Tester le code

INTRO

- Cycle en V : phases de conceptions et tests

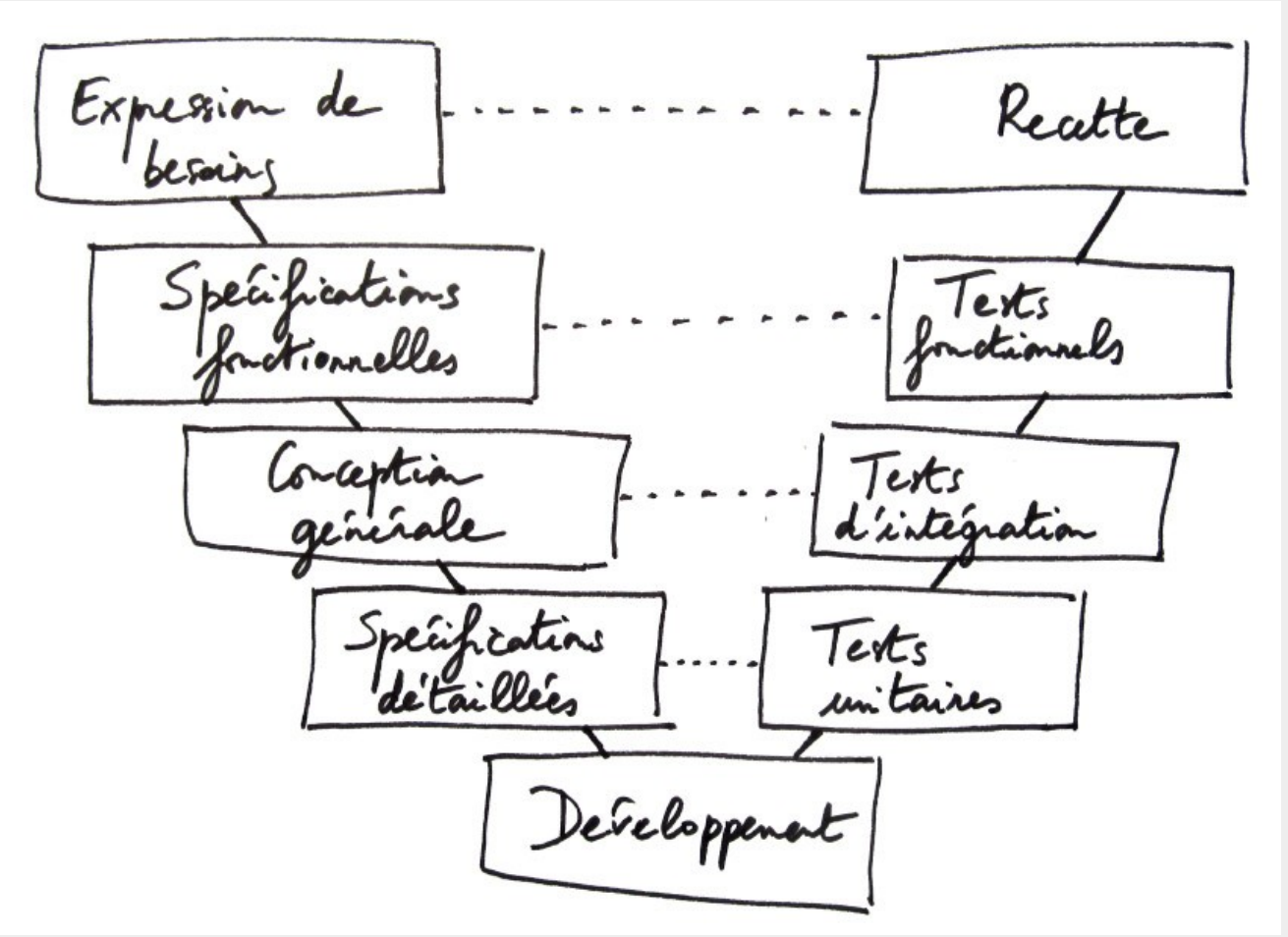
  - Langage naturel

    informel
  - Langage naturel

    formalisé, métriques
  - UML :

