Tests

Institut de la filière numérique

Quelle est la conséquence ?

Enjeux humains

Enjeux financiers importants

Enjeux financiers faibles

Perte de confort

Différents types de test

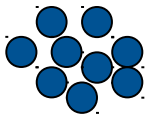
Tests de l'interface graphique. Identiques à ceux réalisés par la MOA.

Recette

Tests fonctionnels exécutés avant une livraison (qualification, validation).

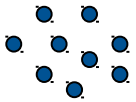
Acceptation

Boite noire



Valide le fait que toutes les parties développées indépendamment fonctionnent bien ensemble.

Intégration

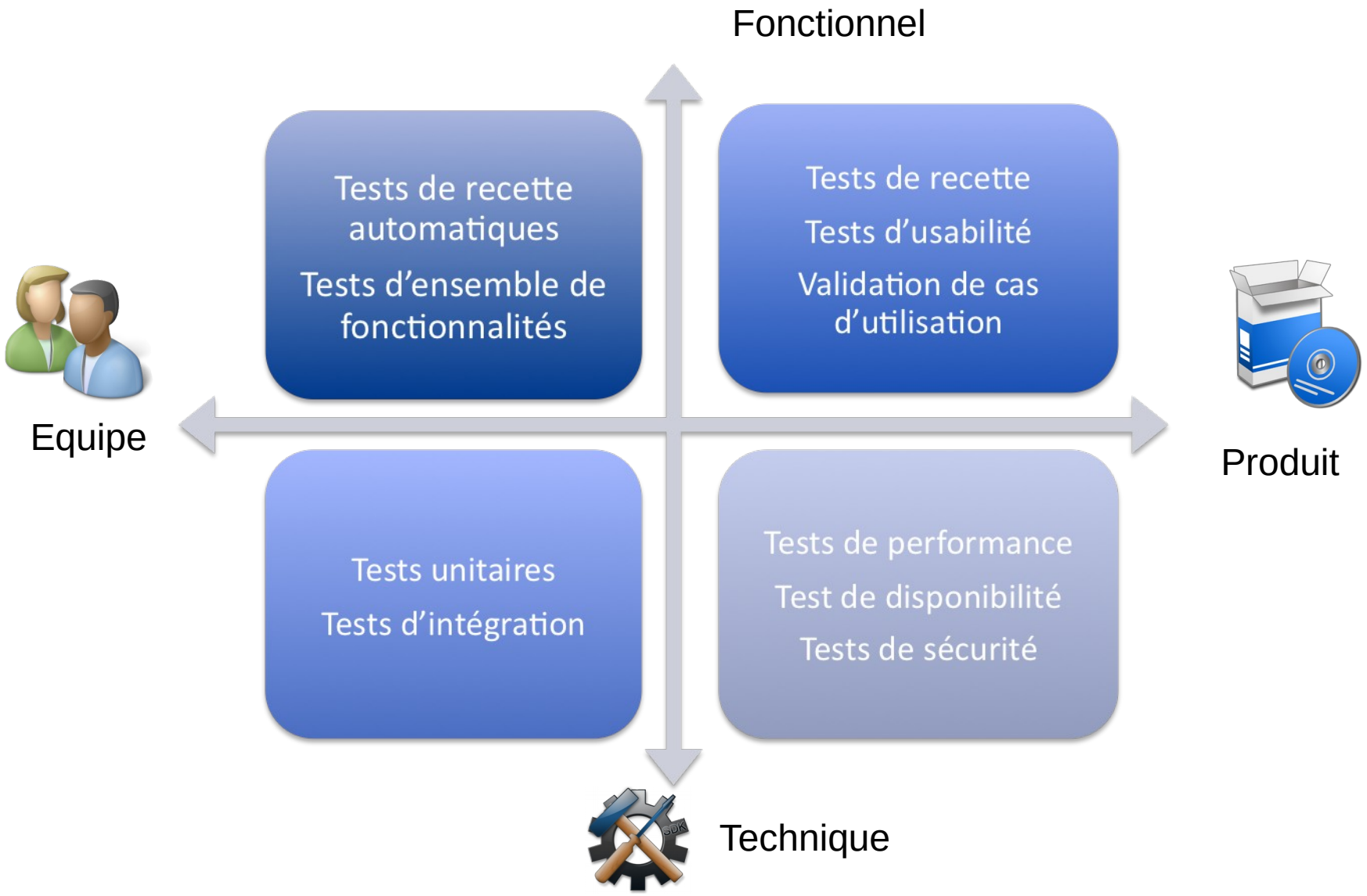


Teste une partie donnée, indépendamment du reste du programme.

