Chapitre 2: Cycle de vie du Logiciel

Le cycle de vie de développement des logiciels (SDLC, *Software Development Life Cycle*) est un processus utilisé par l'industrie du logiciel pour concevoir, développer et tester des logiciels de haute qualité qui répondent ou dépassent les attentes des clients en respectant les délais et les coûts requises. Le SDLC est un cadre définissant les tâches effectuées à chaque étape du processus de développement logiciel. L'ISO/IEC 12207 est une norme internationale pour les cycles de vie du logiciel qui vise à définir toutes les tâches requises pour le développement et la maintenance des logiciels.

I. SDLC et ses activités?

Le SDLC consiste en un plan détaillé décrivant comment développer, maintenir, remplacer et modifier ou améliorer des logiciels spécifiques. Un cycle de vie de développement de logiciel typique comprend les étapes suivantes :

Étape 1: Planification et analyse des besoins (exigences)

L'analyse des exigences est l'étape la plus importante et la plus fondamentale de SDLC. Elle est effectuée par les **membres seniors** de l'équipe avec les contributions **du client**, du **service des ventes**, des **études de marché** et des **experts du domaine** de l'industrie. Cette information est ensuite utilisée pour planifier l'approche de base du projet et pour mener **l'étude de faisabilité** du produit dans les domaines économiques, opérationnels et techniques.

La planification des exigences pour l'assurance qualité et l'identification des risques associés au projet se font également à cette étape. Le résultat de l'étude de faisabilité technique est de définir les différentes approches techniques qui peuvent être suivies pour mener à bien le projet avec un minimum de risques.

Étape 2: Définition des exigences

Une fois l'analyse des besoins effectuée, l'étape suivante consiste à définir et documenter clairement les exigences du produit et à les faire approuver par le client ou les analystes du marché. Cette phase est difficile car le client et les développeurs ne parlent pas le même langage. Le livrable de cette phase est un document de spécification des exigences SRS (*Software Requirement Specification*) qui comprend toutes les exigences du produit à concevoir et développer pendant le cycle de vie du projet.

Étape 3: Conception de l'architecture du produit

Le SRS est un document de référence pour les architectes du produit afin qu'ils puissent proposer la meilleure architecture pour le produit à développer. Sur la base des exigences spécifiées dans le SRS, plus d'une approche de conception de l'architecture du produit est proposée et documentée dans un document de spécification de conception DDS (*Design Document Specification*). Ce DDS est revu par tous les acteurs importants et sur la base de divers paramètres tels que l'évaluation des risques, la robustesse du produit, la modularité du design, les contraintes budgétaires et temporelles, la meilleure approche de conception est sélectionnée pour le produit.

Une approche de conception définit clairement tous les modules architecturaux du produit ainsi que sa représentation de communication et de flux de données avec les modules externes et tiers (le cas échéant). La conception interne de tous les modules de l'architecture proposée doit être clairement définie avec le plus petit des détails possibles dans le DDS. Si le produit est centré sur l'utilisateur, la conception propose une ébauche de l'interface utilisateur.

Étape 4: Construction ou le développement du produit

Dans cette étape du SDLC, le développement réel commence et le produit est construit. Le code de programmation est généré selon le DDS. Le codage transforme les solutions proposées lors de la conception en un code opérationnel. La conception et le codage peuvent toutes les deux produire du code source, mais c'est l'étape de codage qui rend ce code opérationnel. Les développeurs doivent suivre les directives de codage définies par leur organisation. Les outils de programmation tels que les compilateurs, les interprètes, les débogueurs, etc. sont utilisés pour générer le code. Différents langages de programmation de haut niveau tels que C, C ++, Java et PHP sont utilisés pour le codage. Le langage de programmation est choisi en fonction du type de logiciel en cours de développement.

Étape 5: Test du produit

Les tests déterminent la qualité du logiciel. Dans les modèles modernes de SDLC, le test est généralement un sous-ensemble de toutes les étapes. Ils déterminent si le logiciel fait ce qu'on attend de lui par rapport aux spécifications et normes de qualité définies dans le SRS. Parmi les types de test, deux principaux sont utilisés : Les **tests unitaires** orientés code qui se rédigent durant l'activité de codage et se revérifient pendant la phase de test; **Les tests d'acceptation** qui vérifient les attentes d'un produit logiciel.

Étape 6: Déploiement sur le marché et maintenance

Une fois le produit testé et prêt à être déployé, il est officiellement commercialisé sur le marché approprié. Parfois, le déploiement du produit se fait par étapes selon la stratégie commerciale de l'organisation. Le produit peut d'abord être lancé dans un segment limité et testé dans l'environnement réel de l'entreprise (test d'acceptation de l'utilisateur – UAT (User acceptance testing)).

Ensuite, sur la base des commentaires, le produit peut être publié tel quel ou avec des améliorations suggérées dans le segment de marché de ciblage. Après la sortie du produit sur le marché, sa maintenance est effectuée pour la clientèle existante.