    diagramme de séquence
  - UML :

    diagramme de classe



TEST

- Procédure de test
  - **A**rrange : préparation du test (instanciations, connexions, descripteurs de fichier…)
  - **A**ct : le calcul à évaluer (appel de fonctions, méthodes)
  - **A**ssert : évaluation de la valeur calculée à l'aune de la valeur attendue
  - **C**leanup : libération de variables, fermeture de connexions ...

TDD

- Créer des issues sous forme de « **users stories** »
  - Une « **user story** » est une description de tâche rédigée du **point de vue de l'utilisateur final**
  - On écrit une user-story **en une seule phrase**
  - Un « **epic** » décrit une « grande user story » à décomposer en user stories
  - On cherche à respecter les critères **INVEST**
    - **I** pour Indépendante : au moins sur le sprint en cours.
    - **N** pour négociable : le contenu fait l'objet d'une **concertation**.
    - **V** pour valeur : chaque user story doit apporter de la valeur ajoutée aux clients / utilisateurs
    - **E** pour Estimable : dans le temps **« due date »** ou en complexité **« points de sprint »**
    - **S** pour Suffisamment petite : découpage fin pour livraison au sein d'un seul Sprint.
    - **T** pour Testable : On doit pouvoir **déduire un test** de l'énoncé (cf TDD ou BDD).

TDD

- Créer des issues sous forme de « **users stories** »
  - Description type utilisant le gabarit Given When Then
  - Then : critères d'acceptation qualitatif ou quantitatifs (valeurs possibles, intervalles, valeur vide/nulle)

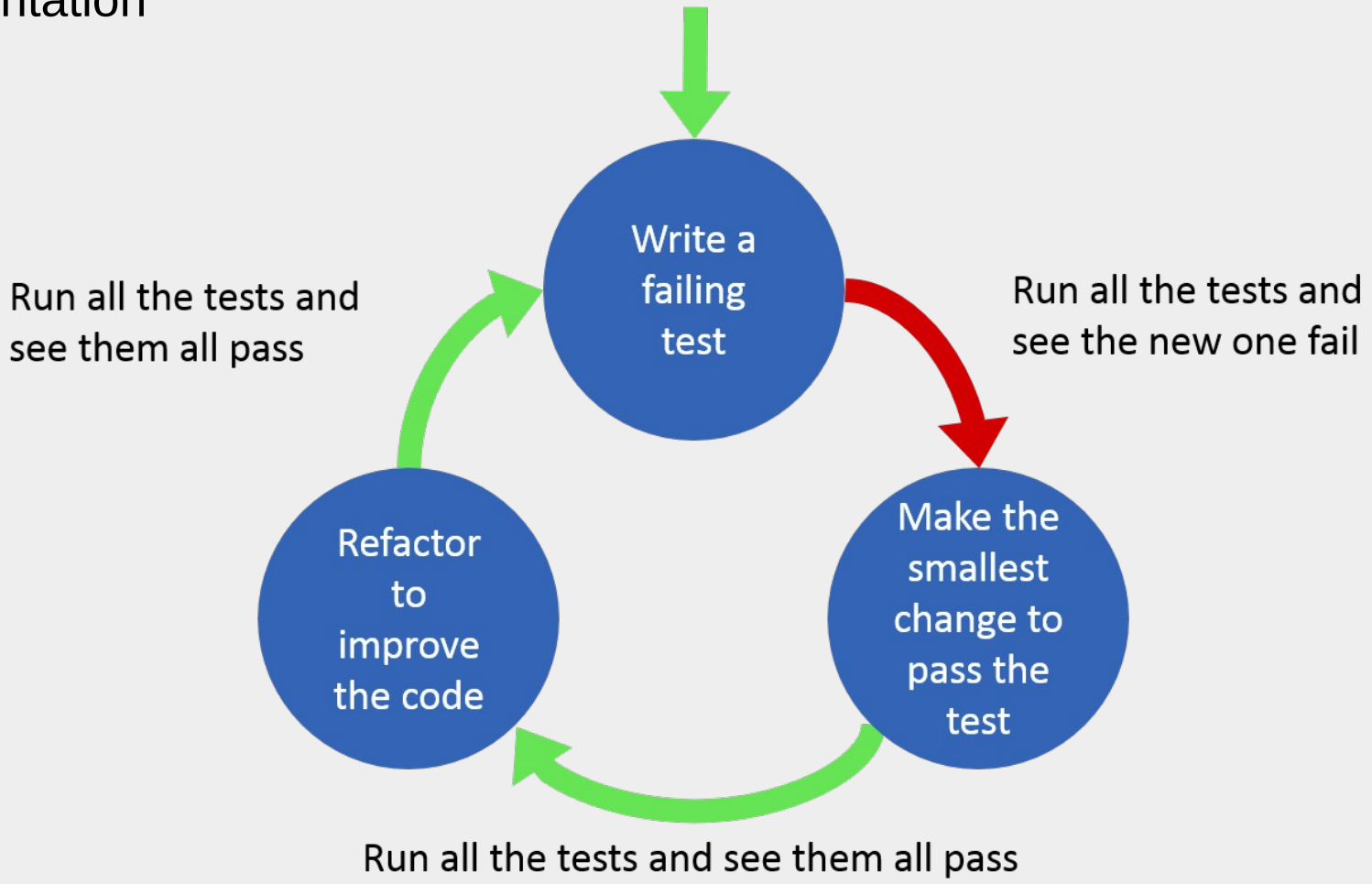
```
### Given Context ...
En tant que <utilisateur(s)>

### Action To Do ...
Je veux pouvoir <action à faire commençant pas un verbe à l'infinitif>

### Expected Result
Afin de <objectif poursuivi et justifiant l'US>
```