Unitaire

Boite blanche

Différents types de test

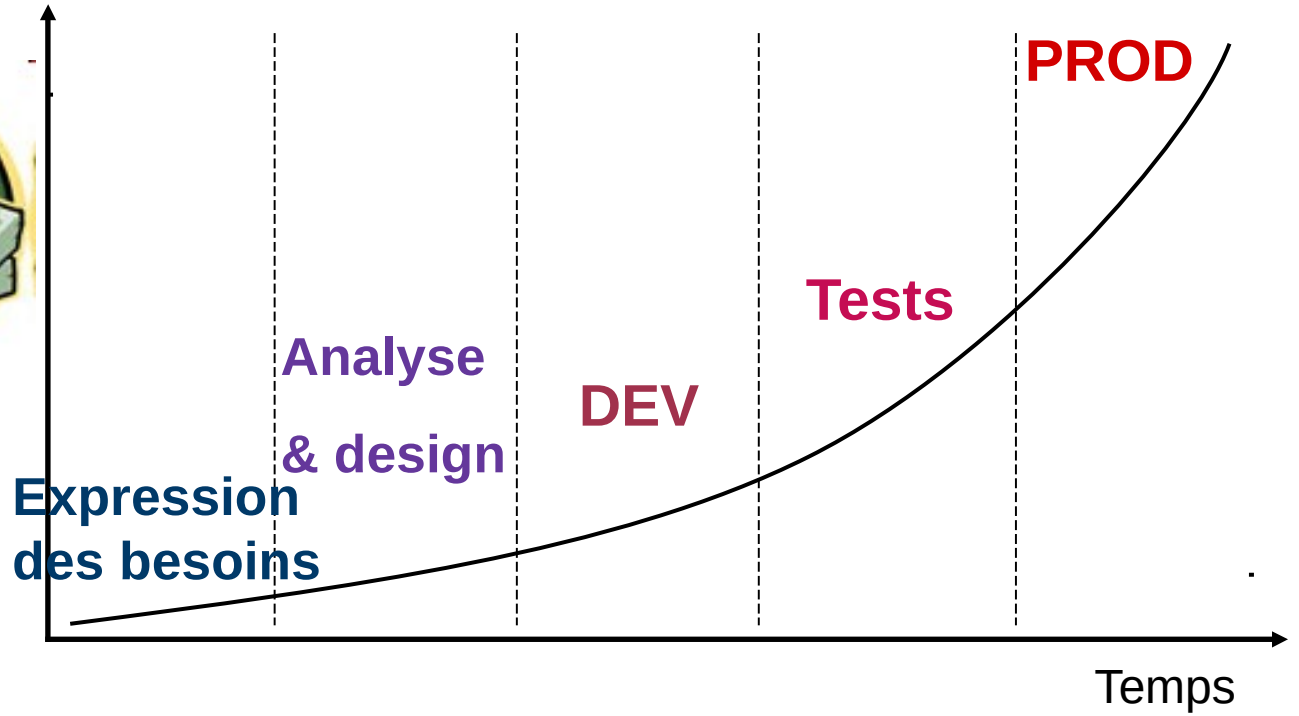


A quoi servent les tests?

- Assurer la qualité
  - L'application fonctionne
  - L'application fait ce qui est attendu
  - L'application a de bonnes performances
- Mais pas que ça
  - Assurer la **non régression**
  - Permettre le **refactoring**
  - Permettre de **comprendre** le code (les tests explicitent le code)
  - Ne plus avoir peur de tout casser