Pour faire simple, nous dirons que, tous les projets de développement ont des activités communes qui sont l'**Analyse de besoins, la Conception, le Codage, le Tests et la Maintenance.**

II. Modèles de cycle de vie (SDLC

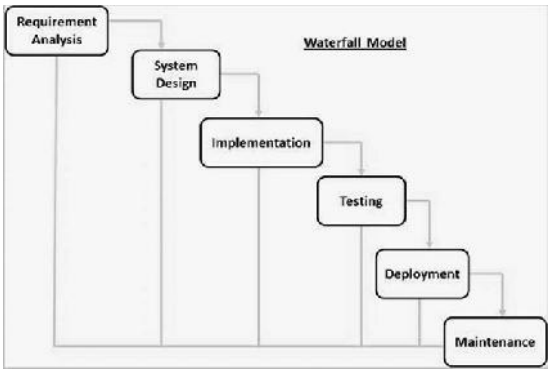
Plusieurs modèles de cycle de vie de développement de logiciel sont utilisés pendant le processus de développement logiciel. Chaque modèle de processus suit une série d'étapes uniques en son genre pour assurer le succès du projet. Deux grandes philosophies ou écoles de pensée permettent de les classifier :

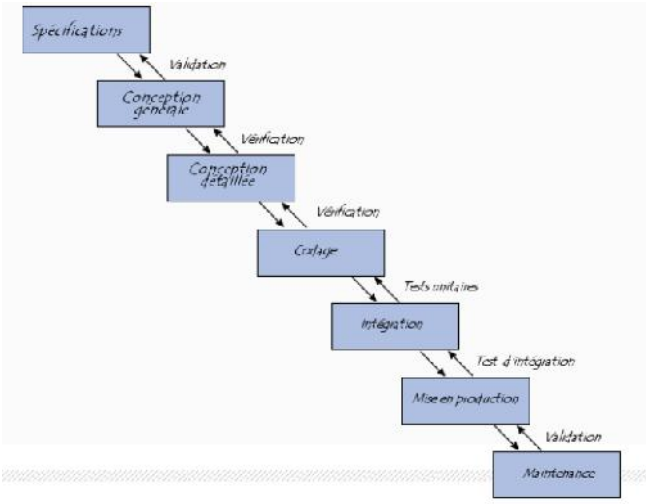
- La **philosophie classique (Modèles classiques)** adaptée aux gros projets et les projets gouvernementaux qui est stricte et exige des étapes très clairement définies et une Documentation très fournie. Des exemples sont le **Modèle en cascade**, le **modèle en V**, le **prototypage,** etc.
- La **philosophie agile (Modèles agiles)** qui convient aux projets de petite et moyenne taille. Ces modèles sont incrémentaux et itératifs avec des petites et fréquentes livraisons et dont l'accent est mis sur le code et moins sur la documentation. Des exemples sont **Scrum**, **XP** (eXtreme Programming), **UP** (Unified Process)**,** etc.

Les modèles de SDLC les plus importants et populaires de suivis dans l'industrie de production de logiciels sont les suivantes.

a) Modèle en cascade (Waterfall) - Méthodes classiques

L'approche en cascade a été l'un des premiers modèles SDLC à être largement utilisé en génie logiciel pour assurer le succès du projet. Dans cette approche, l'ensemble du processus de développement logiciel est divisé en phases distinctes. Le résultat d'une phase agit généralement comme une entrée séquentielle pour la phase suivante. Les phases de SDLC fonctionneront les unes après les autres de manière linéaire. C'est-à-dire que lorsque la première phase est terminée, seule la deuxième phase commence et ainsi de suite. L'illustration suivante est une représentation des différentes phases du modèle en cascade.





- **Analyse des exigences -** Toutes les exigences possibles du système à développer sont capturées dans cette phase et documentées dans un document de spécification des exigences.
- **Conception du système -** Les spécifications des exigences de la première phase sont étudiées dans cette phase et la conception du système est amorcée. Cette conception du système aide à spécifier les exigences matérielles et système et aide à définir l'architecture globale du système.

- **Implémentation** - Avec les entrées de la conception du système, le système est d'abord développé dans de petits programmes appelés unités, qui sont intégrés dans la phase suivante. Chaque unité est développée et testée suivant ses fonctionnalités.
- **Intégration et test** - Toutes les unités développées dans la phase d'implémentation sont intégrées dans un système après l'essai de chaque unité. Après l'intégration, l'ensemble du système est testé pour détecter toutes fautes et défaillances.
- **Déploiement du système** - Une fois les tests fonctionnels et non-fonctionnels effectués; le produit est déployé dans l'environnement du client ou sur le marché.
- **Maintenance** – Permet de faire des corrections (maintenance corrective) ou améliorer le produit pour (maintenance évolutive).

i. Quand l'utiliser ?

Chaque logiciel développé est différent et nécessite une approche SDLC appropriée à suivre en fonction des facteurs internes et externes. Certaines situations où l'utilisation du modèle en cascade est la plus appropriée sont :

- Les exigences sont très bien documentées, claires, fixes et stables
- La technologie est comprise, maîtrisée et n'est pas dynamique.
- De nombreuses ressources avec l'expertise requise sont disponibles pour soutenir le produit.
- Lors de la création d'une nouvelle version d'un produit
- Le projet est court.

ii. Avantages

Avec le modèle en cascade un calendrier peut être fixé avec des délais pour chaque étape de développement et un produit peut passer par les phases du modèle du processus de développement un par un. Certains des principaux avantages sont:

- Simple et facile à comprendre et à utiliser
- Idéal pour la gestion et le suivi de projets en raison de la rigidité du modèle.
- Un procédé structuré pour une équipe inexpérimentée
- Fonctionne très bien quand la qualité est plus importante que les coûts et les délais
- Le processus et les résultats sont bien documentés.

