**Test Driven Development ou le développement piloté par les tests en Java**

**Objectifs de formation**

A l'issue de cette formation, vous serez capable de :

* Mettre en œuvre les principes fondamentaux et les bonnes pratiques du TDD
* Utiliser le Framework de test JUnit dans une approche TDD
* Mettre en œuvre du TDD en présence de code hérité (legacy)
* Pratiquer le refactoring d'un code développé en TDD.

**Modalités, méthodes et moyens pédagogiques**

Formation délivrée en présentiel.

Le formateur alterne entre méthode démonstrative, interrogative et active (via des travaux pratiques et/ou des mises en situation).

Les moyens pédagogiques mis en œuvre sont :

* Ordinateurs PC , connexion internet fibre, tableau blanc ou paperboard, vidéoprojecteur
* Environnements de formation installés sur les postes de travail ou en ligne
* Supports de cours et exercices

**Prérequis**

Avoir une pratique professionnelle de Java.

**Public concerné**

Développeurs, responsables tests, chefs de projets, responsables qualité.

**Programme**

**Le test dans le processus de développement**

* Processus, qualité, tests
* Types de tests
* Architectures de test
* TDD et Extreme Programming (XP)

**Frameworks de tests automatisés**

* Les frameworks de test
* Bonnes pratiques associées aux frameworks de test

**Principes fondamentaux du TDD**

* Le cycle de développement du TDD
* Stratégies de Test First
* Stratégies pour faire passer les tests

**Ecrire du code testable**

* Composition plutôt qu'héritage
* Eviter le code statique
* Inverser les dépendances

**Eléments mis à la disposition du rédacteur de tests**

* Assertions
* Tests d'échec
* Tests paramétrés sur les types et les valeurs
* Les matchers

**Techniques pour organiser les tests de codes complexes**

* Organisation des classes de test
* Organisation du code de test

**Le refactoring en TDD**

* Quelques "mauvaises odeurs"
* Techniques de refactoring en TDD
* Les design patterns

Le contenu de ce programme peut faire l'objet d'adaptation selon les niveaux, prérequis et besoins des apprenants.

**Modalités d’évaluation des acquis**

* En cours de formation, par des études de cas ou des travaux pratiques
* Et, en fin de formation, par un questionnaire d'auto-évaluation

**Jour 1 (7 heures)**

**1. Le test dans le processus de développement (4 heures)**

1.1 Processus, qualité, tests (1 heure)

* **Théorie** (30 minutes)
  + Introduction au cycle de vie du développement logiciel
  + Rôle des tests à chaque phase
  + Importance de la qualité logicielle
* **TP1** (30 minutes)
  + Étude de cas : Identifier les phases de tests dans un projet logiciel.

1.2 Types de tests (1 heure)

* **Théorie** (30 minutes)
  + Tests unitaires, tests d'intégration, tests de système, tests d'acceptation
* **TP2** (30 minutes)
  + Exercice : Classifier des tests existants dans les différentes catégories.

1.3 Architectures de test (1 heure)

* **Théorie** (30 minutes)
  + Architectures de test en couches, mocking et stubbing, frameworks de test populaires
* **TP3** (30 minutes)
  + Implémenter une architecture de test simple avec des mocks.

1.4 TDD et Extreme Programming (XP) (1 heure)

* **Théorie** (30 minutes)
  + Introduction à TDD et XP, valeurs et principes de XP, pratiques de TDD
* **TP4** (30 minutes)
  + Mise en pratique : Écrire des tests unitaires basiques en utilisant TDD.

**2. Frameworks de tests automatisés (3 heures)**

2.1 Les frameworks de test (1.5 heures)

* **Théorie** (45 minutes)
  + Introduction aux frameworks de test, JUnit : installation et configuration, introduction à Mockito
* **TP5** (45 minutes)
  + Configuration d’un projet Maven avec JUnit et Mockito, écriture des premiers tests.

2.2 Bonnes pratiques associées aux frameworks de test (1.5 heures)

* **Théorie** (45 minutes)
  + Structurer ses tests, nommer correctement ses tests, assurer la maintenance des tests
* **TP6** (45 minutes)
  + Réorganisation d’un ensemble de tests existants selon les bonnes pratiques.

**Jour 2 (7 heures)**

**3. Principes fondamentaux du TDD (4 heures)**

3.1 Le cycle de développement du TDD (2 heures)

* **Théorie** (1 heure)
  + Red, Green, Refactor, écrire des tests simples, faire passer les tests
* **TP** (1 heure)
  + Implémentation d’une fonctionnalité simple en utilisant le cycle TDD.