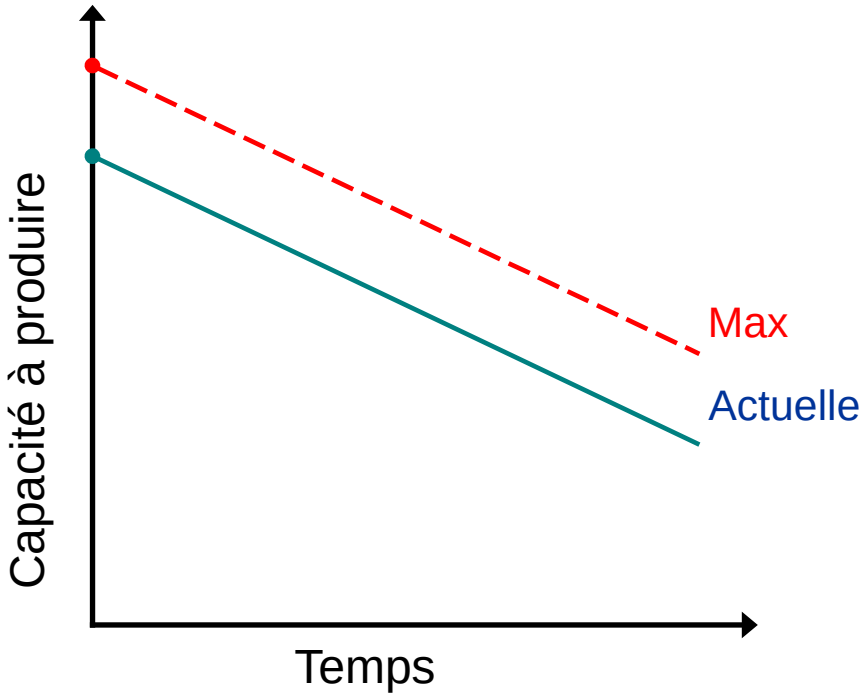
Le coût des bugs

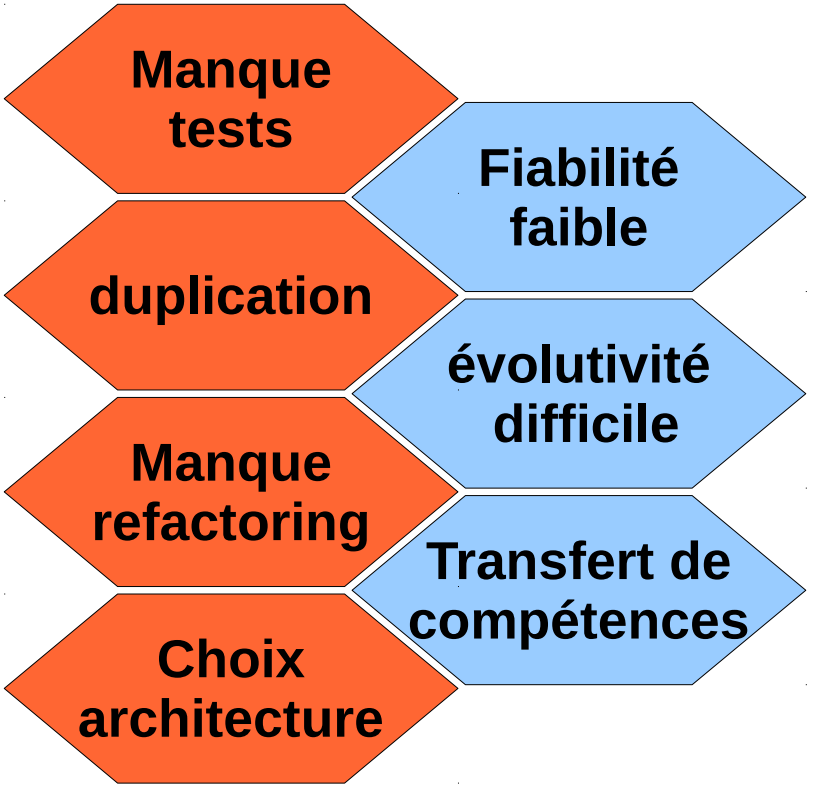
- Les bugs doivent être corrigés
- Mieux vaut le faire le plus tôt possible.



La dette technique

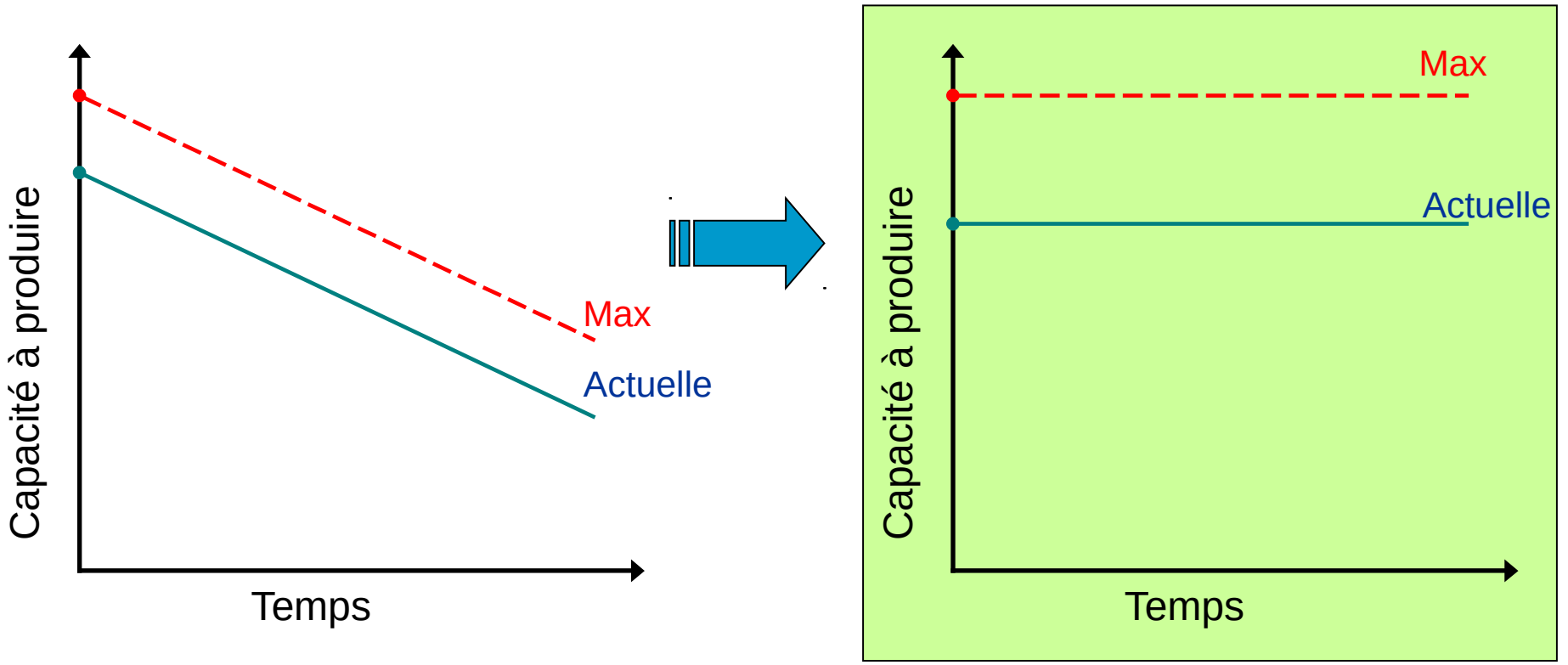
- Plus le temps passe, plus les erreurs commises se font ressentir
- La dette technique fait diminuer la productivité au fil du temps



