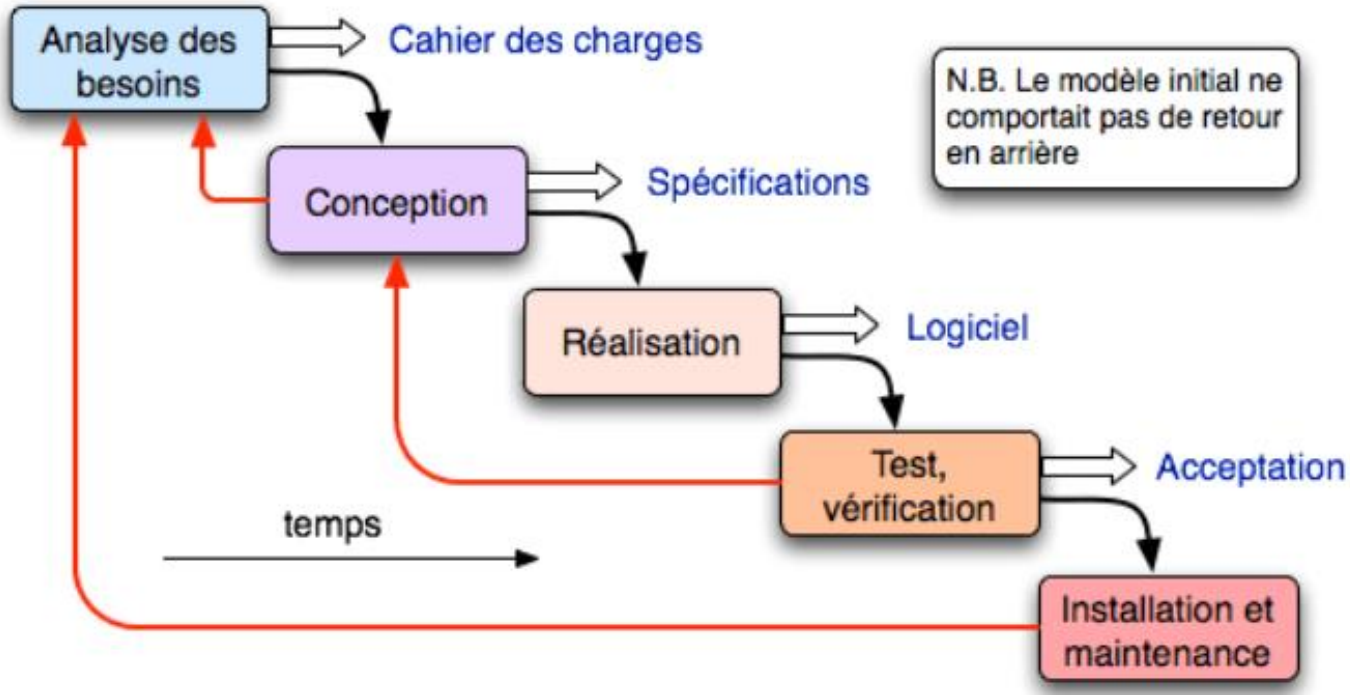
Génie Logiciel

- Dispensé par Dr. Msc. Ir. **MWAMBA KASONGO Dahouda**
  Docteur en génie logiciel et systèmes d'information
  Machine and Deep Learning Engineer

- Assisté Master Student, Ir. Jason MUSA

Heure : 08H00 – 12H00



Introduction au Génie Logiciel

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.1. Historique, définitions et objectifs

1.1.1 Historique

Le **génie logiciel** est une discipline relativement récente qui est apparue en réponse à ce qu'on appelle la "**crise du logiciel**" des années 1960 et 1970

- L'expression « Génie Logiciel » ou « Software Engineering » est introduit la première fois à une conférence en Allemagne en 1968 pour répondre à la crise logiciel.

La crise du logiciel fait référence à la période où les coûts, les délais et la qualité des projets logiciels étaient souvent Insatisfaisants.

❖ **Années 1950-1960 : Premiers programmes**

✓ Dans les premières décennies de l'informatique, les logiciels étaient souvent développés par les mêmes personnes qui construisaient le matériel (hardware).
✓ Le processus de développement était artisanal, et il n'y avait pas de méthodologie formalisée.
✓ Les logiciels étaient petits et spécifiques à des tâches précises.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.1. Historique, définitions et objectifs

1.1.1. Historique

❖ **1968: La crise du logiciel**

- ✓ Lors d'une conférence de l'OTAN à Garmisch, en Allemagne, en 1968, le terme "crise du logiciel" a été introduit pour décrire les problèmes croissants liés au développement de logiciels.
- ✓ Les logiciels devenaient de plus en plus complexes, avec des retards, des dépassements de budget, des bugs non résolus, et des projets annulés.
- ✓ Cette crise a souligné la nécessité de nouvelles méthodes pour améliorer la qualité et la gestion des projets logiciels.

❖ **Années 1970-1980 : Naissance du génie logiciel**

- ✓ C'est à partir de cette période que le génie logiciel a commencé à se structurer comme une discipline à part entière.
- ✓ De nouvelles méthodologies ont émergé, comme le modèle en cascade (waterfall), qui est un processus séquentiel de développement en phases (analyse, conception, implémentation, test, etc.).
- ✓ Des langages de programmation de plus haut niveau, comme Fortran, COBOL, et Pascal, ont permis une meilleure abstraction et réutilisation du code.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.1. Historique, définitions et objectifs

1.1.1. Historique

❖ **Années 1990 : Montée en puissance des méthodologies itératives**

✓ Avec l'augmentation de la taille et de la complexité des logiciels, des méthodologies plus flexibles, comme le modèle spiral et le modèle itératif, ont été introduites.

Le Rational Unified Process (RUP) et Extreme Programming (XP), deux méthodes itératives, ont vu le jour pendant cette période. Elles mettaient l'accent sur l'évolution continue des systèmes et des livraisons fréquentes.

❖ **Années 2000 à aujourd'hui** : **Approches agiles**

✓ Le Manifeste Agile (2001) a marqué un tournant dans la gestion des projets logiciels.
✓ Les méthodologies agiles, comme Scrum et Kanban, ont été adoptées pour répondre aux exigences des environnements en constante évolution.

Ces méthodes favorisent des cycles de développement courts, la collaboration avec le client et la flexibilité dans la gestion des changements.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.1. Historique, définitions et objectifs

1.1.2. Définisions

- L'idée de génie logiciel est d'appliquer les méthodes classique d'ingénierie au domaine du logiciel.

- **Ingénierie** (ou **génie**): Ensemble de fonction allant de la conception et des études à la responsabilité de la construction et au contrôle des équipement d'une installation techniques ou industrielle.

- Tel que défini par Pollice, 2005 et IEEE 1993, le génie logiciel est l'application d'une approche disciplinée quantifiable systématique, au développement, à l'exploitation et à la maintenance des logiciels, ainsi que l'étude de ces approches.

Le génie logiciel est un ensemble **des méthodes**, **des techniques**, et des **outils dédiés** à **la conception**, au **développement** et à **la maintenance** des systèmes informatiques.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.1. Historique, définitions et objectifs

1.1.3. Objectifs

Le génie logiciel (GL) se préoccupe des procédés de fabrication de logiciel de façon à s'assuré que les quatre critères suivant soit satisfait: **Coût (C)**, **Qualité (Q)**, **Fonctionnalité (F)**, et **Délai (D).**

Les concepts de **Coût (C)**, **Qualité (Q)**, **Fonctionnalité (F)**, et **Délai (D)** sont des facteurs clés dans la gestion de tout projet de développement logiciel. Ils constituent les principaux critères d'évaluation du succès d'un projet, explication de chacun:

- ✓ **Coût (C)**: les coûts reste dans les **limites prévu au départ**.
  - Gérer les ressources de manière optimale pour respecter le budget sans compromettre la qualité ou la fonctionnalité.

- ✓ **Qualité (Q)**: La qualité correspond au **contrat du service initial**.
  - Fournir un produit fiable, utilisable, sécurisé et performant qui respecte les normes de qualité définies par les parties prenantes.

- ✓ **Fonctionnalité (F)**: Le système développé doit répondre aux **besoins des utilisateurs**.
  - Assurer que toutes les fonctionnalités requises soient correctement implémentées, et qu'elles répondent aux attentes des utilisateurs finaux.

- ✓ **Délai (D)**: Les délais correspondent au respect des **dates de livraison** prévues.
  - Terminer le projet à temps, tout en maintenant un équilibre entre qualité, coût et fonctionnalités

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.2. Principes de génie logiciel

❖ Rigueur

- Les principales sources de défaillances d'un logiciel sont d'origine humaine.

✓ À tout moment, il faut se questionner sur la validité de son action.

- Des outils de vérification accompagnant le développement peuvent aider à réduire les erreurs.

Cette famille d'outils s'appelle CASE (Computer Aided Software Engineering).

❖ Abstraction

- Extraire des concepts généraux sur lesquels raisonner, puis instancier les solutions sur les cas particuliers.

❖ Décomposition en sous problèmes

Traiter chaque aspect séparément, chaque sous-problème plus simple que problème global.

❖ Modularité

- Partition du logiciel en modules interagissant, remplissant une fonction et ayant une interface cachant l'implantation aux autres modules

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.2. Principes de génie logiciel

❖ Construction incrémentale

- Construction pas à pas, intégration progressive.

❖ Généricité

- Proposer des solutions plus générales que le problème pour pouvoir les réutiliser et les adapter à d'autres cas.
- Un logiciel réutilisable a beaucoup plus de valeur qu'un composant dédié.

❖ Anticipation des évolutions

- Liée à la généricité et à la modularité, prévoir les ajouts/modifications possibles de fonctionnalités.

Documentations

- Essentielle pour le suivi de projet et la communication au sein de l'équipe de projet.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.3. Qualités attendues d'un logiciel

Les qualités attendues d'un logiciel sont des critères essentiels qui déterminent son succès et son utilité pour les utilisateurs. Ces qualités couvrent plusieurs aspects techniques et fonctionnels, garantissant que le logiciel est **fiable**, **performant**, et **adapté** à son environnement d'utilisation.