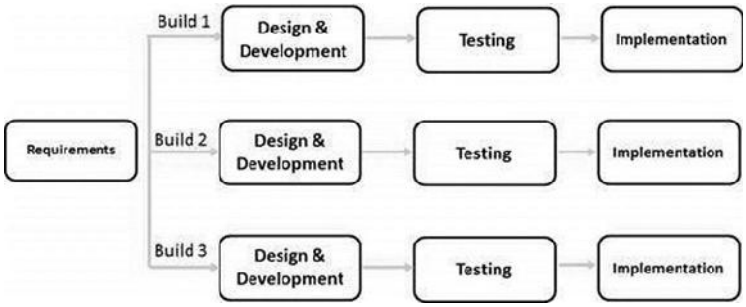
iii. Inconvénients

Le modèle en cascade ne permet pas beaucoup de réflexions ou de révisions. Une fois qu'une application est en phase de test, il est très difficile de revenir en arrière et de modifier quelque chose qui n'était pas bien documenté. Ses principaux inconvénients sont les suivants :

- Le produit n'est visible qu'à la fin
- De grandes quantités de risque et d'incertitude. Les risques se décalent vers la fin
- Très faible implication du client
- Les besoins des clients sont très rarement stables et clairement définis
- Sensibilité aux nouveaux besoins : refaire tout le procédé ;
- Ajuster la portée pendant le cycle de vie peut mettre fin à un projet.

b) Modèle itératif -Agile

Dans le modèle itératif, le processus itératif commence par une implémentation simple d'un petit ensemble d'exigences logicielles et améliore de manière itérative les versions évolutives jusqu'à ce que le système complet soit implémenté et prêt à être déployé. Un modèle de cycle de vie itératif ne cherche pas à faire une spécification complète des exigences. Au lieu de cela, le développement commence par la spécification et la mise en œuvre d'une partie seulement du logiciel, qui est ensuite examinée pour identifier d'autres exigences. **L'idée de base de cette méthode est de développer un système à travers des cycles répétés (itératifs) et en petites portions à la fois (incrémentales).**



Le développement itératif et incrémental est une combinaison de la conception itérative ou et de modèle de construction incrémental pour le développement. "Pendant le développement du logiciel, plus d'une itération du cycle de développement logiciel peut être en cours en même temps." Dans ce modèle incrémental, l'ensemble de besoins est divisé en différentes constructions. Au cours de chaque itération, le module de développement passe par les phases d'exigences, de conception, de mise en œuvre et de test. Chaque version suivante du module ajoute une fonction à la version précédente. Le processus se poursuit jusqu'à ce que le système complet soit prêt selon les exigences.

La clé d'une utilisation réussie d'un cycle de vie de développement logiciel itératif est la validation rigoureuse des exigences et la vérification et le test de chaque version du logiciel par rapport à ces exigences dans chaque cycle du modèle. Comme le logiciel évolue au cours des cycles successifs, les tests doivent être répétés et étendus pour vérifier chaque version du logiciel.

i. Quand l'utiliser ?

L'approche itératif et incrémental est le plus souvent utilisé dans les scénarios suivants :

- Les exigences du système sont clairement définies et comprises
- Les principales exigences doivent être définies. Cependant, certaines fonctionnalités ou améliorations demandées peuvent évoluer avec le temps.
- Il y a un temps lié à la contrainte du marché.
- Une nouvelle technologie est utilisée et est apprise par l'équipe de développement tout en travaillant sur le projet.
- Les ressources avec les compétences nécessaires ne sont pas disponibles et doivent être utilisées à contrat pour des itérations spécifiques.
- Certaines caractéristiques et objectifs à risque élevé peuvent changer à l'avenir.

ii. Avantages

L'avantage de ce modèle est qu'il existe une partie fonctionnelle du système à un stade de développement très précoce, ce qui facilite la recherche de défauts fonctionnels ou de conception. La recherche de problèmes à ce stade permet de prendre des mesures correctives dans un budget limité. Les avantages du modèle itératif et incrémental sont les suivants :

- Certaines fonctionnalités peuvent être développées rapidement et tôt.
- Les résultats sont obtenus tôt et périodiquement.
- Le développement parallèle peut être planifié.
- Les progrès peuvent être mesurés.
- Moins coûteux de changer la portée / les exigences.
- Le test et le débogage lors d'une itération plus petite est facile.
- Les risques sont identifiés et résolus pendant l'itération; et chaque itération est une étape facile à gérer.
- Plus facile à gérer le risque - Une partie à haut risque est effectuée en premier.
- Avec chaque incrément, le produit opérationnel est livré.
- Les problèmes, les défis et les risques identifiés à partir de chaque incrément peuvent être utilisés / appliqués à l'incrément suivant.
- Etc.

iii. Inconvénients

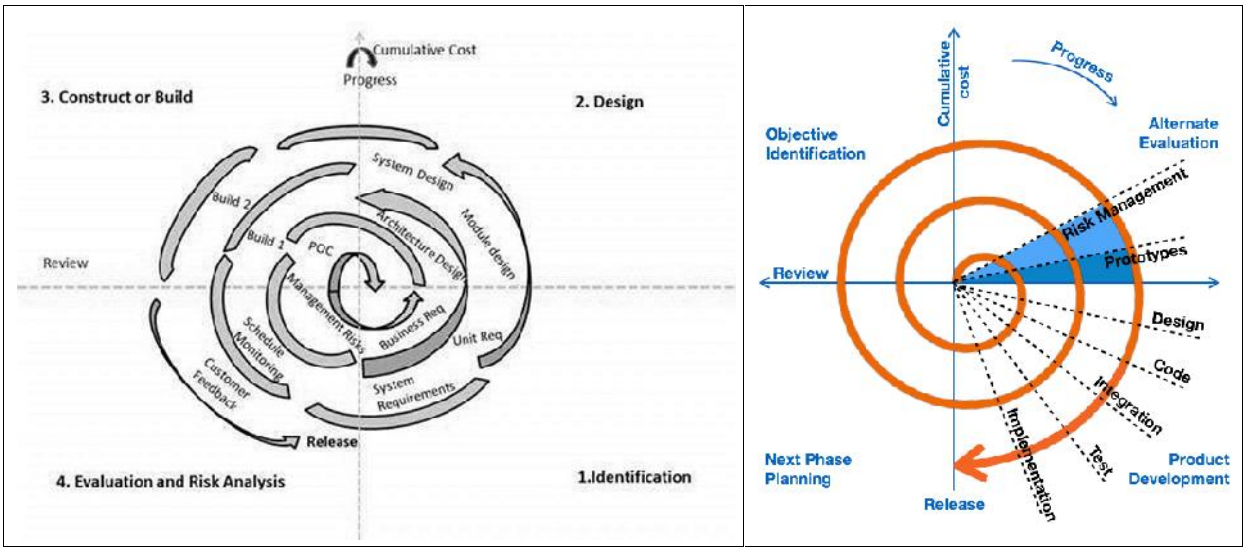
L'inconvénient de ce modèle SDLC est qu'il est applicable uniquement aux grands et volumineux projets de développement de logiciels. En effet, il est difficile de diviser un petit système logiciel en plusieurs petits incréments / modules réparables. Ces inconvénients sont :