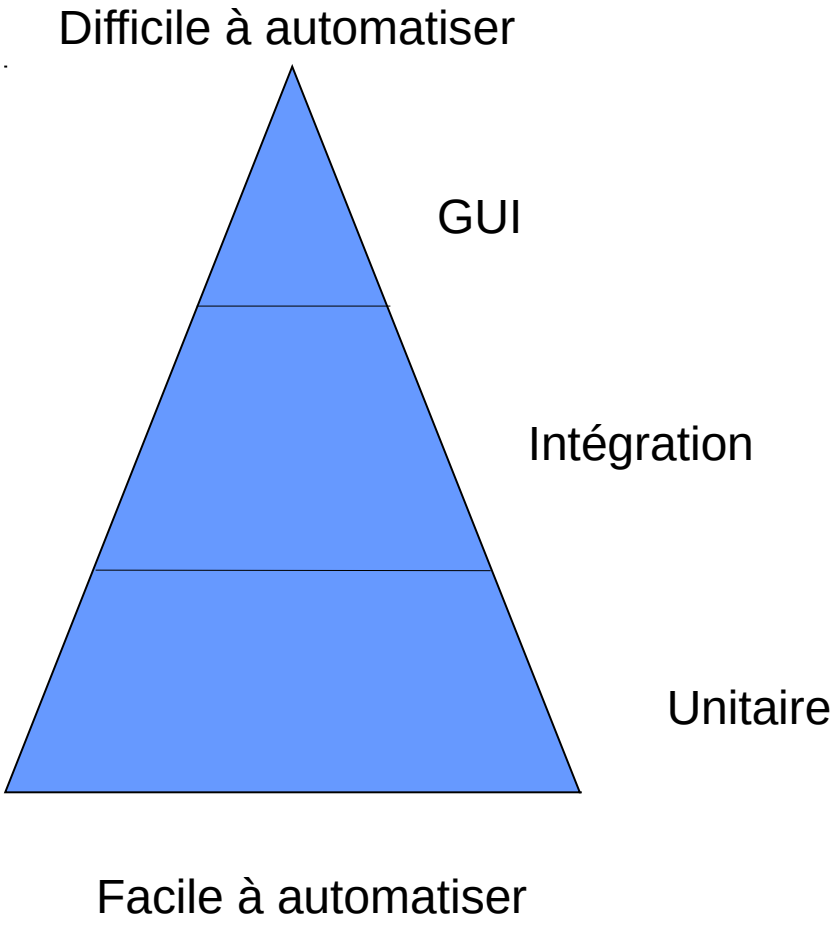


La dette technique

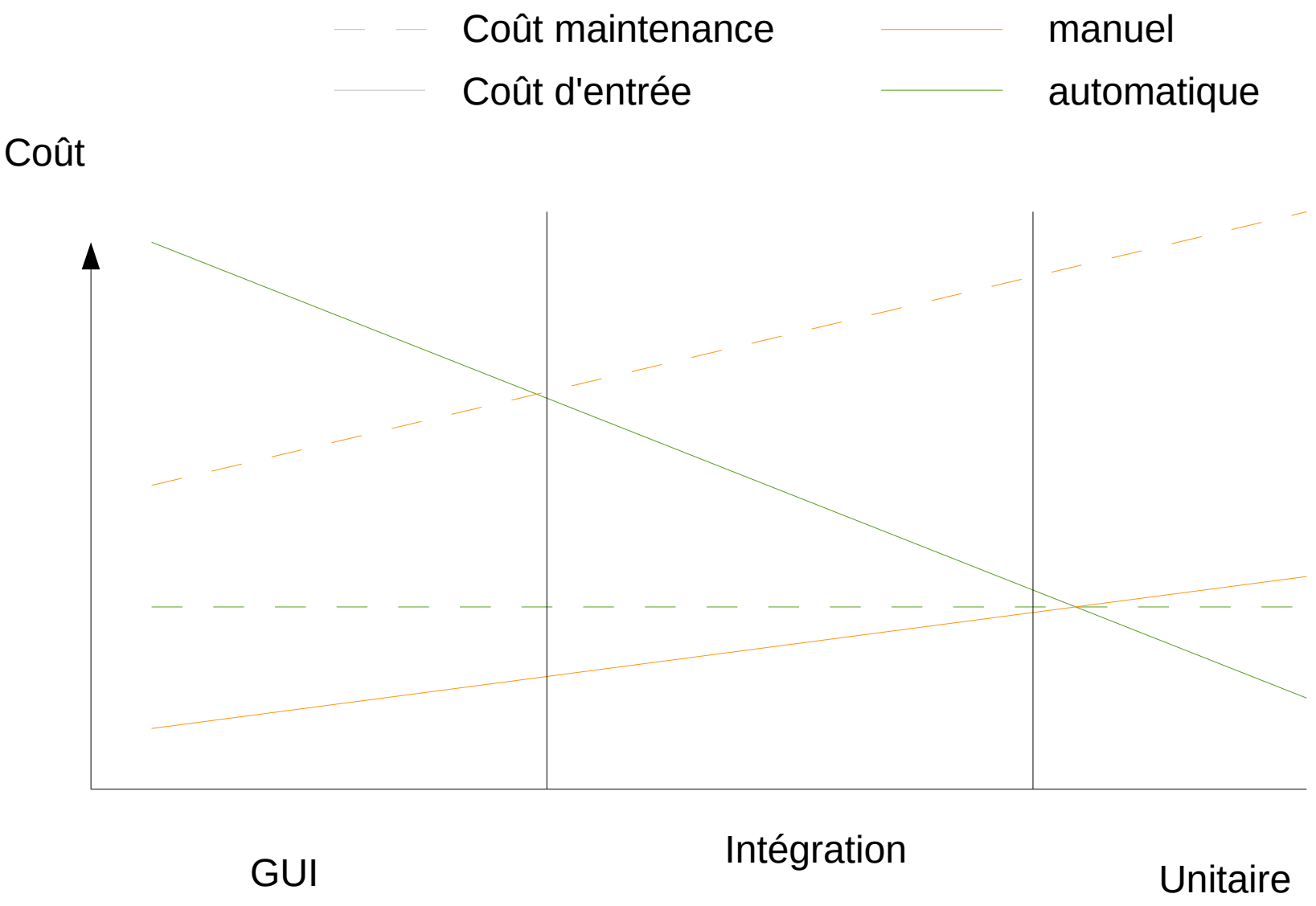
- Produire du code de meilleur qualité et testé à un coût plus élevé uniquement sur le court terme



