Cours Génie Logiciel

Université Abderrahmane Mira de Béjaïa

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Dr. YESSAD Ep. OUATAH Nawal

Faculté des Sciences Exactes -Département Informatique

Table des matières

Objectifs

À l'issue de ce cours, l'apprenant sera capable de :

- Maîtriser les différentes méthodes de conception et de développement d'un logiciel.
- Mettre en relief la conduite d'un logiciel et les différentes métriques de qualité d'un logiciel.
- Comprendre les principes fondamentaux de l'approche orientée objet.
- Identifier les apports de la modélisation UML.
- S'initier aux techniques de modélisation orientées

Chapitre I : Généralités sur le génie logiciel

I

1. Introduction

Le développement de logiciel est une activité complexe, qui est loin de se réduire à la programmation, nécessitant des étapes, des méthodes et des techniques de développement. En effet, le génie logiciel est la branche de l'ingénierie associée au développement de logiciels utilisant des principes, méthodes et procédures scientifiques bien définis. Le résultat de l'ingénierie logicielle est un produit logiciel efficace et fiable.

2. Problématiques historiques du logiciel

Le terme « **crise du logiciel** » désigne la situation générale qui caractérise les échecs de développement logiciel durant les années soixante et soixante-dix. Dans son essence le terme fait référence à la difficulté d'écrire des programmes informatiques corrects, compréhensibles et vérifiables. Les causes de la crise du logiciel étaient nombreuses :

- Qualité du logiciel livré n'était pas à la hauteur des attentes de l'utilisateur ;
- Non-respect des délais prévus pour la livraison de produits logiciels ;
- Coûts de développement d'un logiciel étaient excessivement élevés ;
- Manque de procédé de tests et de vérification logicielle ;
- Maintenance du logiciel difficile et coûteuse ;
- Manque de Fiabilité et de performances (temps de réponse trop lents) ;
- Évolution rapide et croissante des logiciels, autrement dit, Il est rare de réutiliser un logiciel existant ou u de ses composants pour confectionner un nouveau logiciel, même si celui-ci comporte des fonctions similaires ;
- Besoins des utilisateurs qui changent sans cesse, la demande de fonctionnalités sophistiquées, l'inexpérience des développeurs, etc ;
- Manque de communication, ce qui veut dire qu'on s'apercevait souvent que le Logiciel développé ne correspondait pas à la demande.

La naissance de la nouvelle discipline « ***génie logiciel*** » est une réponse à tous ces problèmes et aux défis posés par la complexification des logiciels et de l'activité qui vise à les produire.

3. Naissance d'une nouvelle discipline

À la fin des années 1960 éclate la « crise du logiciel », prise de conscience des difficultés que rencontre le développement de logiciels. De ce constat va naître une nouvelle discipline, le génie logiciel. Le terme « *génie logiciel* », « *software engineering* » en anglais a été mentionné publiquement pour la première fois en 1968 par

Friedrich Bauer et Louis Bolliet pour une conférence organisée par **l'OTAN** (*Organisation du Taité de l'Atlantique Nord)* sur le sujet à Garmisch-Partenkirchen. Elle avait pour objectif de répondre à 2 constations : d'une part, le logiciel n'était pas fiable, et d'autre part, il était difficile à réaliser dans les délais et budgets prévus et en satisfaisant le cahier des charges.

Bauer donne la définition suivante du terme GL: «***establishement and use of sound engineering principales to obtain economically softaware that is reliable and works on real machines efficiently*** »

En 1993, la branche informatique (Computer Society) de l'IEEE (Institute of Electrical and Electronics Engineers) et l'ACM (Organisation professionnelle américaine des informaticiens) ont établi un comité conjoint dont la tâche était de définir un ensemble de critères et de normes appropriés pour la pratique professionnelle du Génie Logiciel (SWEBOK)

4. Définition du Génie logiciel

Le génie logiciel (*software engineering*) est un domaine d'ingénierie de la création, d'étude est la conception, la fabrication, et la maintenance des logiciels. Il consiste à identifier et à utiliser des méthodes, des pratiques et des outils permettant de maximiser les chances de réussite d'un projet logiciel. Dans ce qui suit nous présentons quelques définitions de la notion de « Génie Logiciel »

Définition

Le Génie Logiciel est l'application d'une approche systématique, disciplinée et quantifiable au développement, à l'exploitation et à la maintenance du logiciel. C'est-à-dire, l'application de l'ingénierie au logiciel (*la Norme IEEE 610.12*).

Définition

Bauer, un informaticien allemand, définit le génie logiciel comme l'établissement et l'utilisation de principes d'ingénierie afin d'obtenir des logiciels économiques, fiables et efficaces sur des machines réelles.

Définition

Classical and Object-Oriented Software Engineering with UML and Java de Schach , définit le Génie Logiciel comme une discipline qui a pour but la fabrication du logiciel sans faute, livré dans le délai et le budget prévus à l'avance, et qui satisfait aux besoins du client.

