Mettre en œuvre les Design Patterns dans vos applications

Objectifs	Ce stage vous formera au design des applications et aux pratiques de conception modernes telles que le développement guidé par les tests et le refactoring. Les nombreux cas pratiques vous apprendront à créer des applications évolutives et réutilisables en prenant en compte les principaux patterns de conception.
Participants	Concepteur Développeur. Architecte. Chef de projet.
Prérequis	Connaissance d'un langage objet. Les ateliers réalisés par les stagiaires seront effectués avec le langage de leur choix (C++, JAVA, C# ou VB.NET).
Moyens pédagogiques	1 poste par participant - 1 Vidéo projecteur - Support de cours fourni à chaque participant – Formation présentielle
Durée	5 jours

Code : PATTERNS-APP

Programme.

Présentation du design
Rappel des fondamentaux de la POO et d'UML.
Les apports d'UML pour la conception.
Les enjeux de la conception.
L'utilisation de l'héritage. Avantages et inconvénients.

Principes fondamentaux en conception objet
Les principes d'ouverture/fermeture (OCP) et de substitution de Liskov (LSP).
Concept de polymorphisme, de couplage faible et de forte cohésion.
L'impact de la conception objet sur les projets.

Principes de construction des classes
La gestion des dépendances avec l'inversion de dépendance (DIP).
La réduction de la complexité apparente par la séparation des interfaces (ISP).
La répartition des responsabilités avec le GRASP.

Principes d'organisation en packages
Le package : une unité de conception livraison/réutilisation (REP) et la réutilisation commune (CRP).
Le découpage des packages. Le CCP.
L'organisation entre packages.

Développements pilotés par les tests
Approche Test Driven Development (TDD) versus approche Model Driven Engineering (MDE).
Ecriture des cas et de suites de tests.

Principes des design patterns
Les design patterns pour réutiliser l'expérience.
Périmètre, avantages et limites des design patterns.
Répondre à des problèmes récurrents.
Les patterns fondateurs de Gamma et GOF : les patterns de création, de comportement, de structure.
La refactorisation. Pourquoi refactoriser ?
Modification de la présentation du code et de l'algorithmique des classes. Refonte de la conception.

Architecture logicielle et patterns architecturaux
Des exigences à l'architecture.
Styles architecturaux.
Patterns de distribution (Client / Serveur Style, Data Bus Pattern, Blackboard, Repository).
Patterns de conception de systèmes (MVC, architecture en couches, Plug-in Style, Pipeline).

Processus de développement
Concevoir dans un processus itératif et incrémental.
Le manifeste Agile. XP, SCRUM.