Pyramide de Mike Cohn



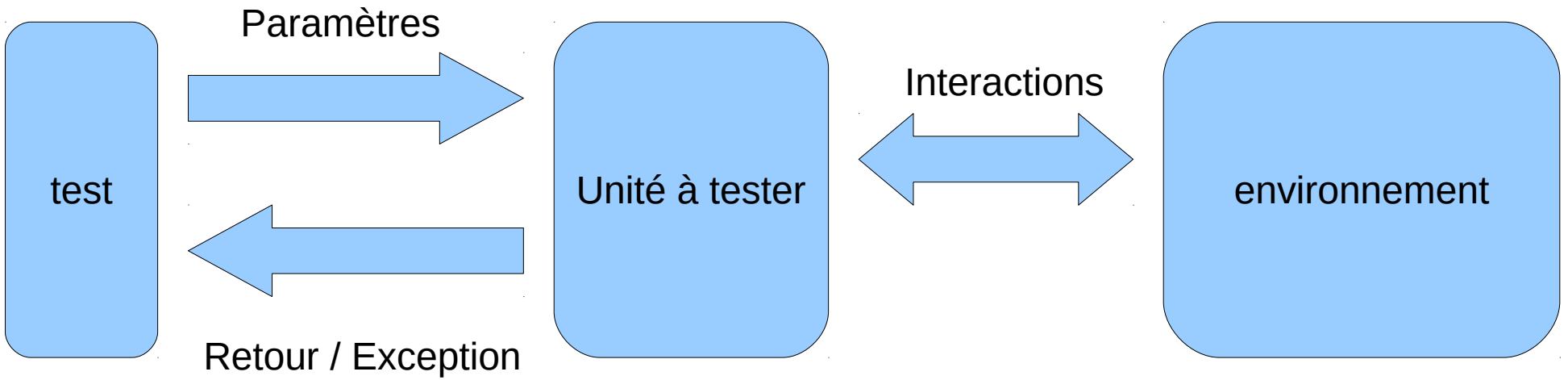
ROI


Coût maintenance
Coût d'entrée
manuel
automatique
Coût
GUI
Intégration
Unitaire

Test Unitaire

- validation des algorithmes indépendants
- limité à une partie restreinte de l'application
- automatisable en écrivant du code
- simple à mettre en place
- extrêmement rapide à exécuter

