Génie Logiciel

I. Définition

Un **Logiciel** est un ensemble de programmes, d'instructions et de données qui permettent à un ordinateur d'exécuter des tâches spécifiques (Word, Chrome, Excel...).

Un **Génie Logiciel** est une discipline de l'informatique qui consiste à concevoir, développer, tester et maintenir des logiciels de manière structurée et méthodique. Son objectif est de créer des logiciels de haute qualité, fiables, et faciles à maintenir.

II. Les étapes de développement logiciel

1. **Étape 1 : Définition des besoins et des exigences**
   Objectif : Comprendre précisément ce dont le client et les utilisateurs ont besoin.
2. **Étape 2 : Analyser le système**
   Objectif : Affiner et approfondir les besoins définis à l'étape précédente.
3. **Étape 3 : Conception du système**
   Objectif : Définir les choix techniques nécessaires pour la réalisation du logiciel.
4. **Étape 4 : Programmer le logiciel**
5. **Étape 5 : Tester le logiciel**
   Objectif : Vérifier que le logiciel fonctionne correctement et qu'il répond bien aux besoins définis au début du projet.
6. **Étape 6 : Déployer**
   Objectif : Installer le logiciel dans l'environnement réel des utilisateurs (serveurs, ordinateurs…).
7. **Étape 7 : Maintenir le système**
   Objectif : Corriger les erreurs, résoudre les problèmes et mettre à jour le logiciel.

III. Cycle de vie d'un logiciel (Cascade, V, Spirale, Évolutif)

1. Le modèle en cascade

Les étapes de développement se font de manière séquentielle, une par une, sans retour en arrière. Chaque étape doit être terminée avant de passer à la suivante.

Étapes :

- **Étude de faisabilité** : Vérifie si le projet est réalisable.
- **Analyse des besoins** : Détermine les fonctionnalités du logiciel avec le client.
- **Conception générale** : Planifie le fonctionnement global du logiciel.
- **Conception détaillée** : Précise les aspects techniques du logiciel.
- **Réalisation** : Développe le logiciel.
- **Intégration** : Assemble et teste les différentes parties du logiciel.
- **Déploiement** : Installe le logiciel pour les utilisateurs puissent l'utiliser.

Avantages :

- **Facile à suivre** : Les étapes sont claires et bien organisées.
- **Étapes définies** : Chaque étape est distincte, ce qui facilite le suivi.
- **Bonne documentation** : Des notes précises sont prises à chaque étape.
- **Bonne gestion** : Idéal pour les projets avec des objectifs clairs dès le départ.

Inconvénients :

- **Peu flexible** : Impossible de revenir en arrière après chaque étape.
- **Risque d'erreurs élevé** : Les erreurs ne sont découvertes qu'à la fin.
- **Temps d'attente pour les clients** : Le produit final est vu seulement à la fin, ce qui peut générer des insatisfactions si le produit ne répond pas aux attentes.
- **Adaptabilité limitée aux changements** : Moins adapté aux projets avec des exigences changeantes.

---

2. Le modèle en V

Le modèle en V est un processus séquentiel comme le modèle en cascade, mais avec un focus sur la vérification et la validation à chaque étape. Chaque étape de développement est associée à une phase de test, formant une structure en "V".

Étapes :

- **Analyse des besoins** : Identifier et documenter les attentes du client et des utilisateurs.
- **Conception système** : Créer une vue d'ensemble du logiciel, en définissant ses fonctionnalités principales et sa structure.
- **Conception des composants** : Diviser le système en composants et concevoir (conception) chaque module en détail.
- **Réalisation** : Coder chaque composant selon les plans de conception.
- **Tests unitaires** : Tester chaque composant indépendamment pour vérifier son bon fonctionnement.
- **Tests d'intégration** : Assembler les composants et vérifier qu'ils fonctionnent ensemble.
- **Tests système** : Tester le système complet pour s'assurer de son bon fonctionnement global.

Avantages :

- **Clarté et organisation** : Les étapes sont définies et suivent un ordre logique, facilitant la gestion.
- **Tests réguliers** : Des tests à chaque étape afin de repérer rapidement les erreurs.
- **Qualité assurée** : Les tests garantissent le bon fonctionnement et la qualité du logiciel.

