Date de publication : 17/02/2026

Formation : Clean Code

Formation pratique - 2j - 14h00 - Réf. CCJ
Prix : 1500 CHF H.T.

NEW

Ce cours vise à transformer votre approche du développement autour de deux piliers fondamentaux : la qualité et la sécurité du code. De la production d'un code clair et efficace à la structuration de fonctions simples et cohérentes, vous verrez comment produire un code compréhensible par tous.

Objectifs pédagogiques

À l'issue de la formation, le participant sera en mesure de :

- Évaluer et réduire la dette technique
- Produire un code lisible et expressif
- Structurer des fonctions simples et cohérentes
- Appliquer concrètement les principes SOLID
- Mettre en œuvre une stratégie de tests efficace
- Refactorer du code legacy en toute sécurité
- Industrialiser la qualité du code
- Intégrer l'IA comme assistant de développement

Public concerné

Développeurs en programmation objet.

Prérequis

Expérience significative en programmation, idéalement avec un langage orienté objet. Bonne aisance avec un IDE et une connaissance de base des tests unitaires

Méthodes et moyens pédagogiques

Travaux pratiques

Analyse, échange, travaux pratiques.

Modalités d'évaluation

Le formateur évalue la progression pédagogique du participant tout au long de la formation au moyen de QCM, mises en situation, travaux pratiques…

Le participant complète également un test de positionnement en amont et en aval pour valider les compétences acquises.

Programme de la formation

PARTICIPANTS

Développeurs en programmation objet.

PRÉREQUIS

Expérience significative en programmation, idéalement avec un langage orienté objet. Bonne aisance avec un IDE et une connaissance de base des tests unitaires

COMPÉTENCES DU FORMATEUR

Les experts qui animent la formation sont des spécialistes des matières abordées. Ils ont été validés par nos équipes pédagogiques tant sur le plan des connaissances métiers que sur celui de la pédagogie, et ce pour chaque cours qu'ils enseignent. Ils ont au minimum cinq à dix années d'expérience dans leur domaine et occupent ou ont occupé des postes à responsabilité en entreprise.

MODALITÉS D'ÉVALUATION

Le formateur évalue la progression pédagogique du participant tout au long de la formation au moyen de QCM, mises en situation, travaux pratiques…

Le participant complète également un test de positionnement en amont et en aval pour valider les compétences acquises.

1 Du codeur à l'artisan logiciel

- La dette technique : mesurer le coût réel d'un code défaillant.
- La règle du boy scout : laisser le code plus propre qu'on ne l'a trouvé.
- L'impact de l'IA : comment ChatGPT et Copilot changent la donne.

Travaux pratiques

Analyse collective d'un snippet de code "sale". Identification des code smells et réécriture assistée par IA pour comparer les résultats.

2 L'art du nommage et de la clarté

- L'intention avant tout : choisir des noms qui révèlent le "pourquoi" et non le "comment".
- Chasser l'ambiguïté : éviter les mappings mentaux et les noms imprononçables.
- Conventions modernes : nommage des classes et méthodes à l'ère des frameworks modernes.

Travaux pratiques

Refactoring de variables et fonctions cryptiques (ex: d, list1, proc()) en concepts métier clairs.

3 Fonctions narratives et complexité

- Single Responsibility Principle (fonctions) : une fonction ne doit faire qu'une seule chose.
- Niveaux d'abstraction : ne pas mélanger le haut niveau (métier) et le bas niveau (détails techniques).
- Arguments et effets de bord : pourquoi éviter les drapeaux (boolean flags) et limiter les arguments.
- Command Query Separation (CQS) : séparer les commandes des requêtes.

Travaux pratiques tutorés

Prendre une "God Function" de 100 lignes et la découper en 10 fonctions atomiques et lisibles.

4 Architecture SOLID appliquée

- SRP (Responsabilité Unique) : cohésion des classes et modules.
- OCP (ouvert/fermé) : étendre sans modifier (l'usage du polymorphisme).
- LSP & ISP : la substitution de Liskov et la ségrégation des interfaces pour éviter les "Fat Interfaces".
- DIP (Inversion de dépendance) : le découplage fort pour faciliter le test.
- DRY (Don't Repeat Yourself) : nuance entre duplication de code et duplication de connaissance.

Travaux pratiques

Refactoring d'un système de notification violant l'OCP et le DIP pour le rendre extensible via injection de dépendances.

MOYENS PÉDAGOGIQUES ET TECHNIQUES

• Les moyens pédagogiques et les méthodes d'enseignement utilisés sont principalement : aides audiovisuelles, documentation et support de cours, exercices pratiques d'application et corrigés des exercices pour les formations pratiques, études de cas ou présentation de cas réels pour les séminaires de formation.
• À l'issue de chaque formation ou séminaire, ORSYS fournit aux participants un questionnaire d'évaluation du cours qui est ensuite analysé par nos équipes pédagogiques.
• Une feuille d'émargement par demi-journée de présence est fournie en fin de formation ainsi qu'une attestation de fin de formation si le participant a bien assisté à la totalité de la session.

MODALITÉS ET DÉLAIS D'ACCÈS

L'inscription doit être finalisée 24 heures avant le début de la formation.

ACCESSIBILITÉ AUX PERSONNES HANDICAPÉES

Pour toute question ou besoin relatif à l'accessibilité, vous pouvez joindre notre équipe PSH par e-mail à l'adresse psh-accueil@orsys.fr.

5 Test Driven Development (TDD)

- Le cycle Red-Green-Refactor : la respiration du développeur.
- Qualité des tests (FIRST): Fast, Independent, Repeatable, Self-Validating, Timely.
- Outillage moderne : focus sur les standards actuels (JUnit 5, Jest, Pytest).
- Mocking versus stubbing : utilisation efficace de Mockito ou Jest Mocks.

Travaux pratiques
en groupe : "TDD Ping-Pong" Par binôme : l'un écrit le test (Red), l'autre écrit le code (Green) puis refactorisation commune.

6 Gérer le Legacy Code (nouveau module)

- Définition du Legacy : code sans tests.
- Techniques de rupture de dépendances : comment tester l'intestable.
- Le refactoring sécurisé : utiliser l'IDE pour refactorer sans casser. Technique du "Sprout Method/Class".

Travaux pratiques
"Gilded Rose Kata" : mise en situation réelle. Ajouter une feature dans un code existant complexe et non testé. Sécurisation préalable par "Golden Master Testing".

7 Clean Code à l'échelle (outillage & IA)

- Linter le code automatiquement (ESLint, Checkstyle).
- SonarQube/Checkmarx/SonarLint : détecter la dette technique et les failles de sécurité en temps réel.
- Code Coverage : mythes et réalités de la couverture de code.
- Utiliser l'IA pour générer de la documentation (JSDoc/JavaDoc) pertinente.
- Prompter efficacement avec l'IA pour générer des tests unitaires.

Travaux pratiques
"Pipeline Qualité". Configuration d'un set de règles strictes dans un Linter. Analyse d'un projet via SonarLint et correction des "blocker issues".

Dates et lieux

CLASSE À DISTANCE
2026 : 15 juin, 17 sep., 7 déc.