Module 2 : Stratégies et Techniques de Test (4h)

Objectifs

- Comprendre les différentes techniques de test (boîte blanche, boîte noire)
- Savoir concevoir des cas de test efficaces
- Mettre en place un plan de test structuré
- Implémenter des tests avec Python

1. Techniques de test boîte blanche

Définition et principes

Les tests **boîte blanche** sont des tests basés sur la structure interne du code. Ils permettent de s'assurer que toutes les parties du code sont bien couvertes.

Types de couverture de code

- **Couverture des instructions** : Vérifie que chaque ligne de code est exécutée au moins une fois.
- **Couverture des branches** : Vérifie que chaque branche (`if`, `else`, `while`, etc.) est testée.
- **Couverture des conditions** : Vérifie que chaque condition logique (`&&`, `||`) est évaluée avec toutes les combinaisons possibles.
- **Couverture des chemins** : Vérifie que toutes les combinaisons de chemins dans le programme sont explorées.

Exemple en Python : Couverture des instructions et des branches

```
import unittest

def division(a, b):
    if b == 0:
        raise ValueError("Division par zéro impossible")
    return a / b

class TestDivision(unittest.TestCase):
    def test_division_normale(self):
        self.assertEqual(division(10, 2), 5)

    def test_division_par_zero(self):
        with self.assertRaises(ValueError):
            division(10, 0)

if __name__ == '__main__':
    unittest.main()
```

Ce test couvre les instructions et les branches `if` et `else`.

2. Techniques de test boîte noire

Définition et principes

Les tests **boîte noire** sont basés sur les entrées et sorties du programme **sans connaître son code interne**.

Principales techniques

- **Partition d'équivalence** : Diviser l'ensemble des entrées en groupes équivalents et tester un échantillon représentatif.
- **Analyse des valeurs limites** : Tester les valeurs extrêmes (exemple : `0`, `-1`, `max_int`).
- **Tables de décision** : Analyser toutes les combinaisons possibles d'entrées pour vérifier le comportement attendu.
- **Transitions d'état** : Tester les changements d'état d'un système (utile pour les applications web et mobiles).

Exemple en Python : Partition d'équivalence et analyse des valeurs limites

```
def est_adulte(age):
    return age >= 18

class TestAdulte(unittest.TestCase):
    def test_moins_de_18(self):
        self.assertFalse(est_adulte(17))

    def test_18_ans(self):
        self.assertTrue(est_adulte(18))

    def test_plus_de_18(self):
        self.assertTrue(est_adulte(25))

if __name__ == '__main__':
    unittest.main()
```

Ce test vérifie les partitions d'équivalence (`<18`, `>=18`) et les valeurs limites (`18`).

3. Conception des cas de test

Éléments d'un bon cas de test

- **Identifiant unique**
- **Description claire**
- **Entrées (données, paramètres)**
- **Conditions préalables**
- **Résultat attendu**
- **Statut du test (succès/échec)**

Exemple de tableau de test

ID	Cas de test	Entrées	Résultat attendu	Statut
TC01	Vérifier la division normale	`10, 2`	`5`	✅
TC02	Vérifier la division par zéro	`10, 0`	`Erreur`	✅
TC03	Vérifier si un utilisateur est adulte	`17`	`False`	✅
TC04	Vérifier si un utilisateur est adulte	`18`	`True`	✅

4. Rédaction de plans de test et scénarios de test

Plan de test

Un plan de test est un document qui décrit **quoi tester, comment tester, et avec quels outils**.

Exemple de plan de test simplifié

- **Objectif** : Tester la fonctionnalité de connexion d'un site web
- **Périmètre** : Authentification des utilisateurs
- **Critères de succès** :
  - L'utilisateur peut se connecter avec des identifiants valides
  - Un message d'erreur apparaît en cas d'identifiants invalides
  - La connexion est sécurisée

5. Gestion des exigences et des priorités

Pourquoi prioriser les tests ?

- **Tests critiques en premier** : Ceux qui impactent la sécurité et la stabilité.
- **Tests des fonctionnalités principales avant les secondaires.**

Méthode de priorisation : Matrice de risque

Fonctionnalité	Probabilité d'échec	Impact	Priorité
Connexion utilisateur	Élevée	Critique	Haute
Paiement en ligne	Moyenne	Très critique	Haute
Page de contact	Faible	Mineur	Moyenne

6. Travaux Pratiques (TP) : Plan de test et implémentation en Python

Objectifs du TP

Rédiger un plan de test basé sur un projet existant
Implémenter les tests en Python avec `unittest` et `pytest`

Exercice pratique : Tester une API en Python

1. Installer `pytest` :

```
pip install pytest
```

2. Écrire un test pour une API simple avec `Flask`

```
from flask import Flask, jsonify
import pytest

app = Flask(__name__)

@app.route('/status')
def status():
    return jsonify({"status": "ok"}), 200

def test_status():
    with app.test_client() as client:
        response = client.get('/status')
        assert response.status_code == 200
        assert response.json["status"] == "ok"

if __name__ == '__main__':
    pytest.main()
```

3. Exécuter le test :

```
pytest test_api.py
```

Vérification automatique du statut HTTP et du JSON retourné.

Résumé du Module 2

- **Techniques de test boîte blanche** (basé sur le code)
- **Techniques de test boîte noire** (basé sur les entrées et sorties
- **Conception de cas de test efficaces**
- **Écriture de plans et scénarios de test**
- **Mise en pratique avec Python (`unittest`, `pytest`)**