Test Unitaire



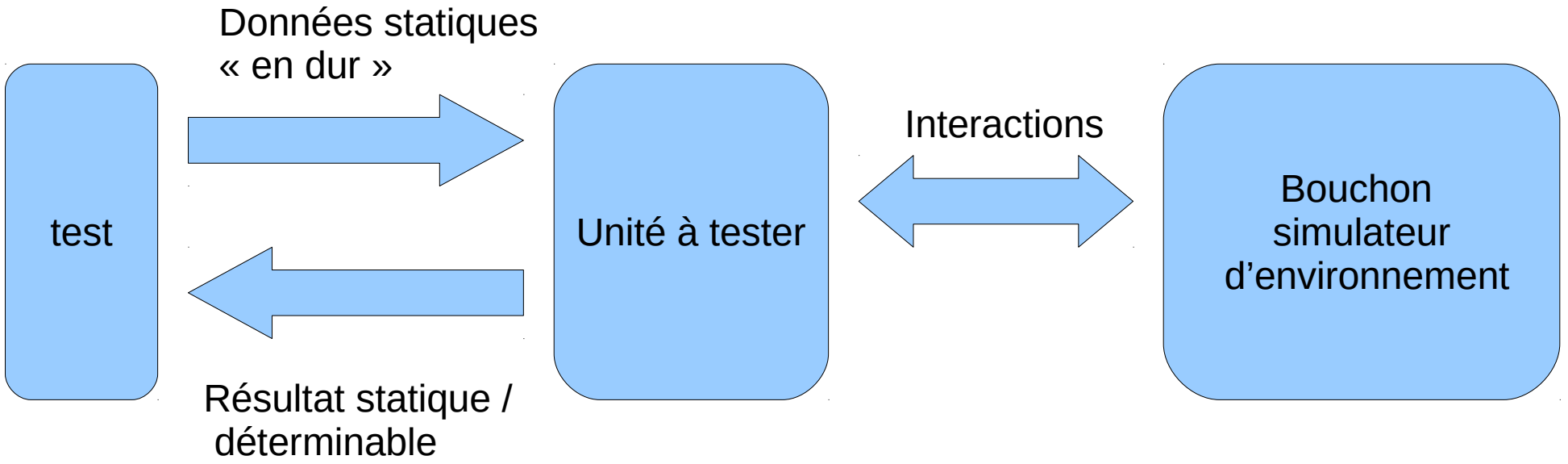
Test Unitaire

- N'est pas un test de bout en bout
- Ne doit éxécuter que l'unité à tester
- Doît être déterministe et répétable
- Ne doit pas dépendre de l'environnment
- Ne doît pas être exhaustif

Test Unitaire – Idées reçues

- Je dois mettre des données en base pour exécuter mon TU
- Je dois paramétrer mon environnement pour faire fonctionner mon TU
- Certains TU ne fonctionnent pas partout
- Mon TU est long car il fait appel à des WS
- Un seul TU par méthode suffit

Test Unitaire



Test Unitaire

- Au moins trois catégories à tester
  - Cas nominaux
    - vérifier que la méthode fait son job en fonctionnement normal
  - Cas d'erreur
    - vérifier que la methode gère les erreurs
  - Cas aux limites / peu commun
    - vérifier que la methode est robuste

Test Unitaire - Bouchon

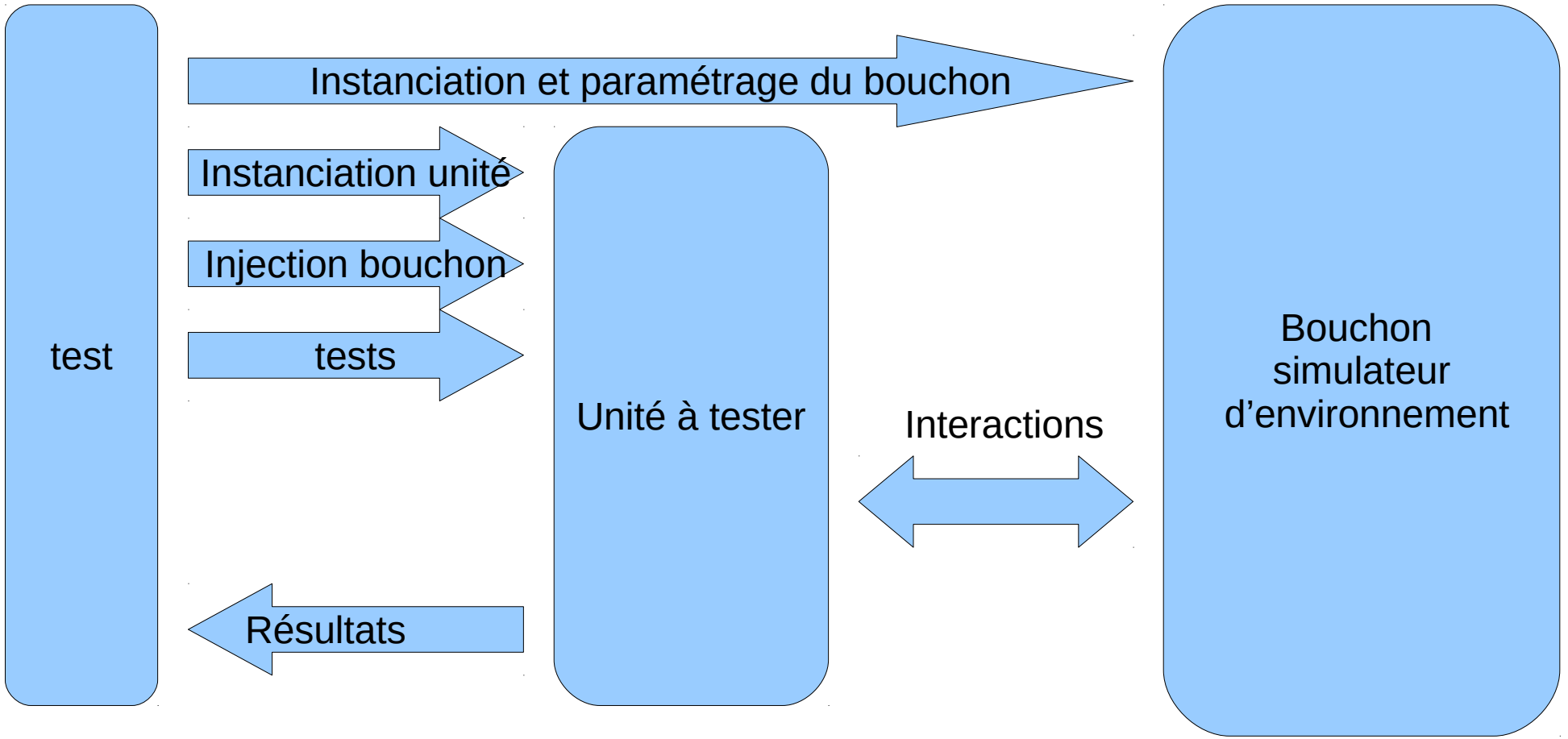
- Un Bouchon permet de
  - Renvoyer le résultat attendu
  - Renvoyer le résultat inattendu
  - Simuler une erreur inattendu
  - Simuler le fait que l'écosystème a été appelé
- Exemple
  - SMTP
  - WS
  - SGBD
  - ESB / JMS

