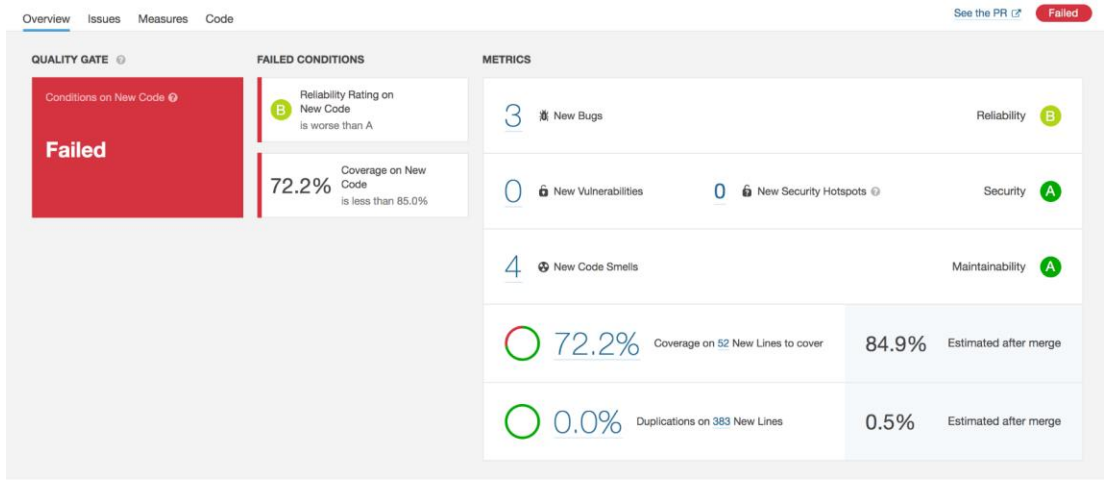
Outils

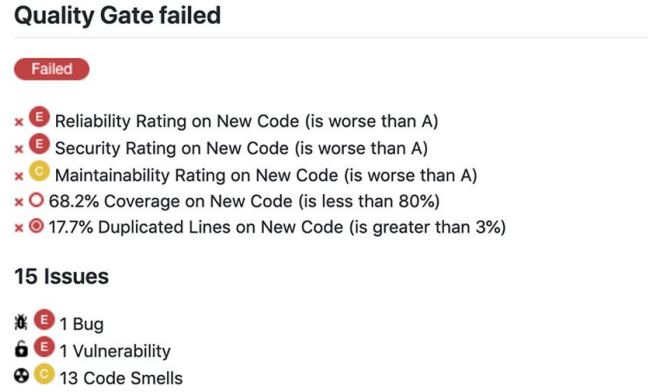
- Outils de contrôle qualité: SonarQube, SonarLint (plugin IDE)
  - SonarQube http://www.sonarqube.org

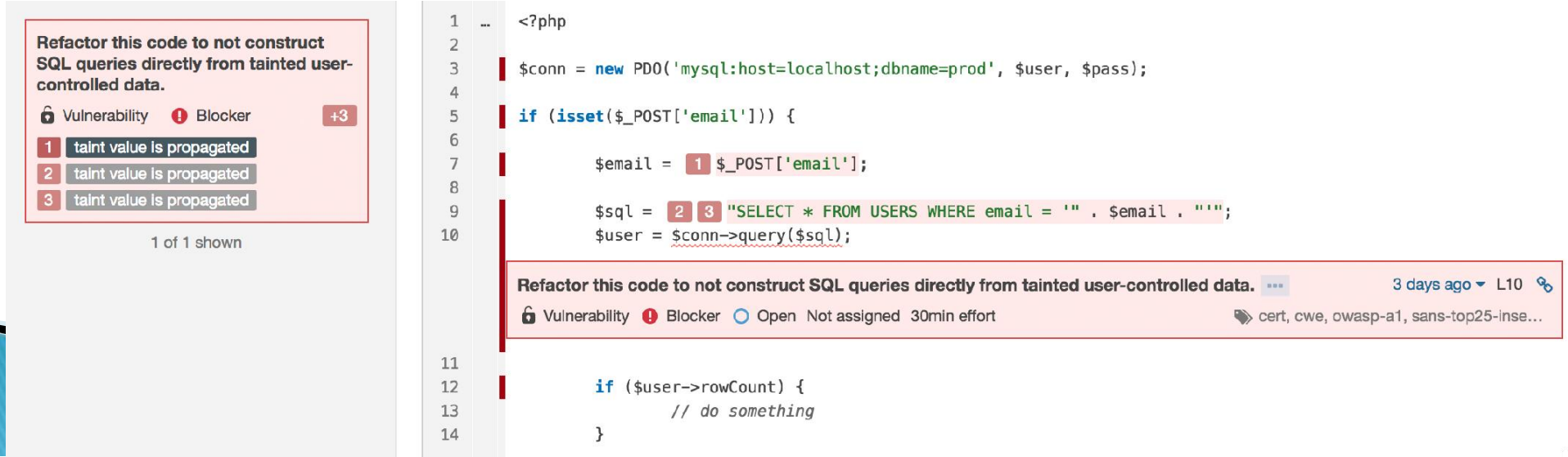
- Tableau de bord complet des différents projets suivis.
- Détection rapide du code à risque.
- Mesures quantitatives : nombre de classes, duplication de code, etc...
- Mesures qualitatives : couverture et taux de réussite des tests, complexité du code, respect des règles de codage...
- Sécurité (détection 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/)