❖ Qualites Externe

- **La validité**: aptitude d'un produit logiciel à réaliser exactement les tâches définies par sa spécification.
- **La robustesse**: aptitude d'un logiciel à fonctionner même dans des conditions anormales.
- **L'extensibilité**: Facilité d'adaptation d'un logiciel aux changement de spécification.
- **Réutilisabilité**: aptitude d'un logiciel à être réutiliser en tout ou en partie pour des nouvelles applications.
- **La compatibilité**: aptitude d'un logiciel à pouvoir être combiner les un avec les autres.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.3. Qualités attendues d'un logiciel

D'autres facteurs de qualité du logiciel sont moins cruciales:

❖ Qualites Externe

- **L'efficacité**: bonne utilisation des ressources du matériel.
- **La portabilité**: facilité aux laquelle le produit être adapté à des différents environnements matériel ou logiciel.
- **Vérifiabilité**: facilité de préparation des procédures de recette et de certification (test).
- **L'intégrité**: aptitude des logiciels à protéger leurs différents composants contre accès et des modifications non autorisés.
- **Facilité d'utilisation**: facilité avec lesquelles les utilisateurs d'un logiciel peuvent apprendre comment l'utiliser, comment le faire fonctionner, comment préparer les données, mais aussi comment interpréter les résultats et les effets en cas d'erreur.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.3. Qualités attendues d'un logiciel

Des critères interne permettent d'atteindre ces facteurs externe de qualité

❖ Qualites Interne

- **La modularité**: c'est la décomposition du logiciel en composant facilement appréhendable et relativement indépendants.
- **La complétude**: c'est le degré d'implantation des spécification.

Un logiciel est complet si toutes ses spécifications externes sont opérationnelles.

- **La cohérence**: c'est la possibilité de faire des retours en arrières dans le cycle de développement.

En particulier de faire remonter une erreur détectée en maintenance au niveau de l'implantation, de la conception, ou de l'analyse.

- **La généralité**: plage d'application potentielle des composants logiciel.
- **L'auto documentation et lisibilité**: possibilité d'extraction de la documentation depuis les composants logiciel.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.4. Cycle de vie d'un logiciel

Le cycle de vie du logiciel modélise l'enchaînement des différentes activités du processus technique de développement du logiciel.

Une activité comprend: des tâches, des contraintes, des ressources, une façon d'être réalisée.

Les grandes activités sont:

- ✓ **Analyse des besoins**
- ✓ **spécification des besoins**
- ✓ **Conception architecturel et détaillé**
- ✓ **Programmation**
- ✓ **Gestion de configuration et d'intégration**
- ✓ **Validation et vérification**
- ✓ **Livraison et maintenance**

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.4. Cycle de vie d'un logiciel

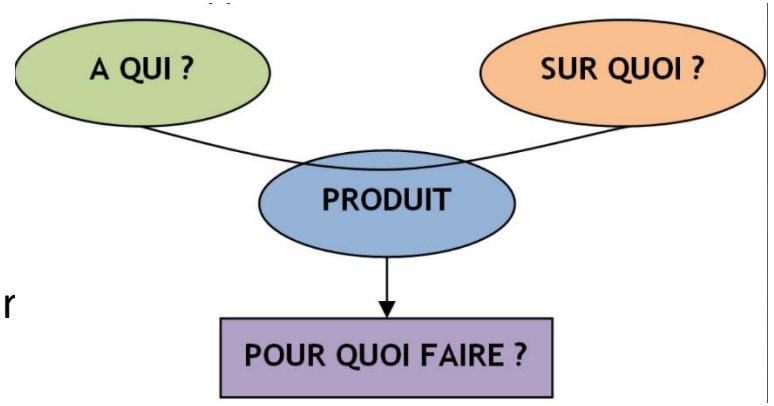
1.4.1. Analyse des besoins

- **Objectif** : Comprendre et définir précisément ce que le client ou l'utilisateur final attend du système.

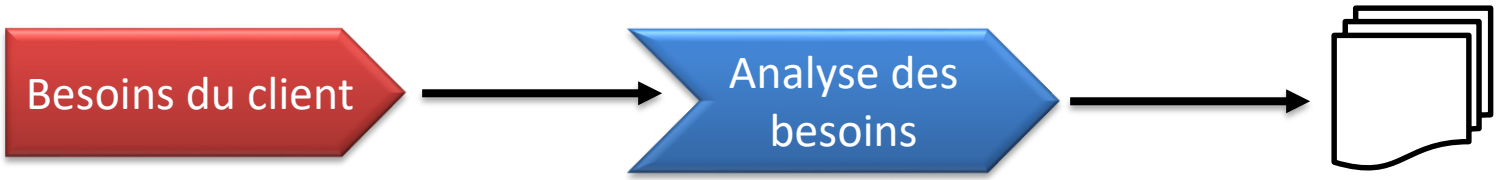
Cette étape implique la collecte, la documentation, et la priorisation des besoins fonctionnels et non-fonctionnels.

- **Actions** :

✓ Identifier les **exigences fonctionnelles** (ce que le logiciel doit faire).
✓ Identifier les **exigences non fonctionnelles** (performance, sécurité, fiabilité, etc.).
✓ Communiquer avec les **parties prenantes** (utilisateurs, clients, responsables).
✓ Créer des **cas d'utilisation** pour illustrer le fonctionnement attendu du système.



Output : Document de spécification des besoins, diagrammes de cas d'utilisation, cahier



- **Experts du domaine d'application**
- **Future Utilisateurs du système**

- **Entretien**
- **Questionnaire**
- **Observation**
- **Etude de situation similaire**

- **Cahier des Charges (CC)**
- **Manuel d'utilisation préliminaire**

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.4. Cycle de vie d'un logiciel

1.4.2. spécification des besoins

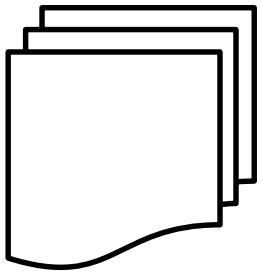
- **Objectif** : Traduire les besoins recueillis lors de l'analyse en spécifications détaillées et globales du système.

C'est la transition entre ce que le **client** souhaite et ce que **l'équipe de développement** peut concevoir.

- **Actions** :

✓ Formaliser les **exigences** dans un document plus technique.
✓ Définir les **interfaces utilisateur (IHM)** et les interactions principales.
✓ Élaborer des spécifications de haut niveau pour les **modules** du système.
✓ Valider les exigences auprès des **utilisateurs** et des **parties prenantes** pour s'assurer qu'elles sont complètes et cohérentes.

Output : Document de spécification technique globale, maquettes d'interfaces utilisateur, diagrammes UML de haut niveau.

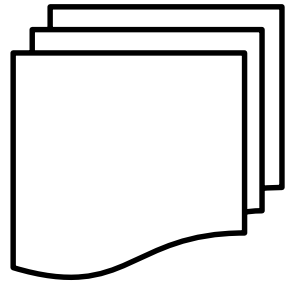


- **Cahier des Charges (CC)**
- **Manuel d'utilisation préliminaire**

spécification des besoins

1- Spécification en langage naturel
2- Spécification semi-formelle
3- Spécification formelle

- **Modèle E/A**
- **Diagrammes d'etats transition**



- **Cahier des Charges fonctionnel**
- **Technique et faisabilité informatique**

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

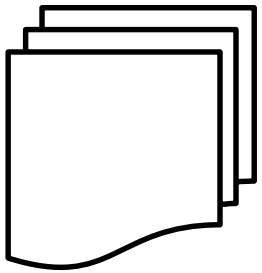
1.4. Cycle de vie d'un logiciel

1.4.3. Conception architecturel et détaillée

- **Objectif** : Concevoir la structure technique du système, en définissant comment les différentes parties du système interagiront entre elles (architecture) et les détails techniques de chaque composant (conception détaillée).

✓ **La conception architecturale** (ou conception globale) a pour but de décomposer le logiciel en composants plus simples, déf inis par leurs interfaces et leurs fonctions (les services qu'ils rendent).

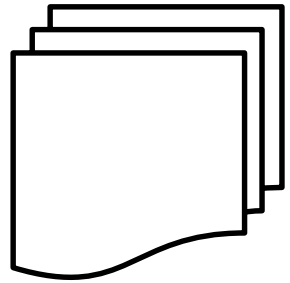
✓ **La conception détaillée** a pour but de détailler les spécifications de chaque composant.
Créer des diagrammes UML comme les diagrammes de classes, de séquence, ou d'état pour illustrer le fonctionnement interne.



- **Cahier des Charges (CC)**
- **Manuel d'utilisation préliminaire**

Conception
La conception architecturale
La conception détaillée

- **Conception fonctionnelle et conception orienté object**
- **Pour chaque composant, ecrire les algorithmes**



- **Dossier de conception (Architecture et Algorithme)**

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.4. Cycle de vie d'un logiciel

1.4.4. Programmation

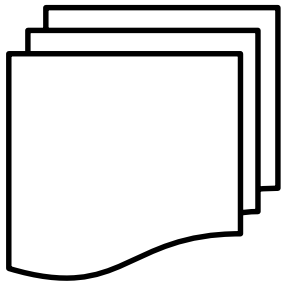
- **Objectif** : Développer le code correspondant aux spécifications techniques et à l'architecture définie.

Cette étape implique la **réalisation** concrète du système à travers l'écriture du code.

- **Actions** :

✓ Implémenter les fonctionnalités conformément à la **conception détaillée**.
✓ Suivre des **standards de codage** et des pratiques comme **l'intégration continue** pour tester et intégrer fréquemment du nouveau code.
✓ Effectuer des **tests unitaires** et des **tests d'intégration** pendant le développement.
✓ Utiliser des outils de **gestion de version** (Git) pour suivre l'évolution du code.

- **Output** : Code source, composants logiciels fonctionnels, documentation technique du code.



- Dossier de conception (Architecture et Algorithme)

- Choix de l'environnement
- Choix du langage de programmation

- Code source
- Manuel final d'utilisation
- Documentations

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.4. Cycle de vie d'un logiciel

1.4.5. Gestion de configuration et d'intégration

- **Objectif** : Gérer les différentes versions du système, s'assurer que tous les composants sont intégrés correctement, et coordonner les évolutions du code.

➢ **Gestion de configuration** :

Elle a pour but de maîtriser l'évolution et la mise à jour des composants tout au long du processus de développement.

✓ Suivre les modifications du code et des configurations.
✓ Maintenir un historique des versions pour identifier les changements apportés.

