Génie logiciel – Synthèse

Définition

Le **génie logiciel** est une branche de l'informatique qui s'intéresse au développement structuré des logiciels, couvrant toutes les phases du cycle de vie :

- Analyse des besoins
- Spécifications
- Conception
- Programmation
- Tests
- Maintenance

Il englobe plusieurs disciplines comme la gestion de la qualité, l'architecture logicielle, l'analyse des besoins, la documentation, l'implémentation de normes (ISO, IEEE), et l'application des modèles CMM/CMMI.

Enjeux et dimensions

Les principaux enjeux du génie logiciel incluent :

- Répondre aux besoins des utilisateurs
- Respecter les délais
- Assurer la performance et la facilité de maintenance du logiciel.

Le génie logiciel couvre tout le cycle de vie d'un logiciel, depuis l'analyse initiale jusqu'à sa maintenance post-déploiement.

Architecture logicielle

L'architecture logicielle vise à structurer l'application pour :

- Réduire les coûts
- Améliorer la qualité

Elle repose sur deux principes fondamentaux :

1. **Séparation des responsabilités** : Diviser le logiciel en sous-parties distinctes et bien définies.
2. **Responsabilité unique** : Chaque sous-partie doit avoir une seule raison de changer.

Compétences en génie logiciel

Les compétences clés incluent :

- Conception et développement de logiciels selon des principes d'ingénierie.
- Analyse des problèmes pour proposer des solutions économiquement viables.
- Définition d'objectifs mesurables pour la sécurité, la fiabilité et la productivité.
- Mise en œuvre de solutions structurées.
- Vérification de la conformité des logiciels aux objectifs.
- Gestion de projets et coordination des équipes.
- Évaluation de la maturité des processus de développement.

Bonnes pratiques

Voici quelques bonnes pratiques :

- **Utilisation des diagrammes UML** :
    - Diagrammes de classes
    - Diagrammes de cas d'utilisation
- **Méthodes de développement efficaces** :
    - Concevoir l'architecture avant de coder.
    - Séparer le code en fichiers ayant une unité sémantique.
    - Anticiper l'évolution future du code.

Différence entre MVP et MVVM

MVP (Model View Presenter), évolution de MVC :

MVP est une évolution du modèle MVC (Model-View-Controller) et est principalement utilisé dans les applications de type desktop ou mobile.

Composants :

- Model : Gère la logique métier et l'accès aux données.
- View : Responsable de l'affichage des données et de l'interface utilisateur. Elle est passive et dépend du Presenter.
- Presenter : Gère la logique de présentation, interagit avec le modèle, et met à jour la vue. La communication est bidirectionnelle.

Fonctionnement :

- La Vue déclenche des événements (actions utilisateur).
- Le Presenter reçoit ces événements, récupère/modifie les données via le Model.
- Le Presenter met à jour la Vue en conséquence.

MVVM (Model View ViewModel) :

MVVM est largement utilisé dans les applications modernes avec des frameworks de développement d'interface utilisateur comme Angular, Vue.js, React, ou WPF (Windows Presentation Foundation). Le ViewModel sert de liaison entre la vue et le modèle, permettant le data binding.

Composants :

- Model : Contient la logique métier et la gestion des données.
- View : Interface utilisateur qui affiche les données.
- ViewModel : Gère la logique d'affichage et expose des données à la vue, souvent via le data binding (facilite la mise à jour automatique de l'UI).

Fonctionnement :

- La Vue interagit avec le ViewModel, généralement via un système de data binding (liaison automatique des données).
- Le ViewModel interagit avec le Model pour récupérer ou mettre à jour les données.
- Les changements de données sont automatiquement répercutés sur la Vue grâce au data binding.

Comment choisir ?

- MVP : Si l'application nécessite un contrôle précis de l'interface utilisateur (ex. applications Android natives sans data binding).
- MVVM : Si l'application repose sur des frameworks modernes et nécessite un data binding pour simplifier l'interface utilisateur.