Inconvénients :

- **Rigidité** : Le modèle suit une séquence stricte, rendant les changements difficiles.
- **Coût élevé des tests** : L'accent sur les tests augmente les coûts et le temps de développement.
- **Adaptabilité limitée** : Les changements après la phase de conception peuvent entraîner des retards.

---

3. Le modèle en spirale

Le **modèle en spirale** est une méthode de développement logiciel qui combine la logique séquentielle du modèle en cascade avec des cycles itératifs. À chaque cycle, le produit est amélioré progressivement. Il comprend quatre phases principales, répétées jusqu'à la fin du projet.

Les phases du modèle en spirale sont :

1. **Identification des objectifs et des contraintes** : Définir les objectifs et les limites pour chaque cycle.
2. **Analyse et évaluation des risques** : Identifier les risques et trouver des solutions pour les réduire.
3. **Développement et validation** : Développement initial des fonctionnalités et réalisation des tests et validations.
4. **Planification du prochain cycle** : Évaluer les progrès et planifier le cycle suivant.

Nombre de cycles :

- Le nombre de cycles dépend de la durée du projet, des exigences, des risques et des retours utilisateurs.
- Il n'y a pas de limite fixe, les cycles continuent jusqu'à ce que les risques soient réduits et que le produit soit prêt.

Avantages :

- **Gestion des risques** : Identifier et gérer les risques dès le début, en évitant des erreurs coûteuses en les testant tôt.
- **Flexibilité** : S'adapte facilement aux nouveaux besoins ou retours pendant le développement.
- **Prototypage** : Créer une version simple pour tester les idées et vérifier leur faisabilité avant de continuer.

Inconvénients :

- **Coût élevé** : L'approche itérative et la gestion des risques rendent le modèle plus coûteux que des modèles plus simples.
- **Dépendance à l'expertise** : Le succès repose sur une bonne gestion des risques, et des erreurs dans ce domaine peuvent affecter le projet.
- **Difficile à appliquer aux petits projets** : Pour des projets simples, ce modèle peut sembler trop complexe et coûteux par rapport à des méthodes comme le modèle en V ou en cascade.

---

4. Le modèle évolutif :

Le modèle évolutif est une méthode de développement où le produit est construit progressivement, version par version. Chaque version apporte de nouvelles fonctionnalités ou améliore les existantes, en tenant compte des retours des utilisateurs.

Les phases :

1. **Définir les exigences de base** : Identifier les fonctionnalités minimales pour commencer.
2. **Concevoir l'architecture** : Créer une base solide pour construire les futures fonctionnalités.
3. **Développer un incrément** : Ajouter une fonctionnalité ou amélioration au produit.
4. **Intégrer et tester** : S'assurer que l'incrément fonctionne bien avec les éléments déjà développés.
5. **Collecter des retours** : Recueillir des avis pour affiner le produit.
6. **Planifier l'incrément suivant** : Préparer le prochain cycle d'améliorations.
7. **Répéter les cycles** : Jusqu'à ce que le produit final soit atteint.

Avantages :

- **Flexible** : Permet de changer les exigences en cours de projet.
- **Retour continu** : Les utilisateurs peuvent tester et donner leur avis à chaque étape.
- **Réduction des risques** : Les erreurs sont corrigées plus tôt grâce aux tests fréquents.

Inconvénients :

- **Difficile à estimer** : Le coût et la durée sont difficiles à prévoir.
- **Risque de dérive** : Le projet peut s'allonger avec des ajouts imprévus.
- **Gestion complexe** : Nécessite une organisation rigoureuse pour bien gérer les itérations.

En résumé :

- Le modèle en spirale est itératif et axé sur la gestion des risques, idéal pour les projets complexes.
- Le modèle évolutif est orienté vers un développement progressif basé sur les retours utilisateurs, idéal pour les projets qui doivent s'adapter aux besoins changeants.