- Plus de ressources peuvent être nécessaires.
- Bien que le coût du changement soit moindre, il n'est pas très adapté à l'évolution des besoins.
- Exige plus d'attention dans la gestion.
- Des problèmes d'architecture ou de conception du système peuvent survenir car toutes les exigences ne sont pas rassemblées au début du cycle de vie.
- La définition d'incréments peut nécessiter la définition du système complet.
- La fin du projet peut ne pas être connue, ce qui constitue un risque.
- Des ressources hautement qualifiées sont nécessaires pour l'analyse des risques.
- La progression des projets dépend fortement de la phase d'analyse des risques.

c) Modèle en spirale

Le modèle en spirale combine l'idée de développement itératif et d'un de modèles de SDLC. Cela peut être vu comme si vous choisissez un modèle SDLC et le combinez avec un processus cyclique (modèle itératif). Autre ment dit, ce modèle spiral est une combinaison du modèle itératif et du modèle de développement linéaire séquentiel, c'est-à-dire le modèle en cascade, qui met l'accent sur l'analyse des risques. Il permet la génération des incréments du produit ou un raffinement incrémental à chaque itération autour de la spirale. Le modèle en

spirale comporte quatre phases. Un projet logiciel passe plusieurs fois par ces phases dans des itérations appelées Spirales.



Identification -

Cette phase commence par la collecte des besoins de l'entreprise dans la spirale de base. Dans les spirales suivantes, au fur et à mesure que le produit mûrit, l'identification des exigences du système, des exigences du sous-système et des besoins unitaires est effectuée. Cette phase comprend également la compréhension des exigences du système par une communication continue entre le client et l'analyste de système.

Conception

La phase de conception commence par la conception dans la spirale de base et implique la conception architecturale, la conception logique des modules, la conception physique du produit et la conception finale dans les spirales suivantes.

Construction

La phase de construction fait référence à la production du produit logiciel réel à chaque spirale. Dans la spirale de référence, lorsque le produit est pensé et que le design est en cours de développement, un POC (Proof of Concept) est développé dans cette phase pour obtenir les commentaires des clients. Ensuite, dans les spirales suivantes avec une plus grande clarté sur les exigences et les détails de conception, un logiciel fonctionnel est produit avec un numéro de version. Ces versions sont envoyées au client pour obtenir des commentaires.

Évaluation et analyse des risques

L'analyse des risques comprend l'identification, l'estimation et le suivi de la faisabilité technique et des risques de gestion, tels que les retards de calendrier et les dépassements de coûts. Après avoir testé la construction, à la fin de la première itération, le client évalue le logiciel et fournit des commentaires.

i. Quand l'utiliser ?

Le modèle en spirale est largement utilisé dans l'industrie du logiciel car il est en phase avec le processus de développement naturel de tout produit, c'est-à-dire apprendre avec maturité ce qui implique un risque minimum pour le client ainsi que pour les sociétés de développement. Cette approche est utilisée dans les scénarios suivants :

- Quand le prototypage est exigé
- Quand il y a une contrainte budgétaire et que l'évaluation des risques est importante.
- Pour les projets à risque moyen et élevé.
- Engagement à long terme du projet en raison des changements potentiels aux priorités économiques, car les exigences changent avec le temps.
- Le client n'est pas sûr de ses exigences, ce qui est généralement le cas.
- Les exigences sont complexes et nécessitent une évaluation pour être plus clair.
- Nouvelle gamme de produits qui devrait être publiée en plusieurs phases pour obtenir suffisamment de commentaires des clients.
- Des changements importants sont attendus dans le produit au cours du cycle de développement.
- Quand le projet implique de la recherche et de l'investigation

ii. Avantages

L'avantage du modèle de cycle de vie en spirale est qu'il permet d'ajouter des éléments du produit, lorsqu'ils deviennent disponibles ou connus. Cela garantit qu'il n'y a aucun conflit avec les exigences et la conception précédentes. Un autre aspect positif de cette méthode est que le modèle en spirale force une participation précoce de l'utilisateur dans l'effort de développement du système. Ces avantages sont :

- Les exigences changeantes peuvent être accommodées.
- Permet une utilisation intensive des prototypes.
- Les exigences peuvent être capturées plus précisément.
- Les utilisateurs voient le système tôt (Feedback rapide du client)
- Une évaluation continue du procédé
- Le développement peut être divisé en parties plus petites et les parties risquées peuvent être développées plus tôt, ce qui contribue à une meilleure gestion des risques.

iii. inconvénients

Ce modèle exige une gestion très stricte pour compléter de tels produits et il y a un risque de faire tourner la spirale dans une boucle indéfinie. Ainsi, la discipline du changement et l'ampleur de prendre des demandes de changement est très important pour développer et déployer le produit avec succès. Nous notons également que:

- La gestion est plus complexe.
- La fin du projet peut ne pas être connue tôt.
- Ne convient pas aux projets de faible risque et pourrait être coûteux pour les petits projets.
- Le processus est complexe
- Un grand nombre d'étapes intermédiaires nécessite une documentation excessive.