➢ **Intégration** :

Elle a pour but de réaliser un ou plusieurs systèmes exécutables à partir des composants (Combiner les composants).

✓ Rassembler les différents composants développés dans une seule version cohérente.
✓ Automatiser l'intégration grâce à des outils d'intégration continue.

- **Output** : Système complet intégré, journal de version, documentation de configuration

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.4. Cycle de vie d'un logiciel

1.4.6. Validation et vérification

- **Objectif** : S'assurer que le logiciel répond aux besoins initiaux (validation) et que chaque partie du logiciel fonctionne correctement selon les spécifications techniques (vérification).

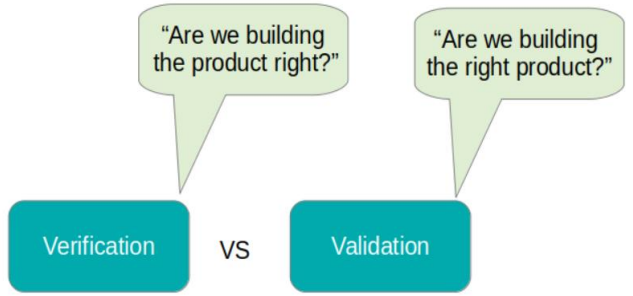
- **La Validation** a pour but de répondre à la délicate question : a-t-on décrit le bon système, celui qui répond à l'attente des utilisateurs.

✓ Valider que le logiciel répond aux attentes des utilisateurs en fonction des exigences.
✓ Faire des tests utilisateurs ou des tests d'acceptation pour vérifier que le produit correspond bien aux attentes.

- **Vérification** répond à la question : le développement est-il correct par rapport à la spécification globale ?

Ce qui consiste à s'assurer que les description successives et le logiciel lui-même satisfont la spécification.
✓ Tester chaque composant (tests unitaires) et l'ensemble du système (tests d'intégration).
✓ Utiliser des tests automatiques et manuels pour valider les performances, la sécurité, et la fiabilité.

- **Output** : Rapports de tests, corrections des bugs identifiés, documentation de validation.



CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.4. Cycle de vie d'un logiciel

1.4.7. Livraison et maintenance

- **Objectif** : Livrer le logiciel au client/utilisateur final et garantir sa maintenance sur le long terme pour corriger les bugs, ajouter des fonctionnalités, ou améliorer les performances.

Il s'agit d'apporter des modifications à un logiciel existant. C'est la phase la plus coûteuse (70% du coût total).

- **Livraison** :

✓ Déployer le logiciel dans l'environnement de production.
✓ Former les utilisateurs et fournir la documentation d'utilisation.

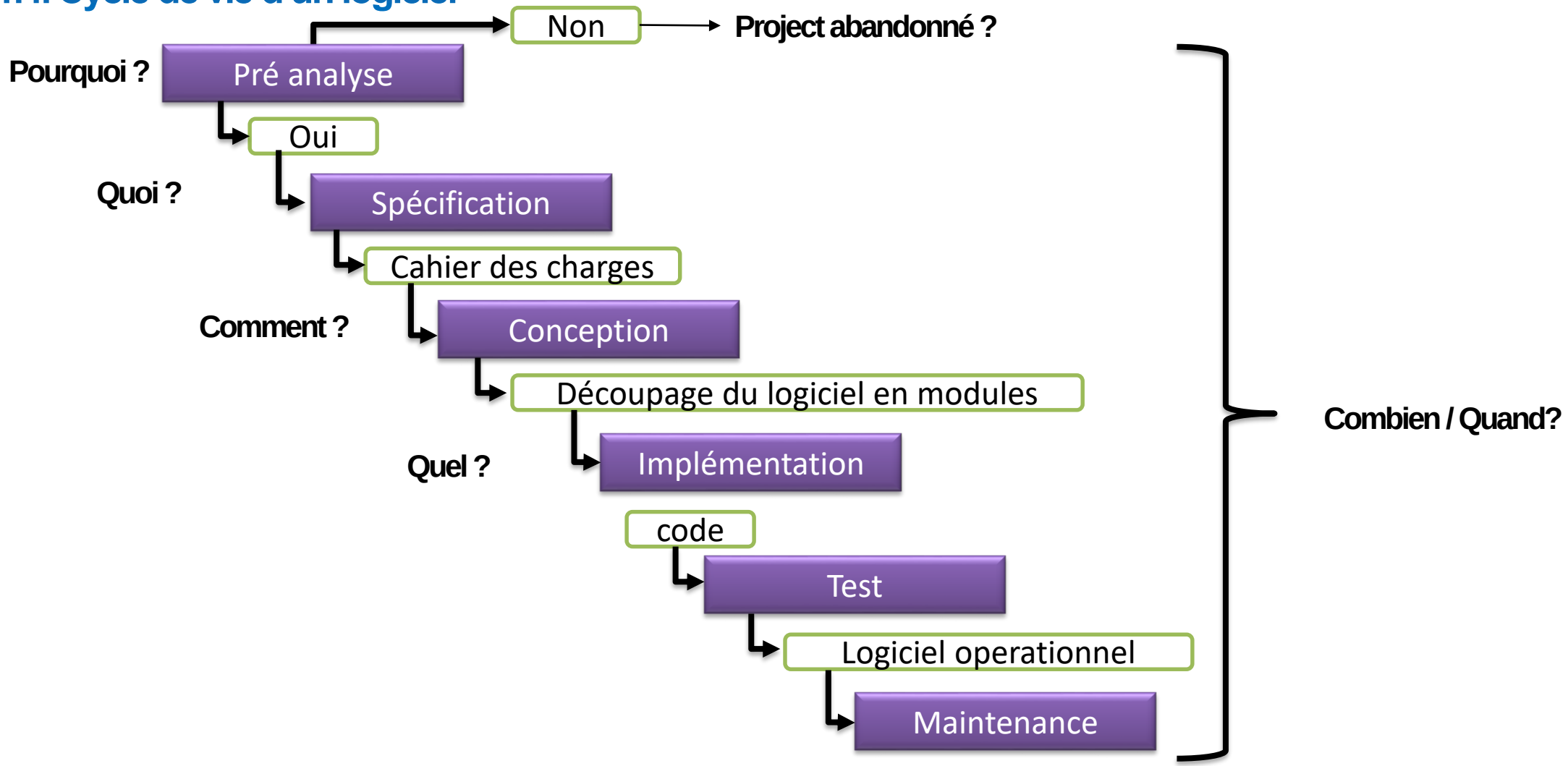
- **Maintenance** :

✓ Corriger les bugs identifiés après la mise en production.
✓ Effectuer des mises à jour pour ajouter des nouvelles fonctionnalités ou optimiser le système.
✓ Assurer la maintenance corrective, évolutive, et préventive.

- **Output** : Version finale du logiciel livrée, documentation utilisateur, plan de maintenance.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.4. Cycle de vie d'un logiciel



CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.4. Cycle de vie d'un logiciel

Documents:

Manuel utilisateur ----------> final Implémentation
Dossier de conception architecturale -----> Conception
Code source ----------> Implémentation
Cahier des charges ----------> Analyse des besoins
Manuel utilisateur préliminaire ----------> Analyse des besoins
Dossier de conception détaillée ----------> Conception
Rapport des tests ----------> Tests
Documentation ----------> Implémentation
Cahier des charges fonctionnel ----------> Spécification

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.5. Le cycle de vie du développement logiciel [Software Development Life Cycle] (SDLC)

L'objectif

- Le cycle de vie du développement logiciel (SDLC) est le processus rentable et efficace en termes de temps que les équipes de développement utilisent pour concevoir et créer des logiciels de haute qualité.

- L'objectif du SDLC est de minimiser les risques du projet grâce à une planification prospective, afin que le logiciel réponde aux attentes du client pendant la production et au-delà.

- Cette méthodologie décrit une série d'étapes qui divisent le processus de développement logiciel en tâches que vous pouvez assigner, compléter et mesurer.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.5. Le cycle de vie du développement logiciel [Software Development Life Cycle] (SDLC)

Pourquoi le SDLC est-il important ?

Le développement de logiciels peut être difficile à gérer en raison de l'évolution des exigences, des mises à jour technologiques et de la collaboration interfonctionnelle.

La méthodologie du cycle de vie du développement logiciel (SDLC) fournit un cadre de gestion systématique avec des livrables spécifiques à chaque étape du processus de développement logiciel.

Ainsi, toutes les parties prenantes s'accordent dès le départ sur les objectifs et les exigences du développement logiciel et disposent également d'un plan pour atteindre ces objectif.

Voici quelques avantages du SDLC :

- ✓ Visibilité accrue du processus de développement pour toutes les parties prenantes impliquées
- ✓ Estimation, planification et ordonnancement efficaces
- ✓ Amélioration de la gestion des risques et de l'estimation des coûts
- ✓ Livraison systématique des logiciels et meilleure satisfaction des client

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.5. Le cycle de vie du développement logiciel [Software Development Life Cycle] (SDLC)

Pourquoi le SDLC est-il important ?

L'objectif du modèle de cycle de vie SDLC est de fournir un logiciel de haute qualité et maintenable qui répond aux exigences des utilisateurs.

❑ Le SDLC dans les modèles d'ingénierie logicielle décrit le plan de chaque étape afin que chaque étape du modèle de développement logiciel puisse accomplir sa tâche efficacement pour fournir le logiciel à **faible coût** dans un **délai donné** qui répond aux **exigences des utilisateurs**.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.5. Le cycle de vie du développement logiciel [Software Development Life Cycle] (SDLC)

Comment fonctionne le SDLC ?

Le cycle de vie du développement logiciel (SDLC) décrit plusieurs tâches nécessaires pour créer une application logicielle.

Le cycle de vie du développement logiciel (SDLC) est un processus structuré utilisé pour concevoir, développer et tester des logiciels de bonne qualité.