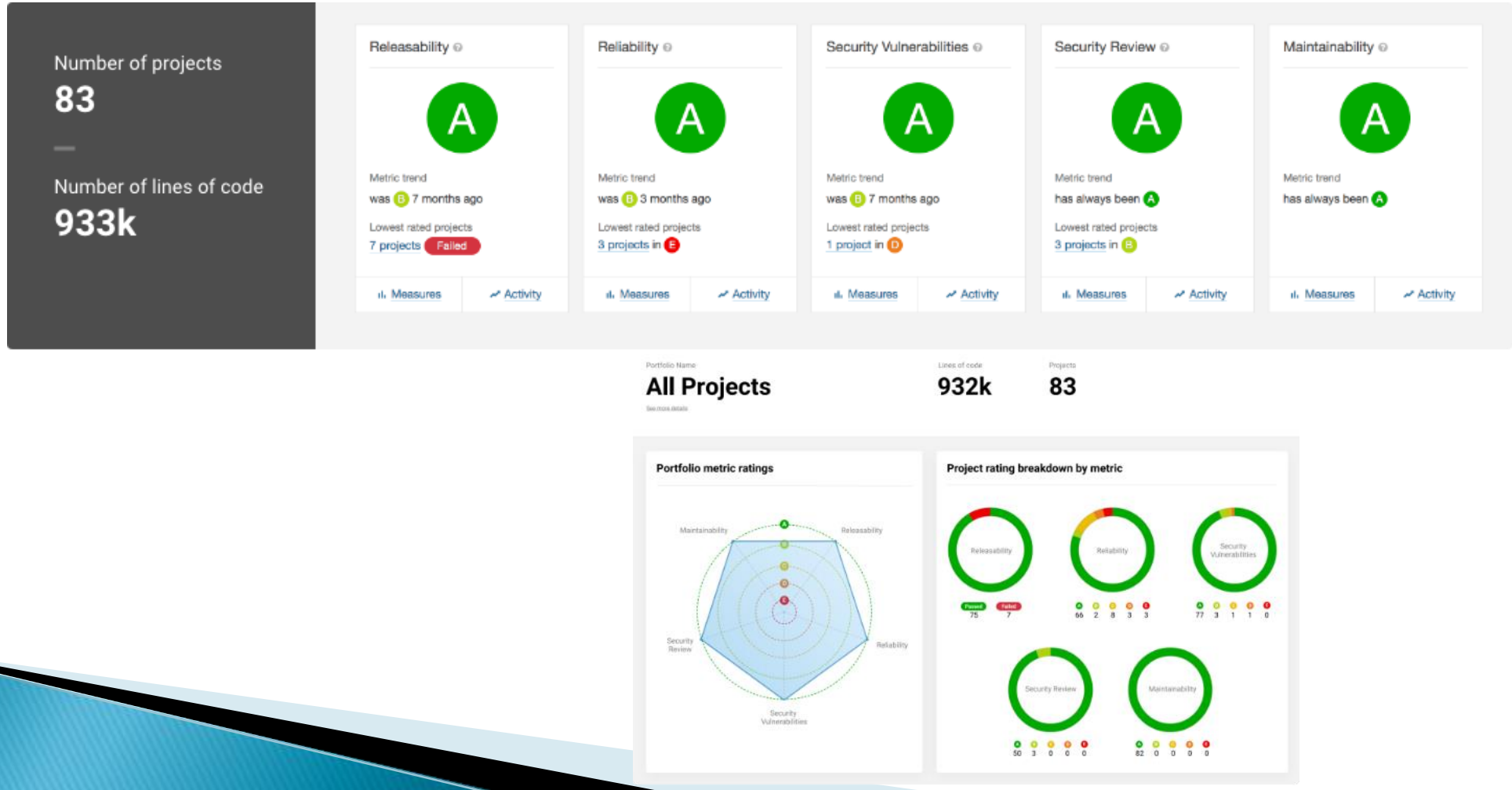






Outils

- 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

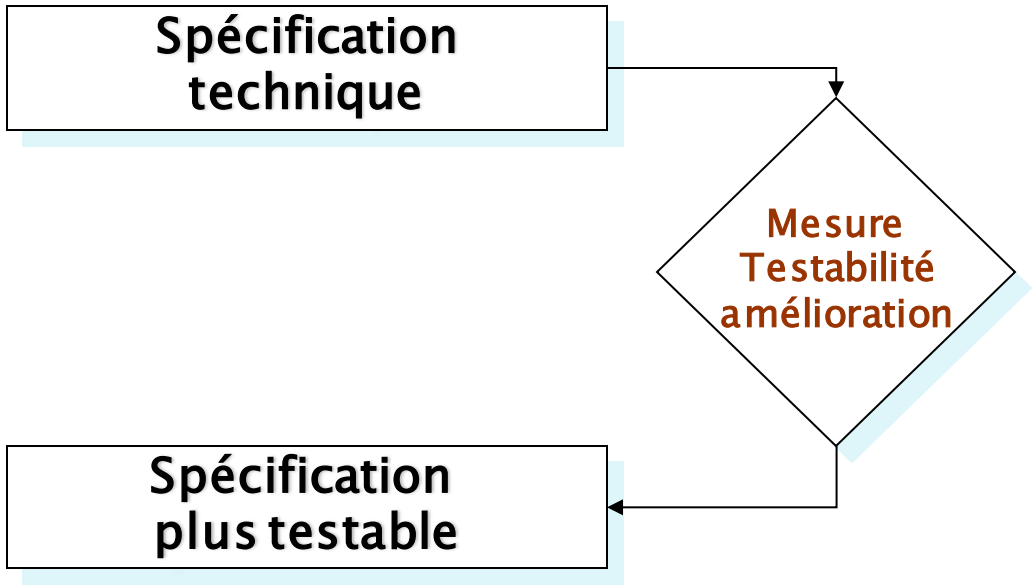
Testabilité