Test Unitaire – norme écriture

test[Methode][Cas][Attendu]

- Exemples
  - testUpdateProviderCatalogForSuccessUpdateHasModifiedDb()
  - testUpdateProviderCatalogForUnexpectedProvidedErrorThrowException()

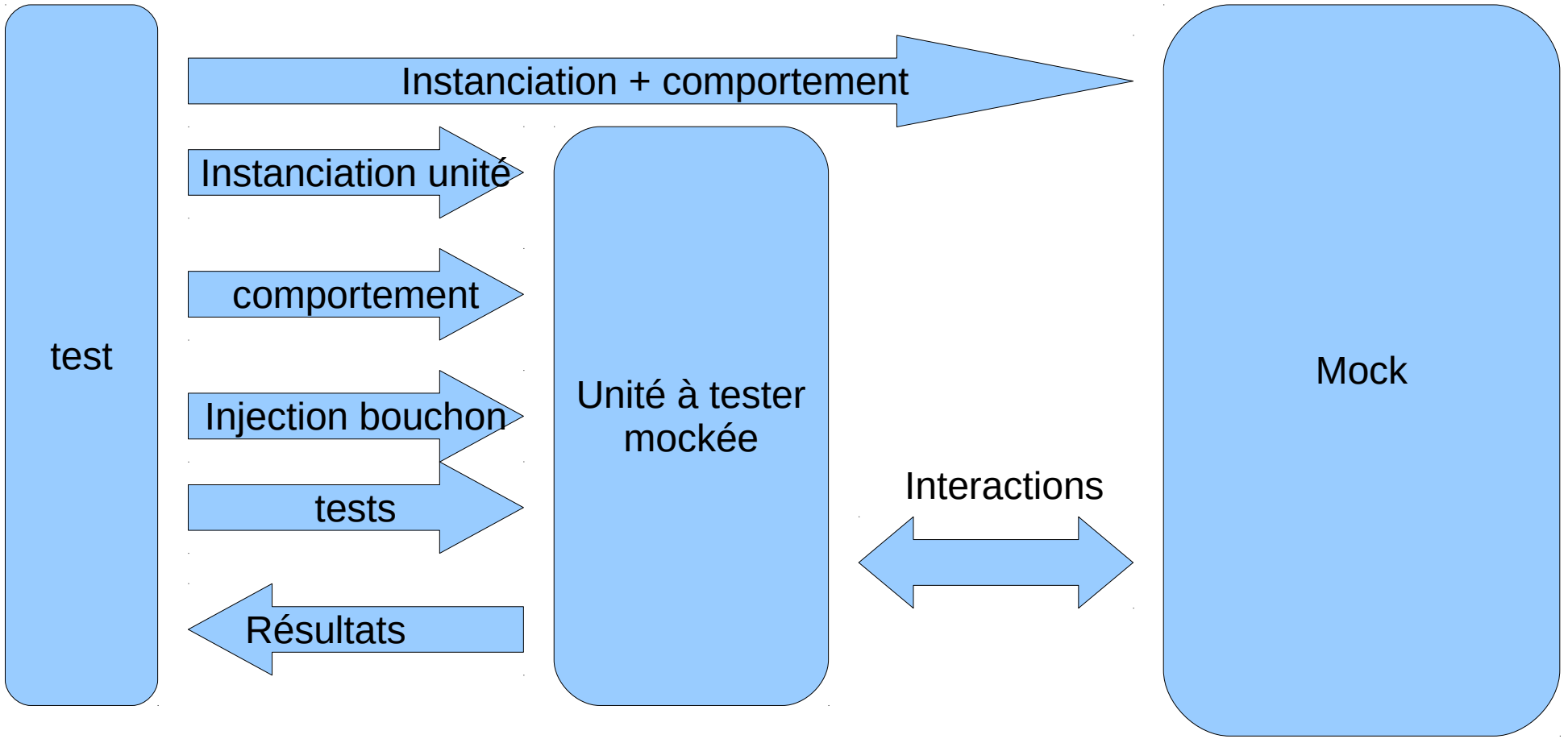
Test Unitaire - Bouchon



Mock

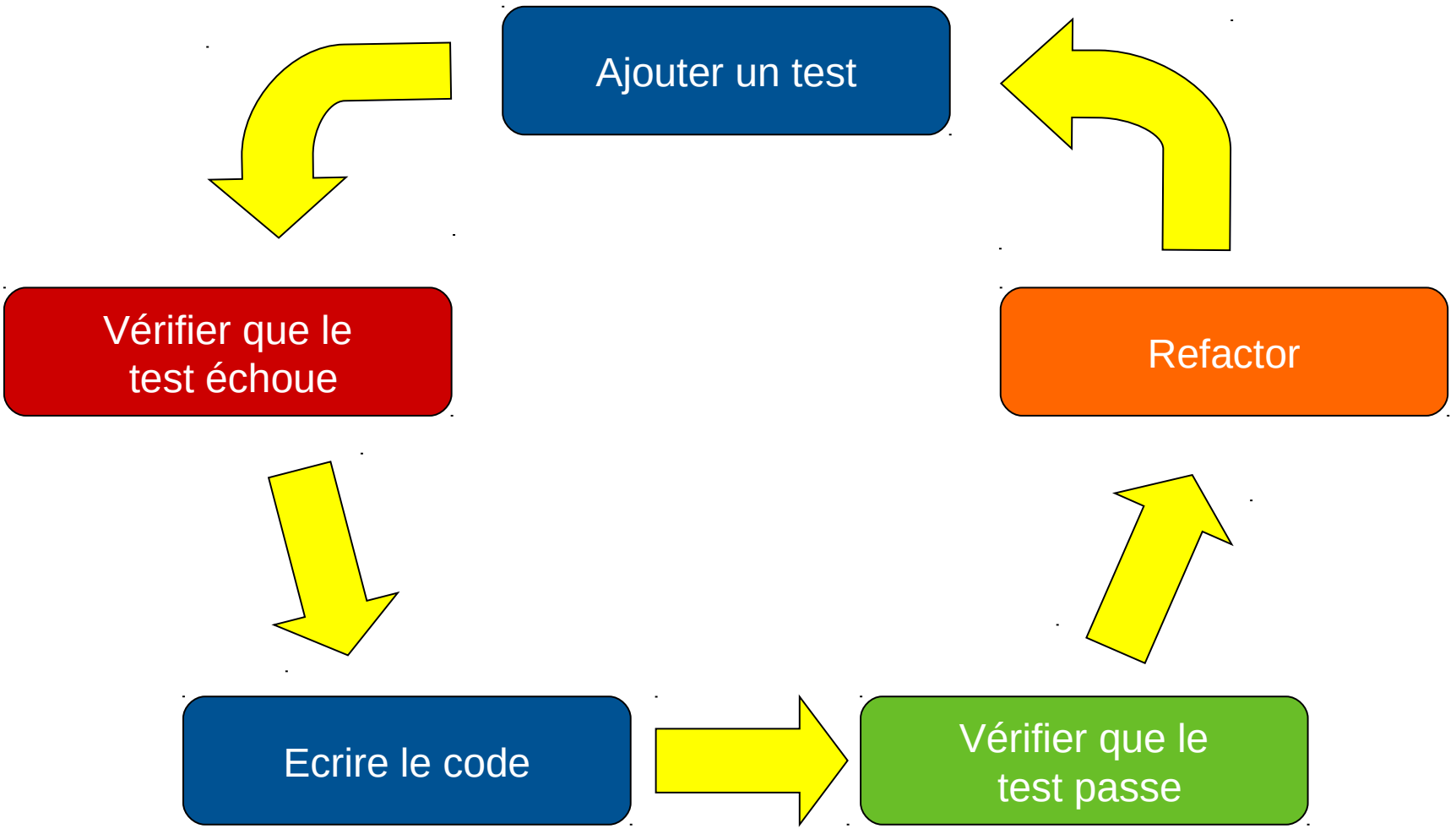
- Bouchon dont le comportement est défini par le test
- Simulacre
- Se fait passer pour ce qu'il n'est pas
- Créé à partir d'une classe
- Comportement redéfini pour les besoins du test

Test Unitaire - Mock



Test Driven Development

- Écrire les tests avant le code
- Ne pas modifier le code si il n'éxiste pas un test en error/failure
- Rajouter un test pour tout bug à fixer / fonctionnalité à ajouter
- Améliorer son code en continu


Ajouter un test
Vérifier que le test échoue
Ecrire le code
Vérifier que le test passe
Refactor

Test Driven Development - refactoring

- Le test est résolu
- On passe au suivant ? Non
- C'est le moment de refactorer
  - C'est le moment car tout les tests sont au vert
  - limite la dette technique