Voici quelques phases courantes du SDLC :

- ✓ Planification
- ✓ Conception
- ✓ Implémentation
- ✓ Test
- ✓ Déploiement
- ✓ Maintenance

Nous avons répertorié ici les cinq modèles SDLC les plus populaires :

- ✓ Modèle en cascade [Waterfall Model]
- ✓ Modèle agile
- ✓ Modèle incremental ou itératif
- ✓ Modèle en spirale
- ✓ Modèle en V

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

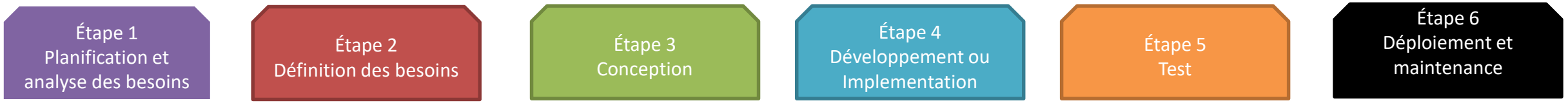
1.5. Le cycle de vie du développement logiciel [Software Development Life Cycle] (SDLC)

Étapes du cycle de vie du développement logiciel ?

- Le cycle de vie du développement logiciel spécifie les tâches à effectuer à différentes étapes par un ingénieur logiciel ou un développeur.
- Il garantit que le produit final est en mesure de répondre aux attentes du **client** et s'inscrit dans le budget global.

- Il est donc essentiel pour un **développeur de logiciels** d'avoir une connaissance préalable de ce processus de développement logiciel.

- Le cycle de vie du développement logiciel est un ensemble de ces six étapes, et les étapes du cycle de vie du développement logiciel sont les suivantes



CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.5. Le cycle de vie du développement logiciel [Software Development Life Cycle] (SDLC)

Exercice 1 : Analyse de besoins

Product Owner: Développement d'un logiciel de recensement de la population en RDC.

Trouver les cas d'utilisation possible pour le logiciel.
- Exigences fonctionnelles
- Exigences Non fonctionnelles

Exercice 2: Analyse de besoins

Product Owner: Développement d'une application de transport (Uber)

Trouver les cas d'utilisation possible pour le logiciel.
- Exigences fonctionnelles
- Exigences Non fonctionnelles

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

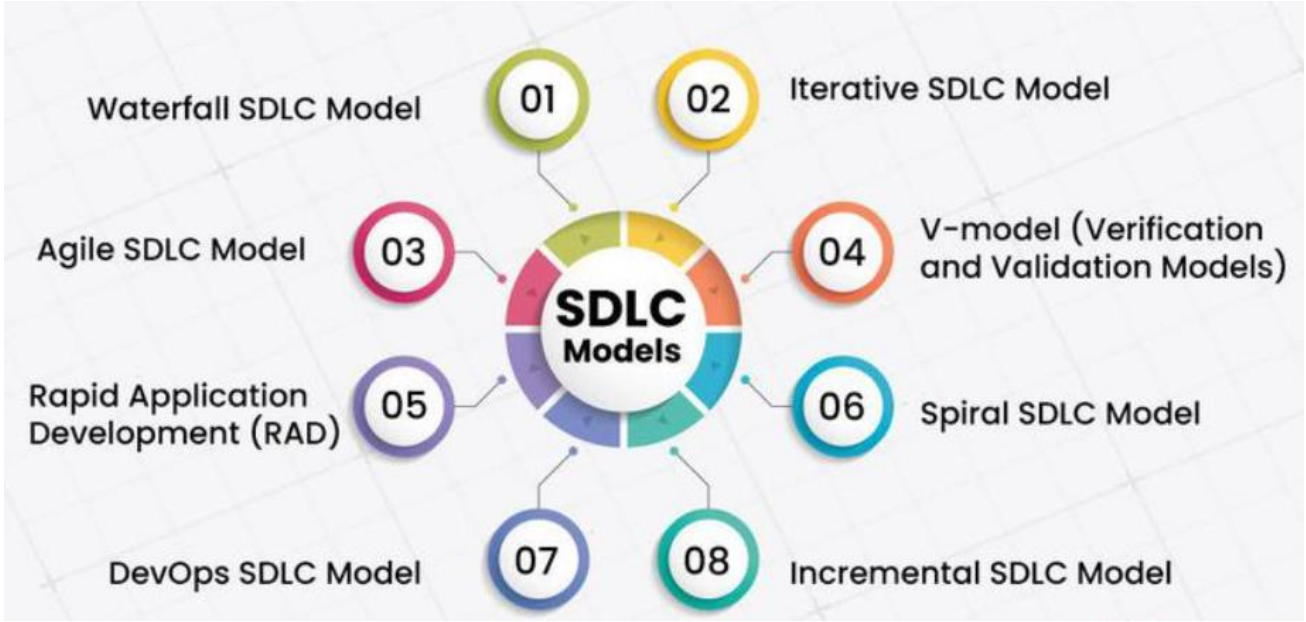
1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

Modèles de cycle de vie du développement logiciel

❑ À ce jour, nous avons plus de 50 modèles SDLC reconnus en usage.
Mais aucun d'entre eux n'est parfait, et chacun apporte ses avantages et ses inconvénients pour un projet de développement logiciel spécifique ou une équipe.

Nous avons répertorié ici les quelques modèles SDLC les plus populaires :

- ✓ Modèle en cascade [Waterfall Model]
- ✓ Modèle Itératif
- ✓ Modèle Agile
- ✓ Modèle en V
- ✓ Modèle RAD
- ✓ Modèle Spirale
- ✓ Modèle DevOps
- ✓ Modèle Incrémental



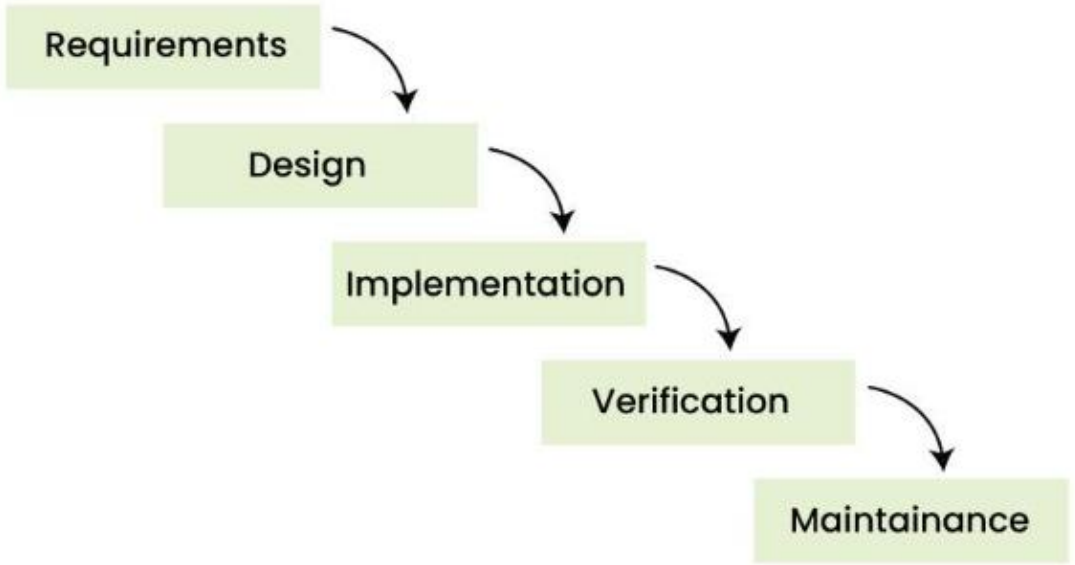
CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.1. Modèles en Cascade [Waterfall Model]

Le modèle en cascade est une version célèbre et efficace du SDLC pour l'ingénierie logicielle.
Le modèle en cascade est un modèle linéaire et séquentiel, ce qui signifie qu'une phase de développement ne peut pas commencer tant que la phase précédente n'est pas terminée. Nous ne pouvons pas superposer les phases dans le modèle en cascade.

Le modèle en cascade fonctionne également de la même manière : une fois qu'une phase de développement est terminée, nous passons à la phase suivante mais ne pouvons pas revenir à la phase précédente.
Dans le modèle en cascade, la sortie d'une phase sert d'entrée pour l'autre phase.



CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.1. Modèles en Cascade [Waterfall Model]

Phase d'exigence : La phase d'exigence est la première phase du modèle en cascade. Dans cette phase, les exigences du système sont collectées et documentées.

Phase de conception : La phase de conception est basée sur la façon dont le logiciel sera construit. L'objectif principal de la phase de conception est de préparer le plan directeur du système logiciel afin qu'aucun problème ne soit rencontré dans les phases à venir et que des solutions à toutes les exigences de la phase d'exigence soient trouvées.

Phase d'implementation : Dans cette phase, le matériel, les logiciels et les programmes d'application sont installés et la conception de la ba se de données est implémentée. C'est la phase la plus longue du modèle en cascade.

Phase de vérification : Dans cette phase, le logiciel est vérifié et il est évalué que nous avons créé le bon produit. Dans cette phase, différent s types de tests sont effectués et chaque domaine du logiciel est vérifié.
L'un des avantages de la vérification est qu'elle réduit le risque de défaillance du logiciel.

Phase de maintenance : il s'agit de la dernière phase de la cascade. Lorsque le système est prêt et que les utilisateurs commencent à l'utiliser, les problèmes qui surviennent doivent être résolus de temps en temps.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

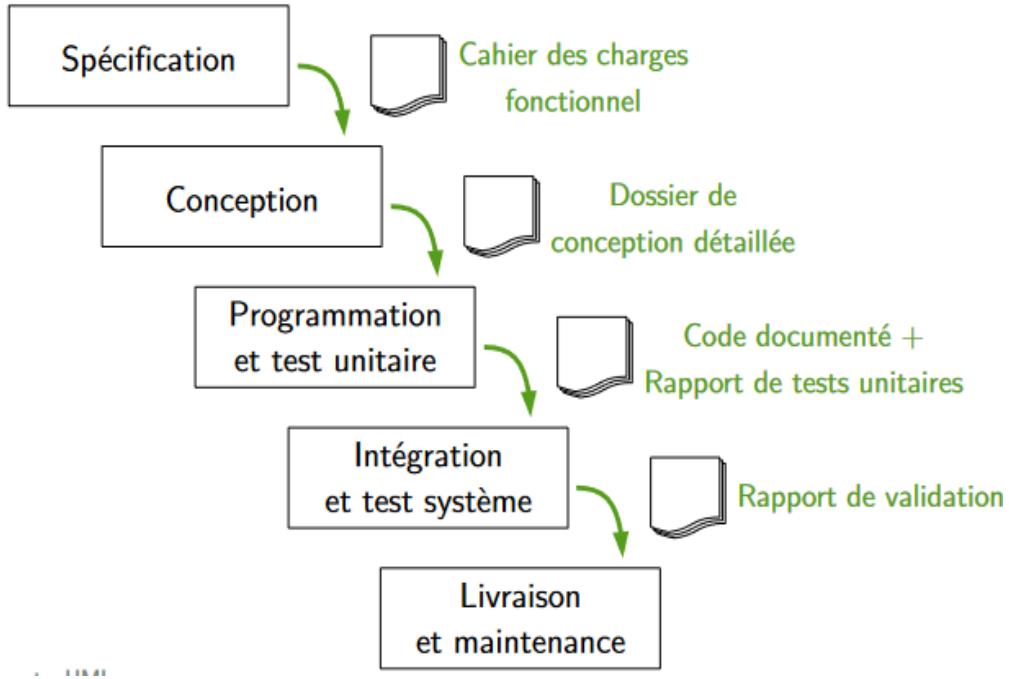
1.6.1. Modèles en Cascade [Waterfall Model]

Avantages

- ✓ Le planning est établi à l'avance et le chef du projet sait précisément ce qui va lui être livré et quand il pourra en prendre livraison.