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

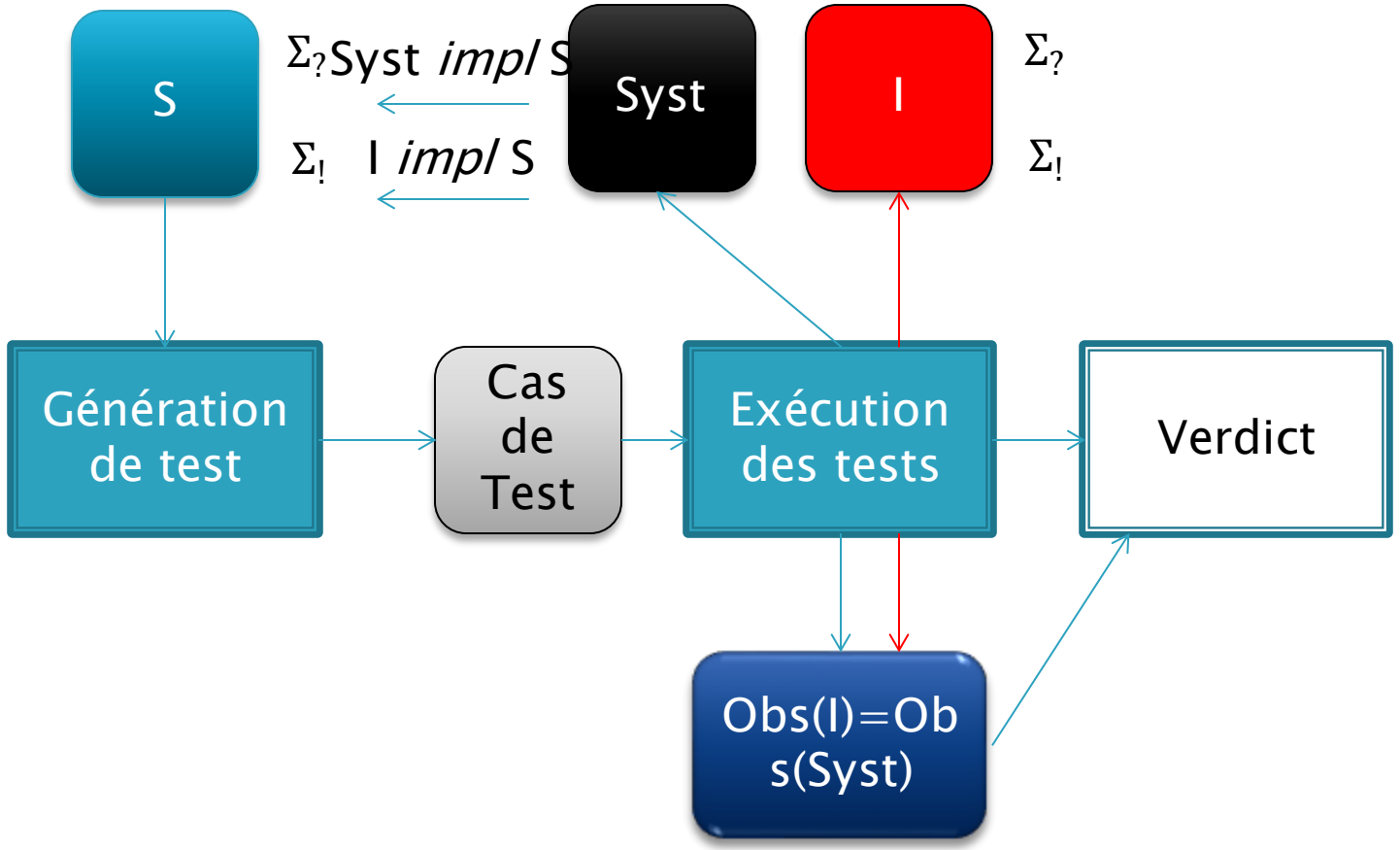
Test basé modèle

- 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 ?