d) Modèle Agile

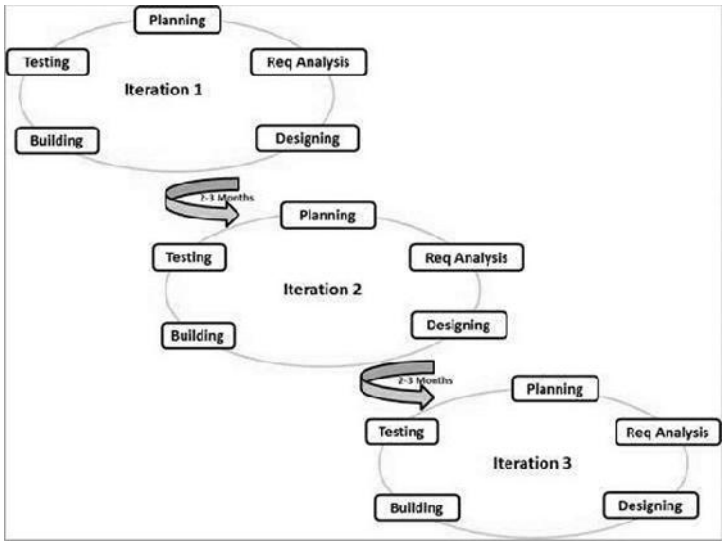
Au milieu des années 90, un groupe d'expert en Cycles de vie de logiciels voulaient proposer de nouveaux modèles plus « légers » (moins de documentation et moins de contrôle sur le procédé), dédiés à des projets de petite ou moyenne taille avec une équipe réduite et pouvant s'ajuster rapidement aux changements des spécifications tout en garantissant des livraisons fréquentes : Ces modèles sont qualifiés de « **modèles agiles** ».

Un modèle SDLC Agile est une combinaison de modèles de processus **itératif** et **Incrémental** mettant l'accent sur l'adaptabilité des processus et la satisfaction des clients grâce à la livraison rapide de logiciels fonctionnels. Les méthodes Agile décomposent le produit en petites constructions incrémentales. Ces versions sont fournies dans les itérations. Chaque itération dure généralement environ une à trois semaines. Chaque itération implique des équipes inter-disciplinaires travaillant simultanément sur différents domaines comme : *la planification, l'analyse des besoins, la conception, le codage, les tests unitaires et les tests d'acceptation.*

À la fin d'une itération, un produit fonctionnel est présenté au client et aux parties prenantes importantes.

i. Concept Agile

Le modèle Agile considère que chaque projet doit être géré différemment et que les méthodes existantes doivent être adaptées au mieux aux exigences du projet. Dans la philosophie agile, les tâches sont divisées en petites périodes afin de fournir des fonctionnalités spécifiques à une version du système. A cet effet, l'approche itérative est considérée et la construction d'un logiciel fonctionnel est fournie après chaque itération. Chaque construction est incrémentale en termes de fonctionnalités; la version finale contient toutes les fonctionnalités requises par le client. Une illustration graphique de l'Agile est :



Les méthodes Agile les plus populaires incluent le **RUP** (Rational Unified Process, 1994) **Scrum** (1995), Crystal Clear, Extreme Programming (XP) (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995), UP7, etc. Ces modèles sont maintenant collectivement appelés

méthodologies Agiles, après la publication du Manifeste Agile en 2001. Les principes du Manifeste Agile sont les suivants :

Individus et interactions au lieu de processus et outils - Dans le développement Agile, les collaborateurs sont la clé du succès, de même que les interactions comme la programmation par paire.

Logiciel fonctionnel au lieu de documentation massive – La démonstration d'un logiciel fonctionnel est considérée comme le meilleur moyen de communication avec les clients pour comprendre leurs exigences, au lieu d'une dépendance accrue aux documents qui sont souvent des « **mensonges** » formels. La production des documents aussi courts que possible est conseillée.

Collaboration du client au lieu de négociation de contrats - Comme les exigences ne peuvent pas être entièrement collectées au début du projet en raison de divers facteurs, une interaction continue avec le client est très importante pour obtenir les exigences du produit.

Réagir aux changements au lieu de suivre le plan - Le développement Agile est axé sur des réponses rapides au changement et au développement continu. Une meilleure stratégie est de faire de très courtes planifications (02 semaines à 01 mois).

ii. Comparaison entre les Modèles Agile et les modèles traditionnels

Les modèles agiles sont basées sur les méthodes de **développement logiciel adaptatif**, alors que les modèles traditionnels comme le modèle en cascade sont basés sur une approche **prédictive**. Les équipes dans les modèles traditionnels travaillent généralement avec une planification détaillée et disposent d'une prévision complète des tâches et fonctionnalités exactes à effectuer dans les prochains mois ou pendant le cycle de vie du produit.

Les méthodes prédictives dépendent entièrement de **l'analyse des besoins et de la planification effectuées en début de cycle**. Tout changement à intégrer passe par une gestion stricte du contrôle des changements et une hiérarchisation des priorités.

Les méthodes Agile utilisent une **approche adaptative** où il n'y a pas de planification détaillée et où il n'y a de clarté sur les tâches futures qu'en ce qui concerne les caractéristiques de ce qui doivent être développées. Il y a un développement basé sur les fonctionnalités et l'équipe s'adapte dynamiquement aux exigences changeantes du produit. Le produit est testé très fréquemment, à travers des itérations, minimisant ainsi le risque de défaillances majeures à l'avenir.