---

5. Méthodes agiles :

Pourquoi les méthodes agiles ont été créées :

- **Centrées sur le développement** : Elles se concentrent sur la programmation, ce qui plaît aux ingénieurs.
- **Basées sur des itérations** : Elles avancent par petites étapes répétées, permettant des améliorations continues.
- **Livraison rapide de logiciels** : Les clients peuvent tester le produit, donner leur avis et demander des ajustements facilement.

Les méthodes agiles incluent des approches comme :

- **Extreme Programming (XP)** : Est une méthode agile qui privilégie la collaboration avec le client, la livraison rapide et l'amélioration continue du code, avec des pratiques comme les tests automatisés et la programmation en binôme.
- **Feature-Driven Development (FDD)** : Se concentre sur le développement de fonctionnalités spécifiques, ajoutées progressivement. Les projets sont divisés en itérations courtes, avec une forte implication du client, pour répondre aux besoins spécifiques de l'utilisateur.

Ces méthodes visent à rendre le développement plus flexible et réactif.

---

1. Extreme Programming (XP) :

- Méthode agile axée sur la collaboration avec le client, la livraison rapide de fonctionnalités et l'amélioration continue du code.
- Utilise des itérations courtes et une grande flexibilité.
- Pratiques comme les tests automatisés et la programmation en binôme pour assurer la qualité du code.

2. Feature Driven Development (FDD) :

- Méthode de gestion de projet axée sur le développement par fonctionnalités et la gestion des risques.
- Les projets sont divisés en itérations courtes autour de fonctionnalités testables.
- Forte implication du client tout au long du processus.
- L'objectif est de développer des fonctionnalités répondant aux besoins spécifiques des utilisateurs.

FDD est mieux adapté aux grands projets.

- Il offre une organisation structurée, une planification détaillée et une gestion des fonctionnalités complexes.

XP convient aux petits projets.

- En raison de sa flexibilité, de sa réactivité, et de sa forte collaboration avec le client pour garantir la qualité du code.

IV. La conception

La **conception** en génie logiciel est l'étape où l'on planifie et organise le système à créer. Elle se situe entre l'analyse des besoins et la phase de développement (ou codage).

Objectif :

- Créer l'architecture et la structure détaillée du logiciel pour répondre aux exigences de l'analyse.

- Elle décrit le fonctionnement du système, y compris l'architecture, les modules, les interfaces et les détails techniques.

---

1. Conception préliminaire (ou architecturale)

- **Définir la structure générale du système, les composants principaux et leurs interactions.**
- **Choisir les technologies et outils à utiliser (langages de programmation, base de données...).**

Objectif : Définir les grandes lignes de l'architecture du système, y compris le type d'architecture à adopter.

L'**architecture du système** désigne la structure du logiciel, définissant comment ses différentes parties s'organisent et interagissent. Elle inclut la division du système en couches ou modules avec des tâches spécifiques.

Types d'architectures de systèmes :

- **Architecture 2-tiers (client-serveur)** : Divise le système en deux parties principales :
  - **Le client** : L'interface avec laquelle l'utilisateur interagit.
  - **Le serveur** : Gère les données et le traitement des informations.
- **Architecture 3-tiers** : Cette architecture divise le système en trois couches :
  - **Couche de présentation** : L'interface utilisateur, avec laquelle l'utilisateur interagit (site web/application mobile).
  - **Couche métier** : Elle gère la logique de traitement des données et les règles d'affaires (calcul des impôts, gestion des comptes utilisateurs).
  - **Couche de données** : Responsable de la gestion de la base de données, du stockage et de la récupération des informations.

2. Conception détaillée

- Consiste à détailler la conception préliminaire en définissant les modules, interfaces, structures de données et algorithmes.
- On utilise des diagrammes (comme les diagrammes de classe) et des spécifications pour définir les fonctions et les comportements des composants du système.

V. La Modélisation

La modélisation est une étape ou une activité qui consiste à représenter de manière simplifiée un système ou un processus afin de mieux le comprendre ou d'analyser son fonctionnement. Cela se fait à l'aide de diagrammes, schémas ou modèles conceptuels pour visualiser les besoins, solutions ou relations d'un projet.