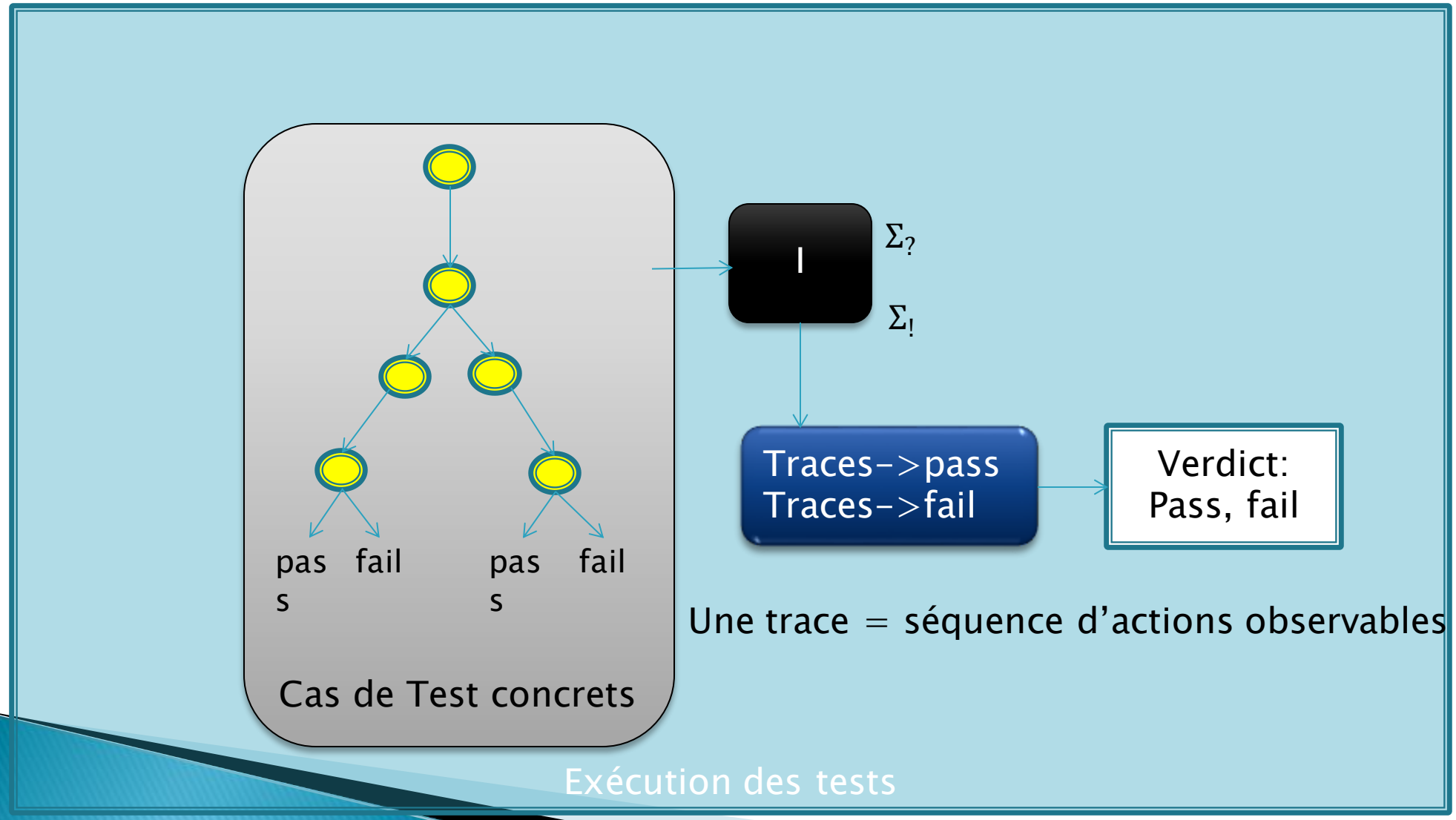
Test basé modèle

- 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



Problématique du test

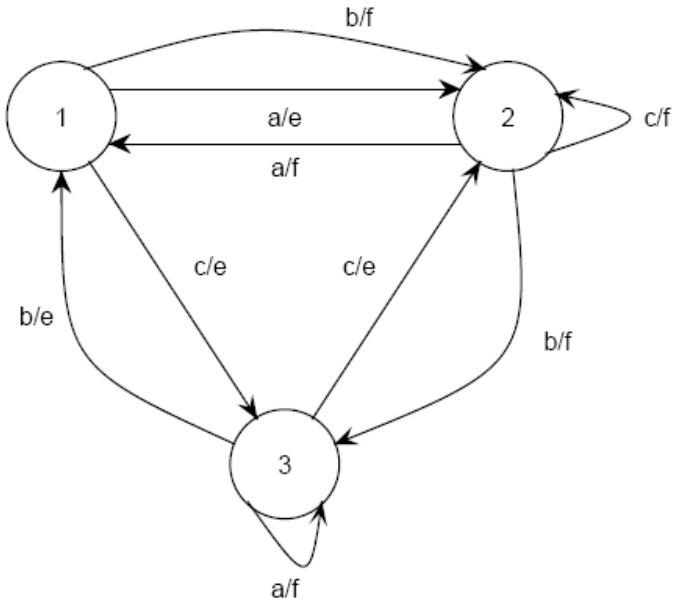
- Besoin d'une suite de test complète et exacte pour que la relation de test soit vraie
  - Obtenir une suite de test complète et exacte est souvent impossible (suite très souvent infinie)
  - Besoin de définir des hypothèses sur les systèmes
    - S Déterminisme, Minimal, fortement connexe
    - I doit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