L'interaction avec le client est l'épine dorsale ou le cœur de la méthodologie Agile, et la communication ouverte avec une documentation minimale sont les caractéristiques typiques de l'environnement de développement Agile. Les équipes agiles travaillent en étroite collaboration les unes avec les autres et sont le plus souvent situées dans le même lieu géographique.

iii. Avantages et inconvénients de méthodes agiles

Les méthodes agiles sont largement acceptées dans le monde du logiciel actuellement. Cependant, cette méthode peut ne pas toujours convenir à tous les produits. Voici quelques avantages et inconvénients du modèle Agile.

- Les avantages du modèle Agile sont les suivants:
- Est une approche très réaliste au développement de logiciels.
- Favorise le travail d'équipe et la formation croisée.
- La fonctionnalité peut être développée rapidement et démontrée ou présentée au client.
- Les besoins en ressources sont minima.
- Convient pour les besoins fixes ou variables
- Fournit des versions partielles fonctionnelles du logiciel assez tôt.
- Bon modèle pour les environnements qui changent régulièrement.
- Règles minimales, documentation facile à utiliser.
- Peu ou pas de planification requise.
- Facile à gérer
- Donne de la flexibilité aux développeurs.

Les inconvénients du modèle Agile sont les suivants:

- Ne convient pas pour gérer les dépendances complexes.
- Plus de risque de durabilité, de maintenabilité et d'extensibilité.
- Un plan d'ensemble, un leader agile et une pratique agile du PM est un impératif sans lequel cela ne fonctionnera pas.
- La gestion stricte de la livraison dicte la portée, la fonctionnalité à livrer et les ajustements pour respecter les délais.
- Cela dépend fortement de l'interaction avec le client, donc si le client n'est pas clair, l'équipe peut être conduite dans la mauvaise direction.
- Il y a une dépendance individuelle très élevée, car il y a une documentation minimale à générer.
- Le transfert de technologie aux nouveaux membres de l'équipe peut être assez difficile en raison du manque de documentation.

e) Modèle RAD (Rapid Application Development)

Le modèle RAD (Rapid Application Development) est basé sur le prototypage et le développement itératif sans planification spécifique. Le processus d'écriture du logiciel lui-même implique la planification requise pour le développement du produit.

Le modèle RAD se concentre sur la collecte des besoins des clients à travers des ateliers ou des groupes de discussion, le test précoce des prototypes par le client en utilisant le concept itératif, la réutilisation des prototypes existants, l'intégration continue et la livraison rapide.

i. Concept RAD

Le développement rapide d'applications est une méthodologie de développement de logiciels qui utilise une planification minimale en faveur du prototypage rapide. Un prototype est un modèle fonctionnel qui est fonctionnellement équivalent à un composant du produit.

Dans le modèle RAD, les modules fonctionnels sont développés en parallèle en tant que prototypes et sont intégrés pour fabriquer le produit complet pour une livraison plus rapide du produit. Comme il n'y a pas de préplanification détaillée, il est plus facile d'intégrer les changements dans le processus de développement.

Les projets RAD suivent un modèle itératif et incrémental et ont de petites équipes comprenant des développeurs, des experts du domaine, des représentants des clients et d'autres ressources informatiques travaillant progressivement sur leur composant ou prototype.

L'aspect le plus important pour que ce modèle réussisse est de s'assurer que les prototypes développés sont réutilisables.

Modèle RAD Design

Le modèle RAD distingue les phases d'analyse, de conception, de construction et de test en une série de cycles de développement courts et itératifs.

Voici les différentes phases du modèle RAD -

Modélisation d'entreprise

Le business model du produit en développement est conçu en termes de flux d'informations et de diffusion d'informations entre les différents canaux métiers. Une analyse complète de l'entreprise est effectuée pour trouver les informations vitales pour les entreprises, comment elles peuvent être obtenues, comment et quand les informations sont traitées et quels sont les facteurs qui déterminent le flux d'informations.

Modélisation des données

Les informations collectées dans la phase Business Modeling sont examinées et analysées pour former des ensembles d'objets de données essentiels pour l'entreprise. Les attributs de tous les ensembles de données sont identifiés et définis. La relation entre ces objets de données est établie et définie en détail en fonction du modèle économique.

Modélisation de processus

Les ensembles d'objets de données définis dans la phase de modélisation des données sont convertis pour établir le flux d'informations métier nécessaire pour atteindre les objectifs métier spécifiques selon le modèle métier. Le modèle de processus pour toute modification ou amélioration des ensembles d'objets de données est défini dans cette phase. Des descriptions de processus pour ajouter, supprimer, extraire ou modifier un objet de données sont données.

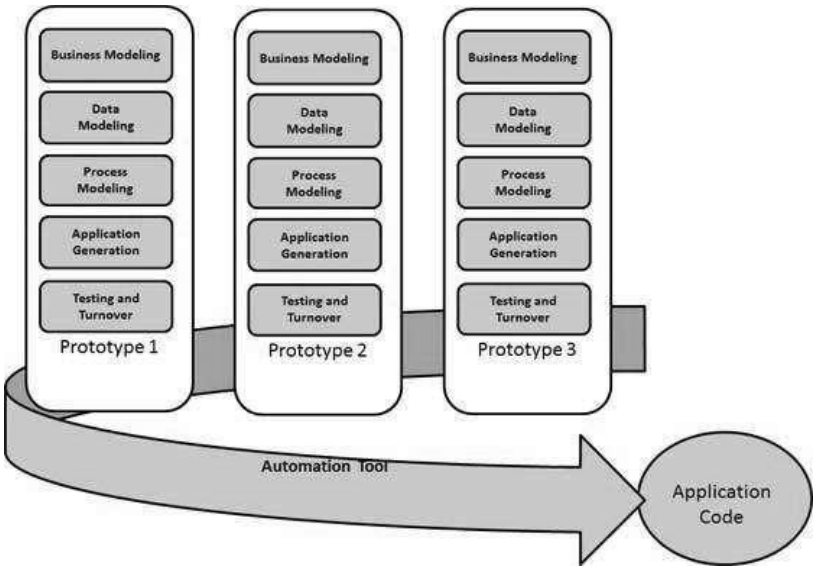
Génération d'application

Le système réel est construit et le codage est effectué en utilisant des outils d'automatisation pour convertir les modèles de processus et de données en prototypes réels.