Objectifs : Modéliser revient à :

- **Analyser les besoins** pour comprendre les fonctionnalités du logiciel.
- **Représenter les solutions fonctionnelles et techniques.**

- **Créer des diagrammes** pour clarifier et partager les idées avec les parties prenantes.

Les diagrammes UML incluent :

- **Diagramme de cas d'utilisation** : Montre les interactions entre les acteurs et le système.
- **Diagramme de séquence** : Illustre l'échange de messages entre objets au fil du temps.
- **Diagramme de classes** : Décrit la structure du système, incluant les classes, attributs et relations.
- **Diagramme d'états** : Représente les états d'un objet et les transitions entre eux.

Différence clé : modélisation et conception

- **Modélisation** = Représentation simplifiée pour comprendre, analyser ou simuler. Elle se concentre sur la représentation visuelle ou abstraite des idées.
- **Conception** = Définition détaillée de la solution et de la structure finale à produire ou à réaliser.

Un diagramme : est une représentation graphique utilisée pour illustrer des informations, des processus, des structures ou des relations de manière visuelle.

Caractéristiques :

- **Simplicité** : Résume des informations complexes de manière facile à comprendre.
- **Clarté** : Utilise des formes, flèches, couleurs et symboles pour organiser visuellement les éléments.
- **Communication** : Facilite la compréhension et l'échange d'idées en rendant les concepts accessibles à tous.

➢ UML : (Unified Modeling Language)

L'**UML** est un langage de modélisation graphique visuel utilisé pour spécifier, concevoir, documenter et analyser des systèmes logiciels à l'aide de diagrammes graphiques.

Types de diagrammes UML :

1. **Diagrammes structurels** : Modélisent la structure du système (classe, objet).
2. **Diagrammes comportementaux** : Illustrent les comportements dynamiques du système (cas d'utilisation).
3. **Diagrammes d'interaction** : Détaillent les échanges entre objets ou composants (séquence).

Diagramme de classe :

Le **diagramme de classe** représente la structure statique d'un système en montrant les classes, leurs attributs, méthodes et les relations entre elles. Il sert à définir la structure d'un système orienté objet.

➢ L'orienté objet :

L'orienté objet est un paradigme de programmation qui organise le code autour d'objets, qui regroupent données (attributs) et comportements (méthodes). Cela permet de modéliser des systèmes de manière plus modulaire, flexible et réutilisable. Il se repose sur des concepts comme l'encapsulation, l'héritage, le polymorphisme et l'abstraction, qui permettent de modéliser des systèmes flexibles et réutilisables.

Les principaux composants d'un diagramme de classe UML sont :

- **Classe** : Représentée par un rectangle divisé en trois parties, elle contient le nom, les attributs (propriétés) et les méthodes (fonctions).
- **Attributs** : Les caractéristiques ou propriétés d'une classe/comportement qu'un objet peut faire.
- **Méthodes** : Les actions ou comportements qu'une classe peut effectuer.
- **Relations** : Les liens entre les classes, comme l'association (représentée par une ligne), l'héritage (une flèche avec un triangle) et la composition ou agrégation (indiquées par des losanges).
- **Visibilité** : Indiquée par des symboles (+ public, - privé, # protégé) avant les attributs et méthodes.
    - **+ (public)** : L'élément est accessible depuis n'importe quelle autre classe.
    - **- (privé)** : L'élément est accessible uniquement au sein de la classe elle-même.
    - **# (protégé)** : L'élément est accessible dans la classe et ses sous-classes (héritées).

Les différents types de relations dans un diagramme de classe :

- **Héritage (Généralisation)** : Permet à une classe d'hériter des attributs et méthodes d'une autre classe. Cela favorise la réutilisation du code et la création de hiérarchies. (Représenté par une flèche avec un triangle pointant vers la classe de base).
- **Composition** : Une classe contient une autre classe de manière forte ; si la classe tout est supprimée, les objets de la classe partie le sont aussi. Représentée par un losange plein.
- **Agrégation** : Une classe contient une autre classe, mais les objets peuvent exister indépendamment. Représentée par un losange vide.
- **Dépendance** : Une classe utilise ou dépend d'une autre classe, généralement pour une opération ponctuelle. Représentée par une ligne pointillée avec une flèche.