Inconvénients

- ✓ Modèle trop séquentiel c.à.d. dure trop longtemps
- ✓ Validation trop tardive (remise en question coûteuse des phases précédentes)
- ✓ Sensibilité à l'arrivée de nouvelles exigences (refaire toutes les étapes)
- ✓ Bien adapté lorsque les besoins sont clairement identifiés et stables



CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.2. Modèle Itératif

Dans le modèle itératif, nous commençons à développer le logiciel avec certaines exigences et lorsqu'il est développé, il est révisé. S'il y a des exigences de modifications, nous développons une nouvelle version du logiciel en fonction de ces exigences. Ce processus se répète plusieurs fois jusqu'à ce que nous obtenions notre produit final.

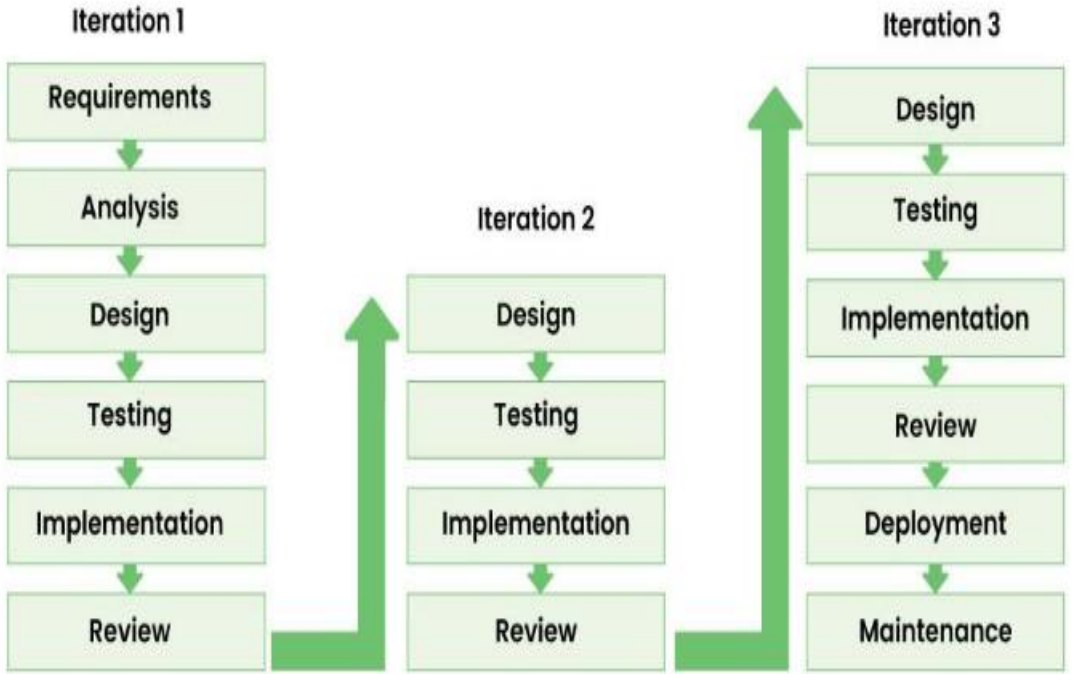
Début du cycle SDLC

- Commencez par quelques exigences et analyses
- Développez la première version du logiciel
- En cas de modifications, une nouvelle version est créée
- En cas de modifications, une nouvelle version est créée

appelée itérations

- Une fois finalisé, déployez le produit

Le cycle SDLC est terminé



CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.2. Modèle Itératif

Collecte et analyse des besoins : Dans cette phase, toutes les exigences logicielles du client sont collectées et analysées pour savoir si ces exigences peuvent être satisfaites ou non. En outre, il est également vérifié si ce projet ne dépassera pas notre budget.

Conception : Dans cette phase, la conception du logiciel est préparée. Pour cela, divers diagrammes tels que le diagramme de flux de données, le diagramme de classes, le diagramme d'activité, le diagramme de transition d'état, etc. sont utilisés.

Implémentation : la conception du logiciel est implémentée dans le codage via divers langages de programmation.

Test : Une fois le codage du logiciel terminé, il est maintenant testé afin que les bugs et les erreurs qu'il contient puissent être identifiés.

Déploiement : le logiciel est remis au client. Après cela, le client commence à utiliser ce logiciel dans son environnement de travail.

Révision : Une fois le logiciel déployé dans son environnement de travail, il est révisé. Si une erreur/un bug est détecté ou si de nouvelles exigences sont présentées au développeur, ces phases sont à nouveau répétées avec une nouvelle itération et une nouvelle version est développée.

Maintenance : Dans cette phase, nous examinons les commentaires des clients, résolvons les problèmes, corrigeons les erreurs, mettons à jour le logiciel,

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.1. Modèle Itératif

- **Avantages du modèle itératif** :

✓ Flexibilité : les modifications peuvent être introduites à n'importe quelle étape du cycle de développement, ce qui le rend idéal pour les projets dont les exigences évoluent.

✓ Détection précoce des risques : les risques ou les problèmes sont identifiés et traités rapidement grâce à des tests et des retours d'information continus.

✓ Affinement progressif : le logiciel s'améliore progressivement, chaque itération rapprochant le projet du produit final.

✓ Retour d'information des clients : l'implication fréquente des clients garantit que le produit répond aux besoins et aux attentes des utilisateurs.

- **Inconvénients du modèle itératif :**

✓ Nécessite une bonne planification : une mauvaise planification peut entraîner une dérive du périmètre (changements incontrôlés ou croissance continue du périmètre d'un projet).

✓ Nécessite beaucoup de ressources : il peut nécessiter plus de ressources que d'autres modèles en raison de cycles répétés de conception, de développement et de test.

✓ Incomplet aux premières étapes : les itérations initiales peuvent ne pas fournir un produit complet ou entièrement fonctionnel, ce qui peut être difficile à évaluer pour les parties prenantes

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

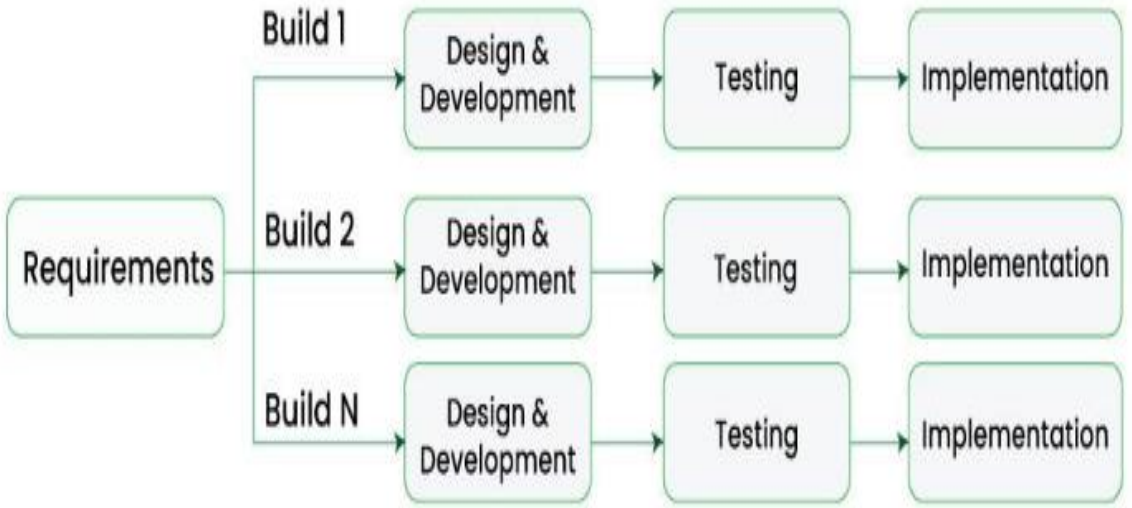
1.6.3. Modèle Incrémental

Dans le modèle incrémental, le processus de développement logiciel est divisé en plusieurs incréments et les mêmes phases sont suivies dans chaque incrément. En termes simples, dans ce modèle, un projet complexe est développé en plusieurs Modules ou builds.

Dans le modèle incrémental, les exigences sont divisées en plusieur modules autonomes du cycle de développement logiciel.

- ✓ Chaque module passe par les phases d'exigences, de conception d'implémentation et de test.
- ✓ Chaque version ultérieure du module ajoute une fonction à la version précédente.

Le processus se poursuit jusqu'à ce que le système complet soit atteint.



CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.3. Modèle Incrémental

Analyse des besoins : Dans la première phase du modèle incrémental, l'expertise en analyse de produit identifie les besoins. Et les exigences fonctionnelles du système sont comprises par l'équipe d'analyse des besoins.
Pour développer le logiciel sous le modèle incrémental, cette phase joue un rôle crucial.

Conception et développement : Dans cette phase du modèle incrémental du SDLC, la conception de la fonctionnalité du système et la méthode de développement sont terminées avec succès.

Test : Dans le modèle incrémental, la phase de test vérifie les performances de chaque fonction existante ainsi que les fonctio nnalités supplémentaires.