TDD

- Définition
  - « Test Driven Development » ou **développement piloté par les tests** :
  - emblématique d'une démarche « **shift Left** » ou décalage à gauche
    - **remontée au plus tôt** des facteurs de risques pour évaluer l'état du développement
    - US => Test => Feature != US => Feature => Test
  - Répétition de cycles de trois étapes
    - écrire un test qui échoue
    - écrire le code minimal passant le test
    - refactoriser le code
  - On exécute toujours le nouveau test avec **tous les tests concernant son périmètre**
    - pour éviter les **régressions** (bug sur l'existant d'une fonctionnalité en succès)
  - Avantages
    - des tests sont toujours disponibles pour l'intégration continue (**automatisation** des tests)
    - le développement bénéficie d'une réflexion préalable sur le périmètre de la fonctrionnalité

TDD

- Représentation



Concepts du Testing

- Sélection des tests
  - Tags, Catégories

    - permettent de **filtrer les tests** à exécuter

  - Activation / Désactivation

    - permet de gérer des tests incomplets, inachevés « **WIP** »

  - Suite

    - classe servant de **conteneur** à une collection de tests à lancer, et sélectionnés selon un package, un tag ...

Concepts du Testing

- Arrange
  - Fixture

    - Désigne toute **ressource**, et par suite toute **méthode**, fournissant le **contexte** d'un ou plusieurs tests

    - Les fixtures sont mutualisées sur plusieurs tests.

    - liées à un cycle de vie de la procédure

    - Elles doivent être libérées en fin d'exécution (connexions distantes, fichiers)

  - Mock

    - Désigne une classe **imitant** l'interface et les valeurs de retour d'un élément **particulièrement lent** nécessaire au test sans pour autant en être l'objet.