Test et chiffre d'affaires

Le temps de test global est réduit dans le modèle RAD car les prototypes sont testés indépendamment à chaque itération. Cependant, le flux de données et les interfaces entre tous les composants doivent être soigneusement testés avec une couverture de test complète. Comme la plupart des composants de la programmation ont déjà été testés, cela réduit le risque de problèmes majeurs.

L'illustration suivante décrit le modèle RAD en détail.



ii. Comparaison entre les Modèles RAD et les modèles traditionnels

Le SDLC traditionnel suit des modèles de processus rigides mettant fortement l'accent sur l'analyse des besoins et la collecte avant le début du codage. Il fait pression sur le client pour qu'il signe les exigences avant que le projet ne commence et le client ne perçoive le produit qu'après un espace de temps assez long.

Le client peut avoir besoin de modifications après avoir vu le logiciel. Cependant, le processus de changement est assez rigide et il peut ne pas être possible d'incorporer des changements majeurs dans le produit dans le SDLC traditionnel.

Le modèle RAD se concentre sur la livraison itérative et incrémentale des modèles fonctionnels au client. Cela se traduit par une livraison rapide au client et une implication du client pendant le cycle de développement complet du produit, réduisant ainsi le risque de non-conformité aux exigences réelles de l'utilisateur.

iii. Quand l'utiliser?

Le modèle RAD peut être appliqué avec succès aux projets dans lesquels une modularisation claire est possible. Si le projet ne peut pas être divisé en modules, RAD peut échouer. Les points suivants décrivent les scénarios typiques où RAD peut être utilisé -

- RAD doit être utilisé uniquement lorsqu'un système peut être modularisé pour être livré de manière incrémentale.
- Il devrait être utilisé s'il y a une grande disponibilité de concepteurs pour la modélisation.
- Il ne doit être utilisé que si le budget autorise l'utilisation d'outils de génération de code automatisés.
- Le modèle RAD doit être choisi uniquement si des experts du domaine sont disponibles avec des connaissances métier pertinentes.

- Doit être utilisé lorsque les exigences changent au cours du projet et que les prototypes de travail doivent être présentés au client dans de petites itérations de 2-3 mois.

iv. Avantages et inconvénients

Le modèle RAD permet une livraison rapide car il réduit le temps de développement global en raison de la réutilisabilité des composants et du développement parallèle. RAD ne fonctionne bien que si des ingénieurs hautement qualifiés sont disponibles et que le client s'engage également à réaliser le prototype ciblé dans le délai imparti. S'il y a un manque d'engagement de part et d'autre, le modèle peut échouer.

Les avantages du modèle RAD sont les suivants:

- Les exigences changeantes peuvent être accommodées.
- Les progrès peuvent être mesurés.
- Le temps d'itération peut être court avec l'utilisation d'outils RAD puissants.
- Productivité avec moins de personnes en peu de temps.
- Temps de développement réduit.
- Augmente la réutilisabilité des composants.
- Des examens initiaux rapides ont lieu.
- Encourage les commentaires des clients.
- L'intégration dès le début résout beaucoup de problèmes d'intégration.

Les inconvénients du modèle RAD sont les suivants:

- Dépendance élevée envers les membres de l'équipe techniquement pour identifier les besoins de l'entreprise.
- Seul le système pouvant être modularisé peut être construit en utilisant RAD.
- Nécessite des développeurs / concepteurs hautement qualifiés.
- Grande dépendance aux compétences de modélisation.
- Inapplicable aux projets moins chers car le coût de la modélisation et de la génération de code automatisée est très élevé.
- La gestion est plus complexe.
- Convient aux systèmes basés sur des composants et évolutifs.
- Nécessite l'implication de l'utilisateur tout au long du cycle de vie.
- Convient pour un projet nécessitant des temps de développement plus courts.

f) Modèle de prototype de logiciel

Le prototypage logiciel fait référence à la construction de prototypes d'applications logicielles qui présente les fonctionnalités du produit en cours de développement, mais qui ne tiennent peut-être pas la logique exacte du logiciel original.

Le prototypage logiciel devient très populaire comme modèle de développement logiciel, car il permet de comprendre les besoins des clients à un stade précoce de développement. Il aide à obtenir des informations précieuses de la part du client et aide les concepteurs de logiciels et les développeurs à comprendre exactement ce que l'on attend du produit en cours de développement.

i. Qu'est-ce que le prototypage logiciel?

Le prototype est un modèle fonctionnel du logiciel avec des fonctionnalités limitées. Le prototype ne tient pas toujours la logique exacte utilisée dans l'application et représente un effort supplémentaire à prendre en compte lors de l'estimation de l'effort.

Le prototypage est utilisé pour permettre aux utilisateurs d'évaluer les propositions des développeurs et de les tester avant la mise en œuvre. Il aide également à comprendre les exigences qui sont spécifiques à l'utilisateur et qui n'ont pas été prises en compte par le développeur lors de la conception du produit.

Une approche par étapes expliquée pour concevoir un prototype de logiciel est la suivante :

Identification des exigences de base

Cette étape implique de comprendre les exigences de base des produits, notamment en termes d'interfaces utilisateur. Les détails les plus complexes de la conception interne et les aspects externes tels que la performance et la sécurité peuvent être ignorés à ce stade.