3.2 Stratégies de Test First (1 heure)

* **Théorie** (30 minutes)
  + Approche Test First vs Code First, techniques pour écrire les premiers tests
* **TP** (30 minutes)
  + Écrire des tests pour une nouvelle fonctionnalité avant d’implémenter le code.

3.3 Stratégies pour faire passer les tests (1 heure)

* **Théorie** (30 minutes)
  + Faire échouer le test, implémenter juste assez de code pour faire passer le test, refactoriser le code
* **TP** (30 minutes)
  + Pratique : Faire échouer, faire passer et refactoriser des tests pour une fonctionnalité donnée.

**4. Écrire du code testable (3 heures)**

4.1 Composition plutôt qu'héritage (1 heure)

* **Théorie** (30 minutes)
  + Avantages de la composition, exemples pratiques
* **TP** (30 minutes)
  + Refactoriser une classe pour utiliser la composition plutôt que l'héritage.

4.2 Éviter le code statique (1 heure)

* **Théorie** (30 minutes)
  + Problèmes avec le code statique, solutions alternatives
* **TP** (30 minutes)
  + Refactoriser du code statique pour le rendre plus testable.

4.3 Inverser les dépendances (1 heure)

* **Théorie** (30 minutes)
  + Principe d'inversion de dépendance, utilisation de l'injection de dépendances
* **TP** (30 minutes)
  + Implémenter l'inversion de dépendances dans un projet existant.

**Jour 3 (7 heures)**

**5. Éléments mis à la disposition du rédacteur de tests (3 heures)**

5.1 Assertions (45 minutes)

* **Théorie** (20 minutes)
  + Types d'assertions, utilisation des assertions dans les tests
* **TP** (25 minutes)
  + Écrire des tests en utilisant différentes assertions.

5.2 Tests d'échec (45 minutes)

* **Théorie** (20 minutes)
  + Importance des tests d'échec, écrire des tests pour les cas d'échec
* **TP** (25 minutes)
  + Écrire des tests pour des scénarios de panne.

5.3 Tests paramétrés sur les types et les valeurs (45 minutes)

* **Théorie** (20 minutes)
  + Utilisation des tests paramétrés, exemples pratiques avec JUnit
* **TP** (25 minutes)
  + Implémenter des tests paramétrés.

5.4 Les matchers (45 minutes)

* **Théorie** (20 minutes)
  + Introduction aux matchers, utilisation des matchers dans les tests
* **TP** (25 minutes)
  + Utiliser les matchers pour écrire des tests plus lisibles.

**6. Techniques pour organiser les tests de codes complexes (2 heures)**

6.1 Organisation des classes de test (1 heure)

* **Théorie** (30 minutes)
  + Structurer les classes de test, utilisation des suites de tests
* **TP** (30 minutes)
  + Réorganiser un projet de tests en classes de tests structurées.

6.2 Organisation du code de test (1 heure)

* **Théorie** (30 minutes)
  + Bonnes pratiques pour structurer le code de test, réutilisation du code de test
* **TP** (30 minutes)
  + Refactoriser le code de test pour le rendre plus maintenable.

**7. Le refactoring en TDD (1 heure)**

7.1 Quelques "mauvaises odeurs" (30 minutes)

* **Théorie** (15 minutes)
  + Identifier les mauvaises odeurs dans le code, exemples pratiques
* **TP** (15 minutes)
  + Identifier et corriger les mauvaises odeurs dans un projet donné.

7.2 Techniques de refactoring en TDD (30 minutes)

* **Théorie** (15 minutes)
  + Techniques de refactoring courantes, exemples pratiques
* **TP** (15 minutes)
  + Appliquer les techniques de refactoring sur un code existant.

**8. Design Patterns (1 heure)**

8.1 Introduction aux design patterns (15 minutes)

* **Théorie** (10 minutes)
  + Importance des design patterns, catégories de design patterns
* **TP** (5 minutes)
  + Discussion sur des exemples concrets de design patterns.

8.2 Exemples de design patterns (45 minutes)

* **Théorie** (20 minutes)
  + Singleton, Factory, Observer, Strategy, Decorator, implémentations en Java
* **TP** (25 minutes)
  + Implémenter et refactoriser une fonctionnalité en utilisant des design patterns.