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

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 }

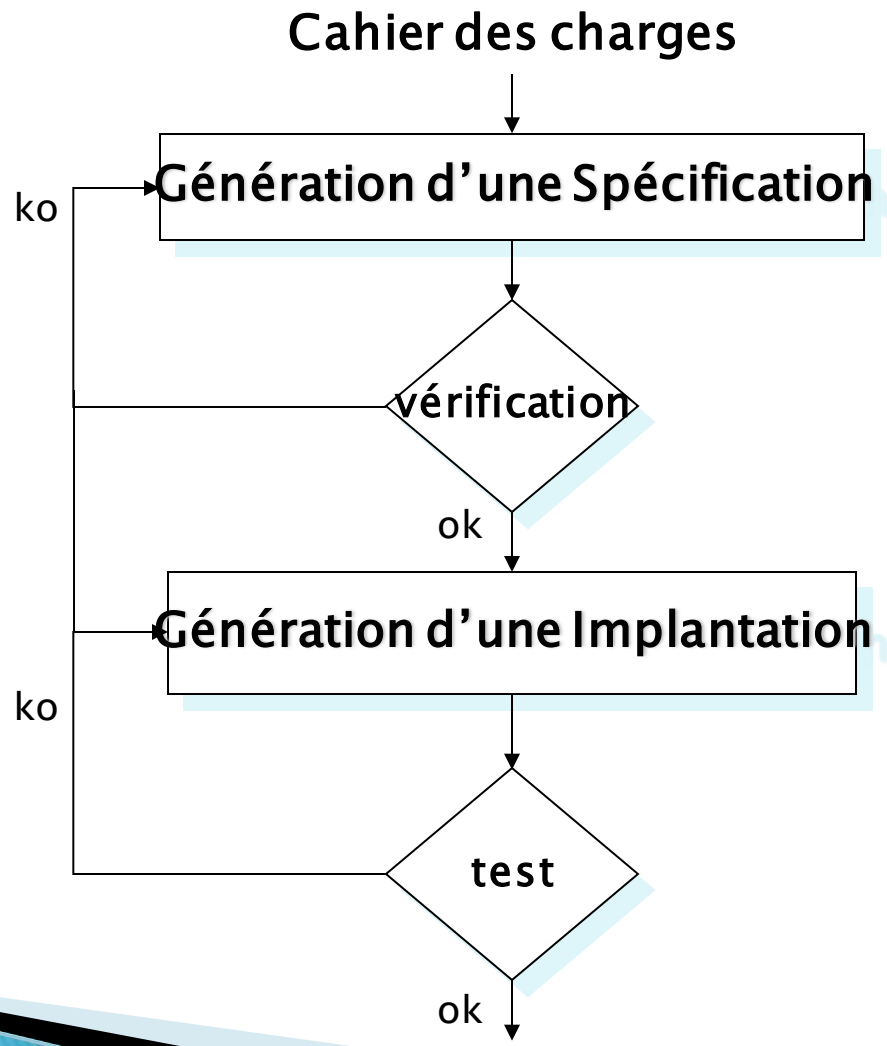
Vérification

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 Test

- Utilisation des GAN (**Generative Adversarial Network) 2014**

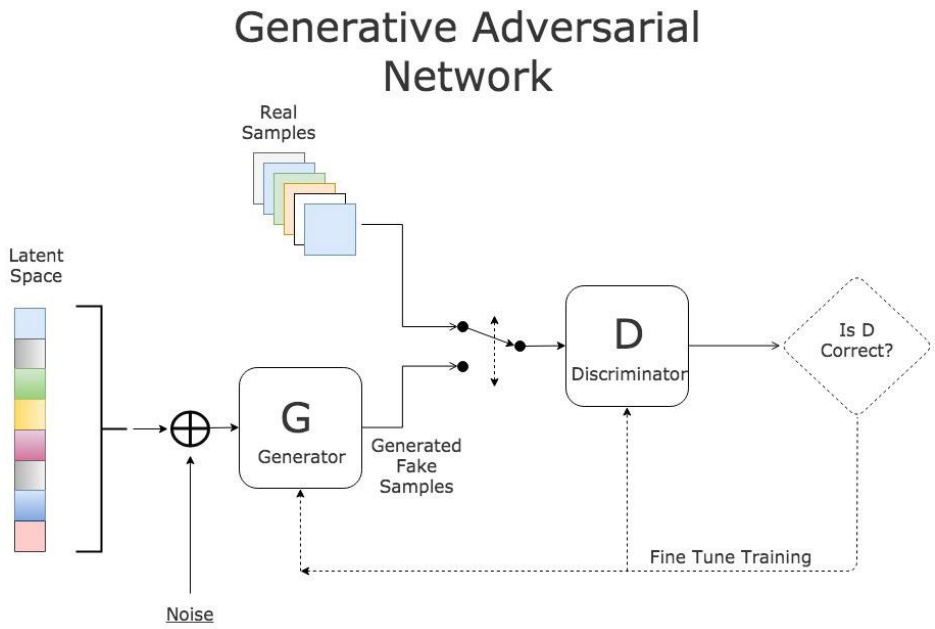
generative models capable of creating new data resembling the training data.

- **Transformers 2017**

Transformers are a type of neural network capable of understanding the context of sequential data, such as sentences, by analyzing the relationships between the words.

- GPT2

GPT-2 is a transformer-based language model with 1.5 billion of parameters trained on a dataset of 8 millions web pages.



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

- 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)