Implémentation : Cette phase permet la phase de codage du système de développement. Elle implique le codage final de la conception dans la phase de conception et de développement et teste la fonctionnalité dans la phase de test.
Une fois cette phase terminée, le nombre de produits fonctionnels est amélioré et mis à niveau jusqu'au produit système final.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

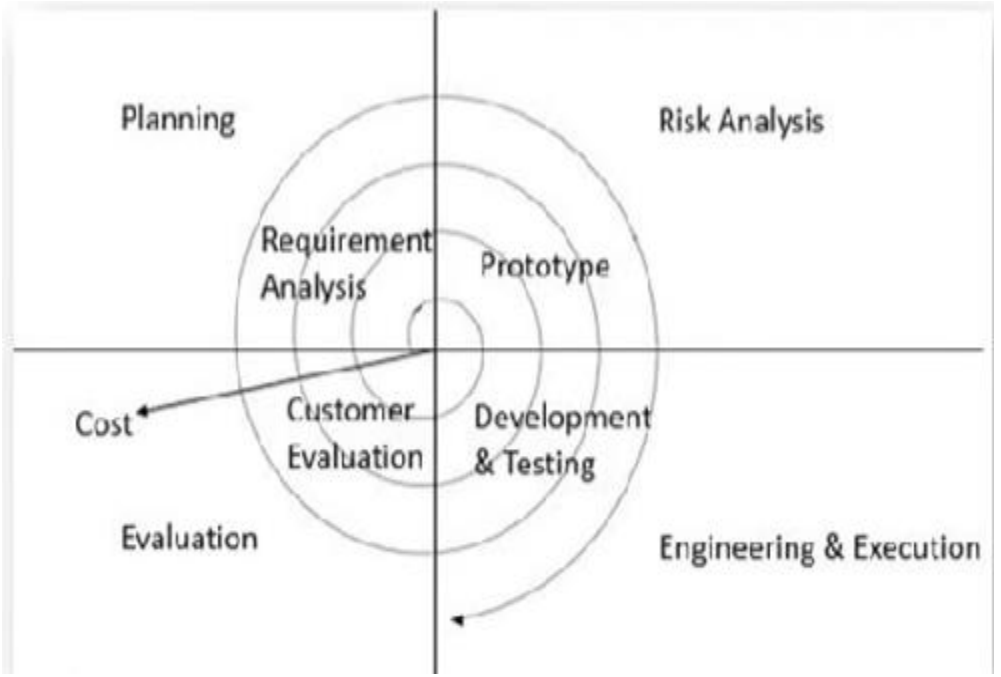
1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.4. Modèle Spirale

- Le modèle en spirale combine des éléments des modèles itératifs et en cascade.
- Il implique des cycles répétés (ou spirales) et est fortement axé sur l'évaluation des risques.

En utilisant le modèle en spirale, le logiciel est développé en une série de versions incrémentielles.
Au cours des premières itérations, la version supplémentaire peut être un modèle papier ou un prototype.
Au cours des itérations ultérieures, des versions de plus en plus complètes du système conçu sont produites.

Exemple : versions du système d'exploitation Microsoft, industrie du jeu



Le modèle en spirale est divisé en quatre parties :
1. Planification
2. Analyse des risques
3. Ingénierie et exécution
4. Évaluation du client

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

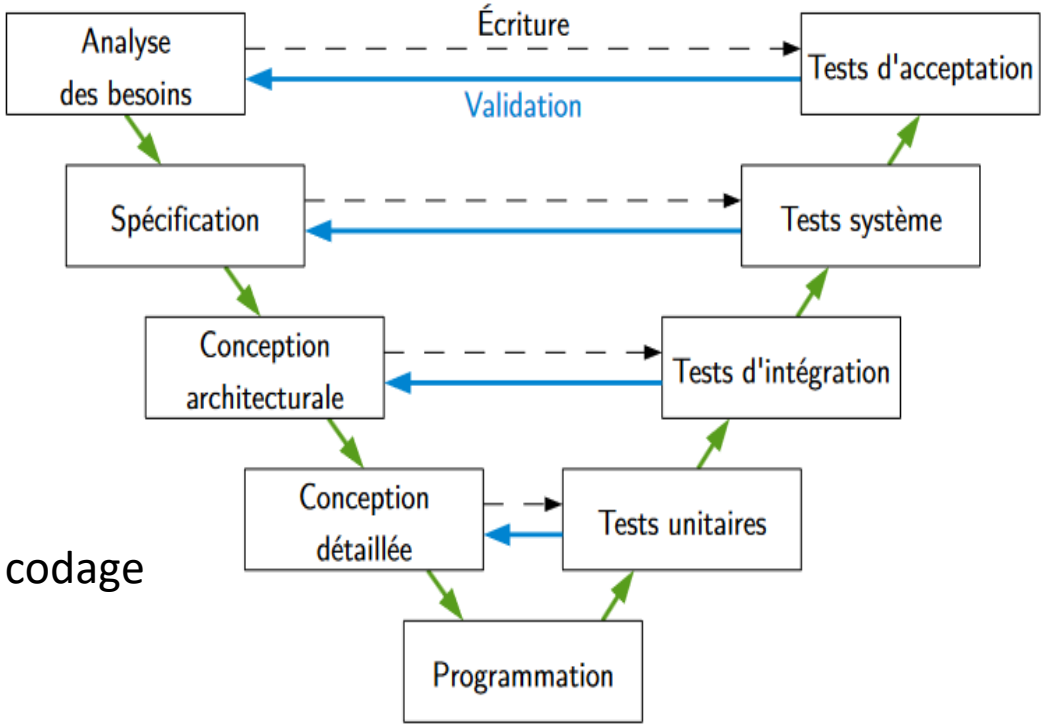
1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.5. Modèle V

- Le modèle en V est une extension du modèle en cascade, dans lequel les tests sont planifiés en parallèle avec les étapes de développement correspondantes. Il met l'accent sur la vérification (avant le codage) et la validation (après le codage).

Après chaque phase de développement, une phase de test lui est associée, et la phase suivante commencera une fois la phase précédente terminée, c'est-à-dire le développement et les tests.

Le modèle en V est également appelé modèle de vérification et de validation.



Le modèle en V contient donc des phases de vérification d'un côté et des phases de validation de l'autre côté.
Le processus de vérification et de validation est relié par une phase de codage en forme de V.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.5. Modèle V

Vérification : Il s'agit d'une méthode d'analyse statique (revue) effectuée sans exécuter de code.
Il s'agit du processus d'évaluation du processus de développement du produit pour déterminer si les exigences spécifié es sont satisfaites.

Validation : Il s'agit d'une méthode d'analyse dynamique (fonctionnelle, non fonctionnelle), les tests étant effectués en exécutant du code.
La validation est le processus de classification du logiciel après l'achèvement du processus de développement pour dét erminer si le logiciel répond aux attentes et aux exigences du client.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.5. Modèle V

Tests unitaires : Dans le modèle en V, les tests unitaires sont développés pendant la phase de conception du module.
Ces tests sont exécutés pour éliminer les erreurs au niveau du code ou de l'unité. Une unité est la plus petite entité qui peut exister indépendamment, par exemple un module de programme. Les tests unitaires vérifient que la plus petite entité peut fonctionner correctement lorsqu'elle est isolée du reste des codes/unités.

Tests d'intégration : Les plans de tests d'intégration sont développés pendant la phase de conception architecturale. Ces tes ts vérifient que les groupes créés et testés indépendamment peuvent coexister et communiquer entre eux.

Tests système : Les tests système sont développés pendant la phase de conception du système. Les tests système garantiss ent que les attentes d'un développeur d'applications sont satisfaites.

Tests d'acceptation : Les tests d'acceptation sont liés à la partie analyse des besoins métier.
Ils incluent le test du produit logiciel dans l'atmosphère utilisateur.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.6. Modèle Agile

- Le modèle Agile se concentre sur le développement itératif et la collaboration. De petites équipes interfonctionnelles travaillent en cycles courts appelés sprints pour livrer rapidement des logiciels fonctionnels.
- Phases : Exigences → Conception → Développement → Tests (en sprints itératifs)
- Avantages : Très flexible, encourage la collaboration des clients et s'adapte aux changements.
- Inconvénients : Nécessite une interaction étroite avec les clients et peut parfois manquer de structure claire.

Scrum (Framework au sein d'Agile)

Scrum est un framework Agile spécifique axé sur de petites équipes interfonctionnelles travaillant sur des itérations courtes et ciblées (généralement de 2 à 4 semaines).

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

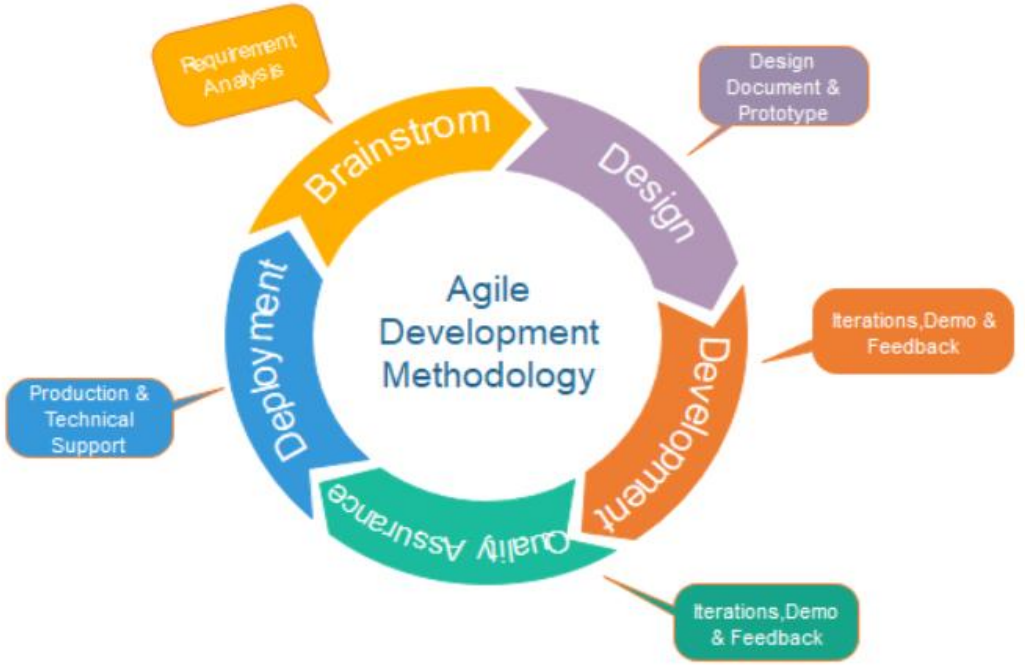
1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.6. Modèle Agile

La division de l'ensemble du projet en parties plus petites permet de minimiser le risque du projet et de réduire les exigences globales en matière de délai de livraison du projet. Chaque itération implique une équipe travaillant sur un cycle de vie comp let de développement logiciel, y compris la planification, l'analyse des exigences, la conception, le codage et les tests avant q u'un produit fonctionnel ne soit présenté au client.