    - Le Mocking permet d'accélérer considérablement les tests

Concepts du Testing

- Assert
  - Les assertions
    - estiment la distance d'un résultat caculé par la fonctionnalité testée et le résultat attendu
    - testent tous les opérateurs de comparaisons, les valeurs booléennes, nulles ...
    - un test « **XFAIL** » inverse la logique du test ; on s'attend à un échec
    - on peut tester des **comportements particuliers**, comme la levée d'une exception, comme résultat attendu
  - Les assumptions
    - permettent de **désactiver un test à l'exécution** en fonction d'un contexte particulier

Concepts du Testing

- Dynamiser les tests
  - Tests répétés

    - permettent de tester la **stabilité** d'un résultat sur un grand nombre de répétition du même test

  - Tests paramétrés

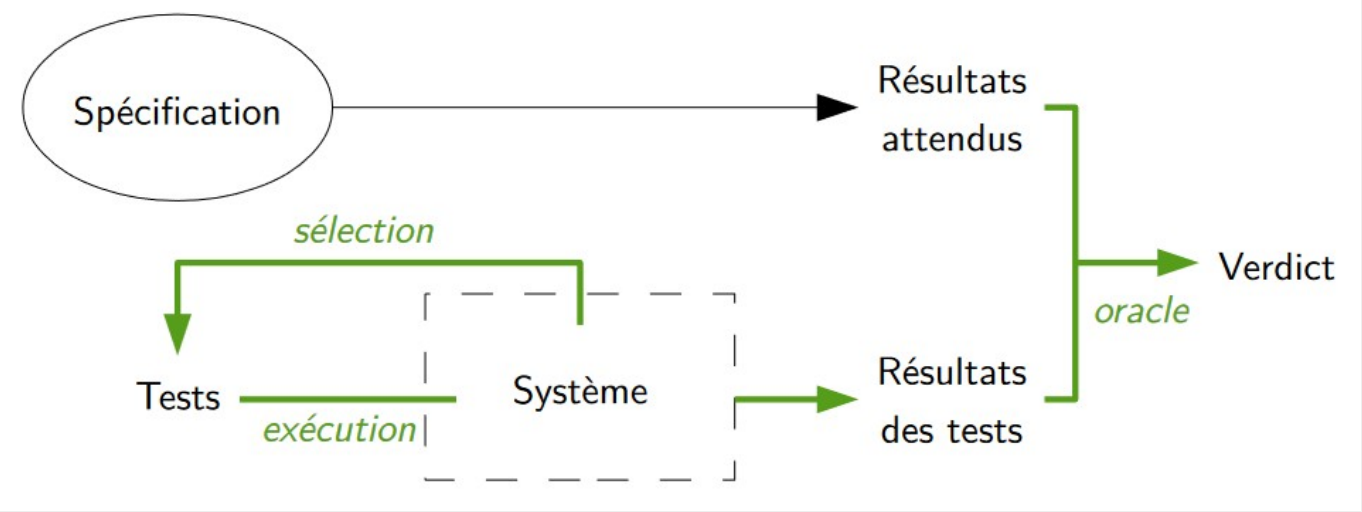
    - permettent de tester le périmètre d'une fonctionnalité en répétant l'exécution du test sur **un jeu de données** d'entrées générées de différentes manière (fonction génératrices, données formatées)

  - Tests dynamiques

    - utilisation d'une **Factory de tests**, qui fournit à l'exécution, une série du même test soumis à de multiples contextes