Développement du prototype initial

Le prototype initial est développé à ce stade, où les exigences très basiques sont présentées et les interfaces utilisateur sont fournies. Ces fonctionnalités peuvent ne pas fonctionner exactement de la même manière en interne dans le logiciel réellement développé. Bien que, les solutions de contournement sont utilisées pour donner le même aspect et la même sensation au client dans le prototype développé.

Examen du prototype

Le prototype développé est ensuite présenté au client et aux autres parties prenantes importantes du projet. Les commentaires sont collectés de manière organisée et utilisés pour d'autres améliorations du produit en cours de développement.

Révision et amélioration du prototype

Les `feedback` et les commentaires sont examinés au cours de cette étape et certaines négociations se déroulent avec le client sur la base de facteurs tels que les contraintes de temps et de budget et la faisabilité technique de la mise en œuvre effective. Les changements acceptés sont de nouveau incorporés dans le nouveau prototype développé et le cycle se répète jusqu'à ce que les attentes du client soient satisfaites.

Les prototypes peuvent avoir des dimensions horizontales ou verticales. Un prototype horizontal affiche l'interface utilisateur du produit et donne une vue plus large de l'ensemble du système, sans se concentrer sur les fonctions internes. Un prototype vertical est une élaboration détaillée d'une fonction spécifique ou d'un sous-système du produit.

Le but du prototype horizontal et vertical est différent. Les prototypes horizontaux sont utilisés pour obtenir plus d'informations sur le niveau de l'interface utilisateur et les exigences métier. Il peut même être présenté dans les démos de vente pour obtenir le marché. Les prototypes verticaux sont de nature technique et sont utilisés pour obtenir des détails sur le

fonctionnement exact des sous-systèmes. Par exemple, les exigences de base de données, les interactions et les charges de traitement de données dans un sous-système donné.

ii. Types de Prototypage

Il existe différents types de prototypes de logiciels utilisés dans l'industrie. Les principaux types largement utilisés sont :

Throwaway / Prototypage rapide (Rapid Prototyping) - Le prototypage par abandon est également appelé prototypage rapide ou fermé. Ce type de prototypage utilise très peu d'efforts avec l'analyse des exigences minimales pour construire un prototype. Une fois que les exigences réelles sont comprises, le prototype est mis au rebut et le système réel est développé avec une compréhension claire des besoins des utilisateurs.

Prototypage évolutif - Le prototypage évolutif, appelé aussi prototypage expérimental, est basé sur la construction de prototypes fonctionnels réels avec des fonctionnalités minimales au début. Le prototype développé constitue le cœur des futurs prototypes sur lesquels l'ensemble du système est construit. En utilisant le prototypage évolutif, les exigences bien comprises sont incluses dans le prototype et les exigences sont ajoutées au fur et à mesure qu'elles sont comprises.

Prototypage incrémental- Le prototypage incrémental fait référence à la construction de plusieurs prototypes fonctionnels des différents sous-systèmes, puis à l'intégration de tous les prototypes disponibles pour former un système complet.

Prototypage extrême - Le prototypage extrême est utilisé dans le domaine du développement Web. Il se compose de trois phases séquentielles. Tout d'abord, un prototype de base avec toutes les pages existantes est présenté au format HTML. Ensuite, le traitement des données est simulé en utilisant une couche prototype de services. Enfin, les services sont implémentés et intégrés au prototype final. Ce processus s'appelle Extreme Prototyping et sert à attirer l'attention sur la deuxième phase du processus, où une interface utilisateur entièrement fonctionnelle est développée avec très peu de considération pour les services réels.

iii. Quand utiliser le prototypage ?

Le prototypage logiciel est très utile dans le développement de systèmes ayant un niveau élevé d'interactions avec les utilisateurs, tels que les systèmes en ligne. Les systèmes qui demandent aux utilisateurs de remplir des formulaires ou de passer par divers écrans avant que les données ne soient traitées peuvent utiliser le prototypage de manière très efficace pour donner l'apparence exacte avant même que le logiciel proprement dit ne soit développé.

Les logiciels qui impliquent trop de traitement de données et dont la plupart des fonctionnalités sont internes avec très peu d'interface utilisateur ne bénéficient généralement pas de prototypage. Le développement de prototypes pourrait être une charge supplémentaire dans de tels projets et nécessiterait beaucoup d'efforts supplémentaires.

iv. Avantages et inconvénients du prototypage

Le prototypage logiciel est utilisé dans des cas typiques où la décision doit être prise très soigneusement afin que les efforts consacrés à la construction du prototype apportent une valeur considérable au logiciel final développé.

Les avantages du modèle de prototypage sont les suivants:

- Implication accrue des utilisateurs dans le produit avant même sa mise en œuvre.
- Le fait qu'un modèle fonctionnel du système soit présenté, les utilisateurs comprennent mieux le système en cours de développement.
- Réduit le temps et le coût car les défauts peuvent être détectés beaucoup plus tôt.
- Un retour d'information plus rapide est disponible, conduisant à de meilleures solutions.
- Les fonctionnalités manquantes peuvent être facilement identifiées.
- Des fonctions confuses ou difficiles peuvent être identifiées.

Les inconvénients du modèle de prototypage sont les suivants:

- Risque d'analyse insuffisante des besoins en raison d'une trop grande dépendance vis-à-vis du prototype.
- Les utilisateurs peuvent être confus dans les prototypes et les systèmes réels.
- Pratiquement, cette méthodologie peut augmenter la complexité du système, car la portée du système peut s'étendre au-delà des plans originaux.
- Les développeurs peuvent essayer de réutiliser les prototypes existants pour construire le système réel, même lorsque cela n'est pas techniquement possible.
- L'effort investi dans la construction de prototypes peut être trop important s'il n'est pas surveillé correctement.