Les phases du modèle Agile sont les suivantes :

- ✓ Collecte des exigences
- ✓ Conception des exigences
- ✓ Construction/itération
- ✓ Test/Assurance qualité
- ✓ Déploiement
- ✓ Retour d'information



CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.6. Modèle Agile

1. Collecte des exigences : Dans cette phase, vous devez définir les exigences. Vous devez expliquer les opportunités comme rciales et planifier le temps et les efforts nécessaires à la construction du projet.

2. Concevoir les exigences : Une fois le projet identifié, travaillez avec les parties prenantes pour définir les exigences. Vous pouvez utiliser le diagramme de flux utilisateur ou le diagramme UML de haut niveau pour montrer le travail des nouvelles fonctionnalités et montrer comment elles s'appliqueront à votre système existant.

3. Construction/itération : Lorsque l'équipe définit les exigences, le travail commence. Les concepteurs et les développeurs co mmencent à travailler sur leur projet, qui vise à déployer un produit fonctionnel.

4. Test : Dans cette phase, l'équipe d'assurance qualité examine les performances du produit et recherche le bug.

5. Déploiement : Dans cette phase, l'équipe publie un produit pour l'environnement de travail de l'utilisateur.

6. Retour d'information : Après la sortie du produit, la dernière étape est le retour d'information. Dans ce cadre, l'équipe reço it des commentaires sur le produit et travaille sur ces commentaires.

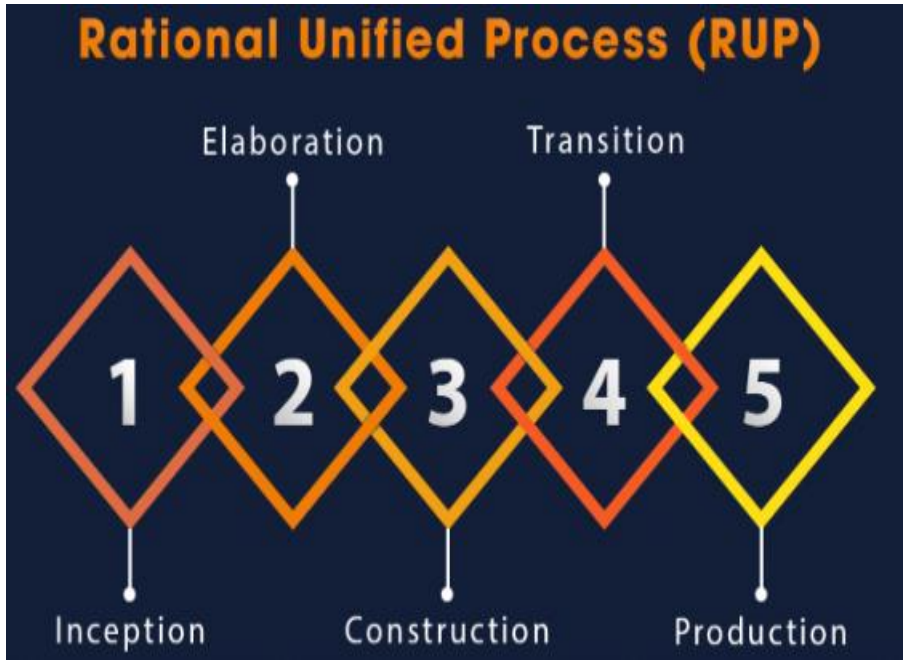
CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.6. Modèle Agile

Les méthodologies de l'approches agiles du développement logiciel :

- ✓ Scrum
- ✓ XP (eXtreme Programming)
- ✓ RUP (Rational Unified Process)
- ✓ UP (Unified Process)
- ✓ Crystal
- ✓ Dynamic Software Development Method(DSDM)
- ✓ Feature Driven Development(FDD)
- ✓ Lean Software Development



CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.6. Modèle Agile

✓ **Scrum**

SCRUM est un processus de développement agile axé principalement sur les moyens de gérer les tâches dans des conditions de développement en équipe.

Il comporte trois rôles et leurs responsabilités sont les suivantes :

Scrum Master : le Scrum peut mettre en place l'équipe principale, organiser la réunion et éliminer les obstacles au processus.

Product Owner : le Product Owner établit le backlog du produit, hiérarchise les retards et est responsable de la distribution des fonctionnalités à chaque répétition.

Scrum Team : l'équipe gère son travail et organise le travail pour terminer le sprint ou le cycle.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.6. Modèle Agile

✓ **Extreme Programming (XP)**

❑ **XP est une méthodologie agile axée sur les valeurs de communication, simplicité, feedback rapide, courage et respect.**
Elle vise à livrer des versions fréquentes et fonctionnelles du produit.

❑ XP ne suit pas une structure de phases spécifique comme RUP ou UP, mais plutôt des itérations rapides et des livraisons fréquentes.
❑ Accent mis sur la communication constante avec le client et la flexibilité pour s'adapter aux changements fréquents.
❑ Caractéristiques :Structure basée sur des valeurs et principes agiles (ex. : satisfaction du client, adaptation rapide).
❑ Des cycles de développement très courts (itérations de 1 à 2 semaines).
❑ Collaboration étroite avec les parties prenantes et forte implication des développeurs.

❑ **Avantages** : Très agile et adaptatif, permettant une haute réactivité aux changements.
❑ I**nconvénients** : Peut être difficile à gérer pour les grandes équipes ou les projets avec des exigences rigides et peu évolutives.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.6. Modèle Agile

✓ **Rational Unified Process (RUP)**

❑ **RUP est une méthodologie de développement itérative créée par Rational Software (maintenant IBM).**
Elle est structurée autour d'un cadre flexible mais prescriptif, fournissant des directives, des modèles et des outils pour gérer des projets logiciels de grande envergure.

Inception : Définir la portée, les risques et les besoins des parties prenantes.
Elaboration : Construire l'architecture de base, affiner les exigences et planifier le projet.
Construction : Développer l'application avec des itérations, ajoutant progressivement des fonctionnalités.
Transition : Déployer le produit, gérer les tests finaux et préparer la version pour les utilisateurs.
Caractéristiques : Basé sur des cas d'utilisation et l'architecture logicielle.

Convient aux projets complexes et aux équipes importantes.
Avantages : Très structuré et documenté, excellent pour les projets qui nécessitent un haut degré de contrôle et de suivi.
Inconvénients : Peut être lourd, surtout pour les petites équipes ou les projets simples.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.6. Modèle Agile

✓ **Unified Process (UP)**

❑ **Unified Process (UP)Description : UP est le cadre général dont RUP est une version plus formelle.**

Il est itératif et incrémental, guidé par une architecture bien définie.

UP suit également les quatre phases principales de RUP (Inception, Elaboration, Construction, Transition), avec une approche similaire.

❑ Caractéristiques :Propose une approche itérative pour gérer les exigences changeantes.

❑ Architecture centrée et basée sur des cas d'utilisation.

Avantages : Plus flexible et adaptable que RUP, mais tout aussi structuré.
Inconvénients : Peut manquer de détails concrets comparé à RUP, nécessitant parfois plus d'adaptation et de personnalisation.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

1.6.7. Modèle RAD

- Le modèle de développement rapide d'applications (RAD) est une approche SDLC qui met l'accent sur le développement rapide et les retours des utilisateurs au cours de longues phases de planification.

Il se concentre sur la création de prototypes et sur l'obtention de retours précoces des utilisateurs pour affiner et améliorer le produit de manière itérative.

- Ce modèle est idéal pour les projets dont les objectifs sont bien définis mais dont les exigences peuvent évoluer au fur et à mesure de l'avancement du projet.

1.6.8. Modèle Big Bang

Le modèle Big Bang dans SDLC est un terme utilisé pour décrire une approche informelle et non structurée du développement logiciel, où il n'y a pas de planification spécifique, de documentation ou de phases bien définies.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

Comment choisir un modèle SDLC ?

Le choix du bon modèle SDLC (Software Development Life Cycle) est essentiel pour la réussite d'un projet.
Voici les facteurs clés à prendre en compte :

1. Exigences du projet :

- ✓ **Exigences claires** : utilisez le modèle en **cascade** ou en **V** si les exigences sont bien définies et peu susceptibles de changer.
- ✓ **Exigences changeantes** : utilisez les modèles **Agile** ou **itératif** si les exigences ne sont pas claires ou susceptibles d'évoluer.

2. Taille et complexité du projet :

- ✓ **Petits projets** : utilisez le modèle en **cascade** ou **RAD** pour les petits projets simples.
- ✓ **Grands projets** : utilisez Agile ou Spiral pour les projets complexes et de grande envergure qui nécessitent de la flexibilité.

3. Expertise de l'équipe :

- ✓ **Équipes expérimentées** : utilisez **Agile** ou **Scrum** si l'équipe est familière avec le développement itératif.
- ✓ **Équipes moins expérimentées** : utilisez le modèle en cascade ou en V pour les équipes ayant besoin de conseils structurés.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

Comment choisir un modèle SDLC ?

4. Implication du client :

- ✓ **Commentaires fréquents des clients** : utilisez **Agile**, **Scrum** ou **RAD** si une interaction régulière avec le client est nécessaire.
- ✓ **Implication minimale du client** : utilisez **Waterfall** ou le modèle en **V** si l'implication du client est faible après la planification initiale.

5. Contraintes de temps et de budget :

- ✓ **Délai et budget fixes** : utilisez **Waterfall** ou le modèle en **V** si vous avez des limites de temps et de budget strictes.
- ✓ **Délai et budget flexibles** : utilisez Agile ou Spiral si vous pouvez ajuster le temps et le budget selon vos besoins.

6. Gestion des risques :

- ✓ **Projets à haut risque** : utilisez **Spirale** pour les projets présentant des risques et des incertitudes importants.
- ✓ **Projets à faible risque** : utilisez **Waterfall** pour les projets présentant des risques minimes.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

Comment choisir un modèle SDLC ?