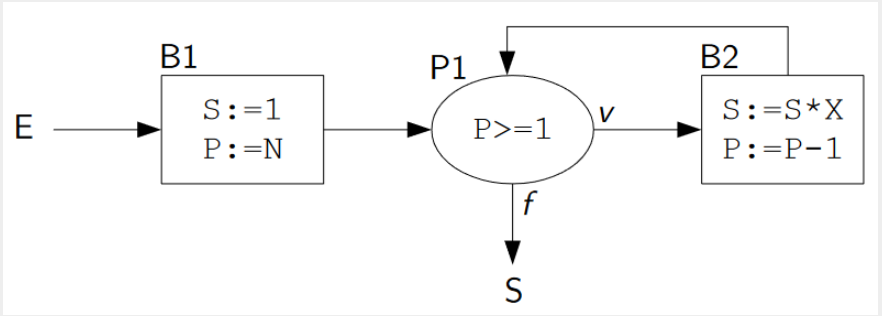
Techniques du Testing

- White box ou Structure based ou code based testing
  - Test écrit à partir du code != TDD,!= DevOps
    - Avantages : on voit les chemins possibles de la fonctionnalité
    - Désavantage : le test est écrit à partir d'une source qui ne respecte p ê pas les specs !!!
  - Techniques basé sur le logigramme du programme :
    - Etude du flux de contrôle : détermination des chemins possibles
    - Etude du flux de données : déinitions, évaluations, calculs



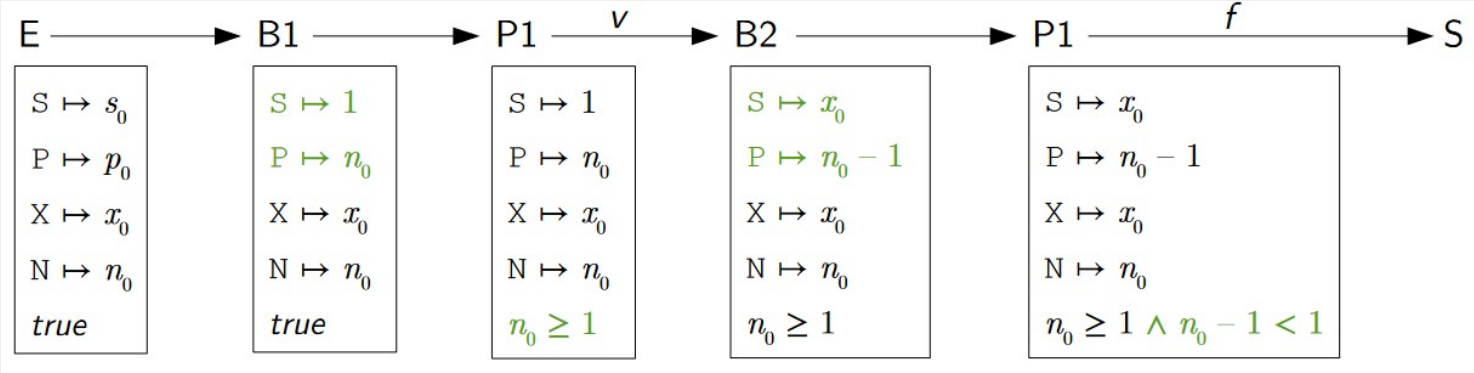
Techniques du Testing

- Test de flux de contrôle : calcul des conditions de chemins
  - On cherche à atteindre un état final dans des conditions données
  - On cherche l'équation des variables du chemin pour en déduire les valeurs



Graphe de X => X**N

Chemin : sortir en une itération



Condition de chemin

- **n0 ≥ 1 ^ n0 – 1 < 1 <=> n0 = 1**

Techniques du Testing

- Test de flux de contrôle : critères de couvertures
  - Couverture des instructions :

    - on cherche les conditions de chemins qui passent par tous les nœuds du logigramme

