Introduction au Génie Logiciel

Séance 6 : Bonnes pratiques de programmation et DevOps
Compléments

L. Laversa
laversa@irif.fr

Université Paris Cité

. Remerciements à E. Bigeon et J. Lefebvre

Pratiques SOLID [1]

Bonnes pratiques en programmation orientée objet

Objectifs

Écrire du code maintenable, évolutif et compréhensible.
Favoriser une conception modulaire et flexible

Contexte

Très à propos de chaque ligne de code, mais améliorent l'ensemble par extension

1. par Robert C. Martin

Pratiques SOLID

5 principes :

Single Responsability Principle

Open/Closed Principle

Liskov Substitution Principle

Inferface Segregation Principle

Dependency Inversion Principle

Single Responsability Principle

> Une classe (ou fonction, ou méthode) doit avoir une et une seule raison d'être modifiée. Elle ne doit avoir qu'un seul but.

Objectif : Favorise la modularité et facilite la maintenance en évitant les classes avec trop de responsabilités. Si plusieurs fonctionnalités sont gérées dans ma classe, il est plus compliqué d'en corriger une sans en casser une autre.

Exemple

Une commande Linux a un but, et on a confiance en ce comportement, la composition se fait avec des pipes.

Open/Closed Principle

> Une entité doit être fermée à la modification directe mais ouverte à l'extension.

Objectif : Permettre l'ajout de nouvelles fonctionnalités sans altérer le code existant. Anticiper les potentiels ajouts.

Exemple

Système de gestion de fiche de paie. Si on veut ajouter une méthode pour calculer les primes, on ne veut pas modifier la méthode qui calcule le salaire. On s'assure que la façon de calculer les salaires permettra d'ajouter des primes, des remboursements, etc.

Liskov Substitution Principle

> Une instance d'une classe enfant doit pouvoir être remplacée par une instance de la classe parent, sans que cela ne modifie la cohérence du programme.

Objectif : Éviter des incohérences dans l'héritage et diviser les responsabilités.

Exemple

Classe parent : Rectangle, classe enfant : Carré. Si seul le rectangle peut être colorié, la méthode ne doit pas apparaître dans la classe Rectangle, Carré ne doit pas restreindre les comportements de Rectangle, et c'est une classe Pinceau qui devrait gérer cette fonction (lien avec le Single Responsability Principle, et évitera duplication si une classe Rond doit elle aussi avoir cette méthode).

Interface Segregation Principle

> Préférer plusieurs interfaces spécifiques plutôt qu'une interface générale.

Objectif : Éviter aux classes de dépendre de méthodes dont elles n'ont pas besoin, réduisant les couplages inutiles. Ne pas devoir implémenter des méthodes inutiles à la classe.

Remarque

Il faut voir l'interface comme une liste de comportements que l'on souhaite ajouter à la classe. Il ne faut pas ajouter des comportements que l'objet ne doit pas avoir.

Dependency Inversion Principle

> Il faut dépendre des abstractions, pas des implémentations.

Objectif : Favoriser la modularité, la flexibilité et la réutilisabilité en réduisant les dépendances directes entre les modules.

Exemple

Dans une voiture, le conducteur n'a accès qu'aux pédales et pas directement au moteur. Les pédales jouent le rôle d'interface, de jonction, entre le conducteur et le moteur. Si le fonctionnement du moteur change, pas besoin de changer le conducteur, mais uniquement les liens entre les pédales et le moteur.