7. Calendrier de sortie du produit :

- ✓ **Lancement rapide nécessaire** : utilisez Agile ou RAD pour livrer les produits rapidement.
- ✓ **Durée de développement plus longue** : utilisez Waterfall ou V-Model pour les projets sans délais urgents.

8. Maintenance et support :

- ✓ **Maintenance à long terme** : utilisez Agile ou DevOps pour les projets nécessitant des mises à jour et un support continus.
- ✓ **Maintenance minimale** : utilisez Waterfall ou V-Model si peu de maintenance future est prévue.

9. Attentes des parties prenantes :

- ✓ **Engagement élevé des parties prenantes** : utilisez Agile ou Scrum si les parties prenantes souhaitent une implication continue.
- ✓ **Faible engagement des parties prenantes** : utilisez Waterfall ou V-Model si les parties prenantes préfèrent s'impliquer uniquement lors des étapes importantes.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.6. Modèles de cycle de vie du développement logiciel [SDLC Model]

En bref :

- **Waterfall** : idéal pour les projets clairs et stables avec un minimum de modifications.
- **V-Model** : idéal pour les projets avec des exigences claires et une forte concentration sur les tests.
- **Agile/Scrum** : idéal pour les projets avec des exigences changeantes et une interaction fréquente avec les clients.
- **Spirale** : adapté aux projets à haut risque avec des exigences évolutives.
- **RAD** : utile pour les projets nécessitant un développement rapide..

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.7. Qualité du logiciel

La qualité d'un produit logiciel est définie en termes d'adéquation à l'usage prévu.
En d'autres termes, un produit de qualité fait exactement ce que les utilisateurs veulent qu'il fasse.

La vision moderne de la qualité associée à un produit logiciel comprend plusieurs méthodes de qualité telles que les suivantes :

Portabilité : un logiciel est dit portable s'il peut être librement conçu pour fonctionner dans divers environnements de système d'exploitation, sur plusieurs machines, avec d'autres produits logiciels.
Utilisabilité : un produit logiciel est plus facile à utiliser si différentes catégories d'utilisateurs peuvent facilement invoquer les fonctions du produit.

Réutilisabilité : un logiciel est très facile à réutiliser si différents modules du produit peuvent être rapidement réutilisés pour développer de nouveaux produits.

Exactitude : un logiciel est correct si diverses exigences spécifiées dans le document ont été correctement mises en œuvre.

Maintenabilité : un logiciel est maintenable si les bogues peuvent être facilement corrigés au fur et à mesure qu'ils apparaissent, si de nouvelles tâches peuvent être facilement ajoutées au produit et si les fonctionnalités du produit peuvent être facilement modifiées.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.8. Gestion de projet logiciel

Qu'est-ce qu'un projet ?

Un projet est un groupe de tâches qui doivent être accomplies pour atteindre un résultat clair.

Les projets sont généralement décrits et approuvés par un chef de projet ou un responsable d'équipe.

Pour un bon développement de projet, certaines équipes divisent le projet en tâches spécifiques afin de pouvoir gérer la responsabilité et utiliser les points forts de l'équipe.

Qu'est-ce que la gestion de projet logiciel ?

- La gestion de projet logiciel est un art et une discipline de planification et de supervision de projets logiciels.

C'est une sous-discipline de la gestion de projet logiciel dans laquelle les projets logiciels sont planifiés, mis en œuvre, surveillés et contrôlés.
Il s'agit d'une procédure de gestion, d'allocation et de chronométrage des ressources pour développer des logiciels informatiques qui répondent aux exigences.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.8. Gestion de projet logiciel

Dans la gestion de projets logiciels, le client et les développeurs doivent connaître la durée, la période et le coût du projet.

La gestion de projets logiciels a trois besoins. Ce sont :

- ✓ Temps
- ✓ Coût
- ✓ Qualité

Il est essentiel pour l'organisation logicielle de livrer un produit de qualité, de maintenir le coût dans les limites du budget du client et de livrer le projet dans les délais.

Chef de projet [Project Manage]

Un chef de projet est un personnage qui a la responsabilité globale de la planification, de la conception, de l'exécution, du suivi, du contrôle et de la clôture d'un projet. Un chef de projet représente un rôle essentiel dans la réalisation des projets.

Un chef de projet est un personnage qui est chargé de prendre des décisions, qu'il s'agisse de projets de grande ou de petite envergure. Le chef de projet est utilisé pour gérer le risque et minimiser l'incertitude. Chaque décision prise par le chef de projet doit profiter directement à son projet.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.8. Gestion de projet logiciel

Rôle d'un chef de projet

1. Leader

Un chef de projet doit diriger son équipe et lui fournir des directives pour lui faire comprendre ce qui est attendu de chacun d'eux.

2. Médiateur :

Le chef de projet est un médiateur entre ses clients et son équipe.
Il doit coordonner et transférer toutes les informations appropriées des clients à son équipe et rendre compte à la haute direction.

3. Mentor :

Il doit être là pour guider son équipe à chaque étape et s'assurer que l'équipe a un attachement.
Il fournit une recommandation à son équipe et l'oriente dans la bonne direction.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.8. Gestion de projet logiciel

Responsabilités d'un chef de projet :

- ✓ Gérer les risques et les problèmes.
- ✓ Créer l'équipe de projet et attribuer les tâches à plusieurs membres de l'équipe.
- ✓ Planifier et séquencer les activités.
- ✓ Suivre et rendre compte de l'avancement.
- ✓ Modifier le plan de projet pour faire face à la situation.

Activités

La gestion de projet logiciel comprend de nombreuses activités, notamment:

- ✓ Planification et suivi du projet
- ✓ Gestion des ressources du projet
- ✓ Gestion de la portée
- ✓ Gestion des estimations
- ✓ Gestion des risques du projet
- ✓ Gestion de la planification
- ✓ Gestion de la communication du projet

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.8. Gestion de projet logiciel

Outils de gestion de projet

Pour gérer le système de gestion de projet de manière adéquate et efficace, nous utilisons des outils de gestion de projet.

Voici quelques outils standard :Diagramme de Gantt, PERT chart, Logic Network,r etc.

Le diagramme de Gantt a été développé pour la première fois par Henry Gantt en 1917.

Le diagramme de Gantt est généralement utilisé dans la gestion de projet et constitue l'un des moyens les plus populaires et les plus utiles pour afficher les activités en fonction du temps. Chaque activité est représentée par une barre.

Le diagramme de Gantt est un outil utile lorsque vous souhaitez voir l'ensemble du paysage d'un ou de plusieurs projets.

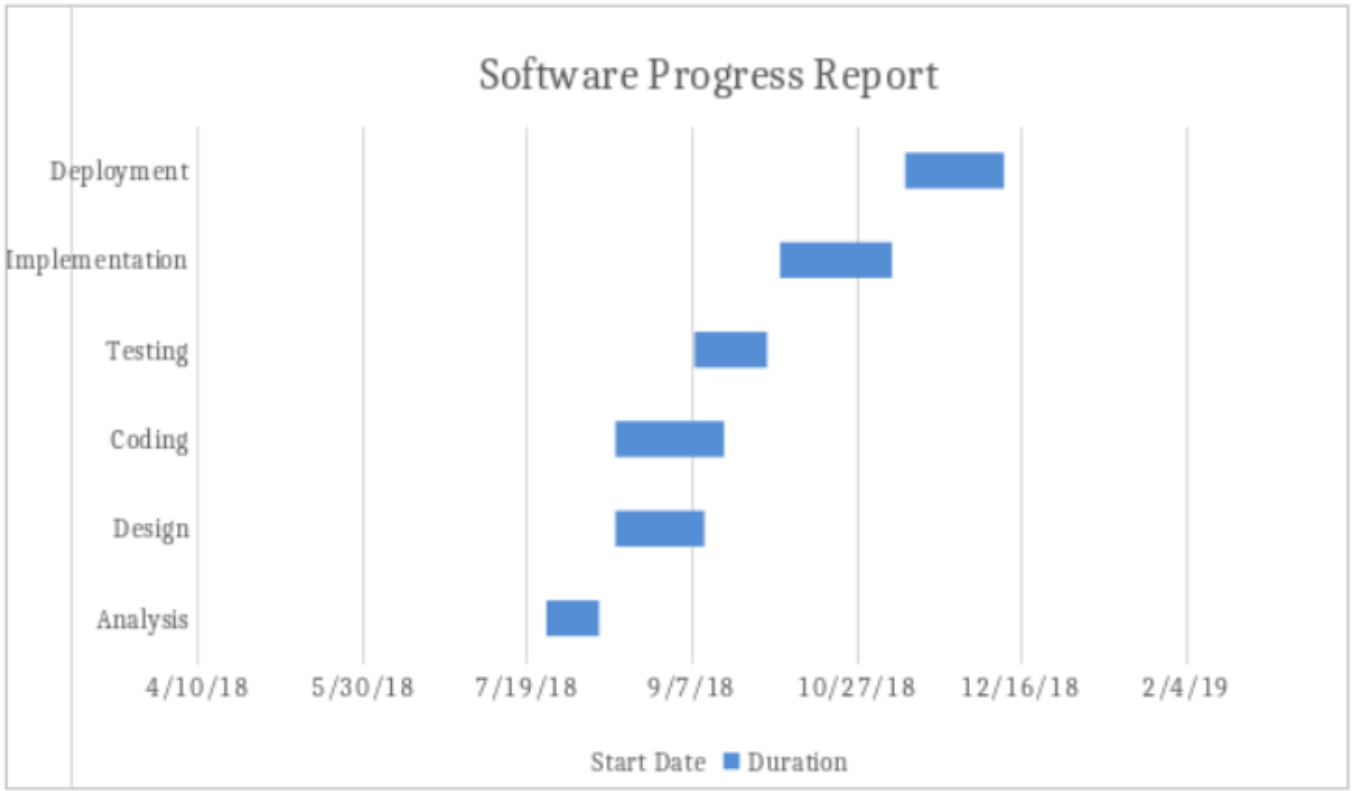
Il vous aide à visualiser les tâches qui dépendent les unes des autres et l'événement à venir.

CHAPITRE 1. INTRODUCTION AU GENIE LOGICIEL

1.8. Gestion de projet logiciel

Outils de gestion de projet

Diagramme de Gantt



Fin Chapitre I