  - Couverture des conditions

    - // // // qui rendent chaque décision tour à tour vrai et fausse

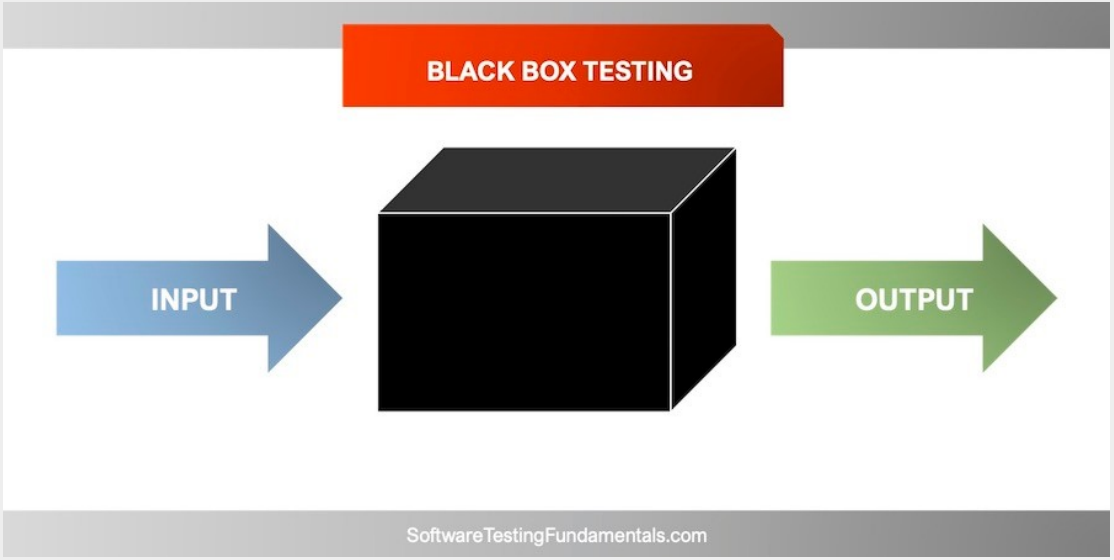
    - tables de décision des conditions plus ou moins complexes

    - ex : A ^ (B | C)

A	B	C	décision	
~~0~~	~~0~~	~~0~~	~~0~~	
~~0~~	~~0~~	~~1~~	~~0~~	
~~0~~	~~1~~	~~0~~	~~0~~	
0	1	1	0	→ A
1	0	0	0	→ B et C
1	0	1	1	→ C
1	1	0	1	→ B
1	1	1	1	→ A

Techniques du Testing

- Black box testing
  - Test écrits sans connaître le code
  - Tests basés sur des specs fonctionnelles ou techniques
  - Techniques associées :
    - partitions d’équivalences
    - analyse des valeurs limites
    - diagrammes transitions états



Techniques du Testing

Partition d'équivalences

- Analyser les **intervalles de valeurs valides et non valides** de toutes les entrées de la spec
  - cet ensemble d'intervalles forment une **partition** (E1V1, E1V2, …, E1I1, E1I2...)
- Analyser les valeurs possibles des **sortie** dans la spec (dont erreur) (S1i, S1j, S2i, ….)
- Déduire des cas de **tests paramétrés** en croisant les distributions d'entrées

Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
E1I1	E1V1	E1V2	E1I2	E1I3	E1I4
-1000	5	50k	150k	50,5	cinq
E2I2	E2I1	E2V1	E2I3	E2I4	E2I5
-10	1	6	11	1,5	six
S1 S5	S2 ?	S3 S4	S1 S5	S1 S5	S1 S5
Erreur	0%	20%	Erreur	Erreur	Erreur
Erreur	?	15%	Erreur	Erreur	Erreur

Techniques du Testing

- Analyse des valeurs limites
  - Définir les valeurs limites des intervalles de partitions
    - en prenant la première valeur au-delà (analyse à 2 valeurs) et en deçà (analyse à trois valeurs)
    - on approxime les valeurs infinies par une valeur finie très grande en regard de la limite
  - Déduire des cas de **tests paramétrés** en croisant les limites d'entrées

E1	E2	S1	S2
$-10^{12}-1$	-101	Erreur	Erreur
-10^{12}	-100	Erreur	Erreur
-10^{12+1}	-99	Erreur	Erreur
-1	-1	Erreur	Erreur
0	0	0%	?
1	1	0%	?
9 999	2	0%	15%
10 000	3	20%	15%