Définition

MC. Gaudel, défintі Le Génie Logiciel comme l'art de spécifier, de concevoir, de réaliser et de faire évoluer, avec des moyens et dans des délais raisonnables, des programmes, des documentations et des procédures de qualité en vue d'utiliser un système informatique pour résoudre certains problèmes.

Définition

L'arrêté du 30 décembre 1983, définit le génie logiciel « ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationaliser la production du logiciel et de son suivi »

Complément

Ou encore... : Le Génie Logiciel est l'art et de concevoir et de produire, avec économie et élégance, des applications, des objets, des Framework et d'autres systèmes informatiques, qui soient corrects, robustes, extensibles, réutilisables, sûrs, efficaces, faciles à maintenir et à utiliser.

5. Besoins et défis du génie logiciel

Les besoins du génie logiciel est dû au taux de changement plus élevé des besoins des utilisateurs et de l'environnement sur lequel le logiciel fonctionne. En effet la motivation première de la mise en œuvre du Génie Logiciel implique la prise en compte :

- Maîtrise et raccourcissement des temps de développement ;
- Maîtrise de la qualité (sûreté, robustesse) ;
- Réduction des coûts de développement du logiciel;
- Maintenance et évolution des logiciels spécifiques ;
- Gestion de l'entropie des logiciels qui ne cesse de croître ;
- Portabilité des logiciels sur toutes les plateformes (windows, linux, etc.) ;
- Adaptation de nouveaux environnements de développement (variété d'outils, de méthodes et des techniques de gestion de processus, des aspects humains au sein de l'équipe de développement des relations que celle-ci entretient avec les commanditaires et les utilisateurs du produit ;

6. Qu'est-ce qu'un logiciel ?

Définition

Un logiciel est un ensemble de séquences d'instructions interprétables par une machine et d'un jeu de données nécessaires à ces opérations. il détermine donc les tâches qui peuvent être effectuées par la machine, ordonne son fonctionnement et lui procure ainsi son utilité fonctionnelle. Un logiciel peut interagir avec des clients, qui peuvent être :des opérateurs humains, d'autres logiciels, des contrôleurs matériels etc. Un logiciel est également une réalisation d'une spécification, autrement dit, son comportement vérifie un ensemble de critères qui régissent ses interactions avec son environnement.

Définition

un logiciel est un ensemble d'entités nécessaires au fonctionnement d'un processus de traitement automatique de l'information. Parmi ces entités, on trouve par exemple : des programmes (en format code source ou exécutables, des documentations d'utilisation et configuration,etc.

6.1. Types de produits logiciels

- Les produits génériques : c'est des produits systèmes autonomes qui sont commercialisés et vendus à un client qui souhaite les acheter. (exemple : produits bureautiques)
- Les produits spécifiques : c'est des produits logiciels personnalisés commandés pas des clients spécifiques pour répondre à leurs propres besoins (exemple : logiciel du contrôle du trafic aérien).

7. Qualités d'un logiciel

Les qualités d'un logiciel se déterminent suivant un ensemble de principes et d'éléments de référence qui permettent de juger, d'estimer et de vérifier régulièrement sa qualité. Dans ce qui suit nous citons quelques caractéristiques essentielles pour un bon logiciel :

Caractéristiques	**Description**
Adéquation et validité	conformité aux spécifications définies par le cahier des charges
Utilisabilité (Facilité d'utilisation)	L'architecture du logiciel doit particulièrement être adaptée aux conditions de travail de l'utilisateur
Extensibilité	Possibilité d'ajout de nouvelles fonctionnalités
Réutilisabilité	Possibilité de réutilisation des parties entières et/ou partielles afin de développer d'autres logiciels similaires;
Efficacité (Performance)	Utilisation optimale des ressources matérielles, en terme d'espace mémoire, et temps d'exécution),
Portabilité	Transfert sous différents environnement matériels et Logiciels
Intégrité	Aptitude du logiciel à conserver sans altération son code et ses données contre des accès non autorisés ;
Documentation	Le logiciel est accompagné d'un manuel utilisateur, ou d'un tutoriel).
Testabilité et Vérifiabilité	Facilité de préparation des procédures de test c'est la possibilité de soumettre un logiciel à une épreuve de confirmation de la tâche à exécuter
Fiabilité (ou robustesse)	Le logiciel fonctionne raisonnablement en toutes circonstances, rien de catastrophique ne peut survenir, même en dehors des conditions d'utilisation prévues
Maintenabilité	Elle correspond au degré de facilité de la maintenance d'un produit logiciel

8. Principes fondamentaux de génie logiciel

Ghezzi propose un certain nombre de grands principes de génie logiciel:

- **La 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”) ;
- **La Généralisation** : regroupement d'un ensemble de fonctionnalités semblables en une fonctionnalité paramétrable (généricité et héritage) ;

- **La Structuration** : c'est la manière de décomposer un logiciel (utilisation d'une méthode bottom-up ou top-down). C'est la décomposition des problèmes en sous-problèmes indépendants.
- **La modularité** : Il s'agit de partitionner le logiciel en modules qui ont une cohérence interne ; et possèdent une interface ne divulguant sur le contenu du module que ce qui est strictement nécessaire aux modules clients ;
- **L'abstraction** : Il s'agit de présenter des concepts généraux regroupant un certain nombre de cas particuliers et de raisonner sur ces concepts généraux plutôt que sur chacun des cas particuliers. Le fait de fixer la bonne granularité de détails permet de raisonner plus efficacement et de factoriser le travail en instanciant le raisonnement général sur chaque cas particulier ;
- **La construction incrémentale**: Un développement logiciel a plus de chances d'aboutir s'il suit un cheminement incrémental ;
- **La Documentation** : correspond à la gestion des documents incluant leur identification, acquisition, production, stockage et distribution. Il est primordial de prévoir les évolutions possibles d'un logiciel pour que la maintenance soit la plus efficace possible. Pour cela, il faut s'assurer que les modifications à effectuer soient le plus locales possibles ;
- **La Vérification** : c'est la détermination du respect des spéciations établies sur la base des besoins identités dans la phase précédente du cycle de vie.

9. Éthiques de l'ingénierie du logiciels

Le Génie logiciel implique des responsabilités plus larges que la simple application des compétences techniques :

- Les ingénieurs logiciels doivent se comporter de façon responsable, honnête et éthique s'ils veulent être respecté en tant que professionnels

- Le comportement éthique est plus simplement faire respecter la loi, mais consiste à la suite d'une série de principes qui sont moralement corrects.

9.1. Les issues de responsabilité professionnelle

- *Confidentialité*: les ingénieurs devraient respecter la confidentialité de leurs employés ou clients, indépendamment d'un accord formel de confidentialité a été signé.
- *Compétence*: les ingénieurs ne devraient pas dénaturer leur niveau de compétence. Ils ne doivent pas accepter le travail qui est en dehors de leur compétence
- *Droits de propriétés intellectuelle*: les ingénieurs devraient etre au courant des lois locales régissant l'utilisation de la propriété intellectuelle tel que les brevets, droits d'auteurs,etc. ils devraient faire attention a ce que la propriètè intellectuelle des eployeurs et des clients soit protégé.
- *Mauvaise utilisation de l'ordinateur*: les ingénieurs ne doivent âs utiliser leurs compétences techniques pour abuser les ordinateurs d'autres personnes . mauvaise utilisation de l'ordinateur varie de relativement trivial (jeu en jouant sur la machine de l'employeur, par exemple) au au extrement graves (diffusion se virus)

Code d'éthique ACM/IEEE : les sociétés professionnelles publient des codes de conduite définissant les normes de comportement attendues de leurs membres.

Exemple

Exemple de code : ACM/IEEE code éthique (voir le livre de sommervilee 2015)

10. Processus de développement logiciel

10.1. Qu'est ce qu'un processus ?

Un processus est une série de tâches qui fournissent un ou plusieurs délivrables. il également définit qui fait quoi, quand et comment pour atteindre un objectif donné.

Un processus de développement logiciel est un ensemble (structuré) d'activités que conduisent à la production d'un logiciel.

10.2. Objectifs d'un processus

- Produire un logiciel de haute qualité en respectant des contraintes de délai, de coûts et de performance.
- Fournit les lignes directives pour un développement efficace d'un logiciel de qualité;
- Fournit une approche pour assigner des tâches et des responsabilités à l'intérieur d'une organisation.
- Réduit les risques et améliore les prévisions;
- Décrit les meilleurs méthodes de travail;
- Facilite la mise en œuvre ;
- Dicte au développeur comment implémenter en utilisant les outils standards de dévellepepment

10.3. Types de processus de développement

Il existe deux catégories d'approches pour les processus de développement des logiciels

10.3.1. Processus piloté par plan (planifié)

Il s'agit d'une catégorie de processus ou toutes les activités du processus sont planifiées à l'avance et les progrès sont mesurés sur ce plan. autrement dit, l'approche planifiée de l'ingénierie logicielle repose sur des étapes de développement distinctes avec des résultats qui devant être produits a chacune de ces étapes planifiées à l'avance et une itération se produit à l'intérieur des activités du processus.

10.3.2. Processus agile

il s'agit d'une catégorie de processus ou la planification est progressive et il est plus facile de modifier le processus afin de refléter l'évolution des besoins des clients. de plus, La spécification, la conception, la mise en œuvre et les tests sont interreliés et les résultats du processus de développement sont décidés par le biais d'un processus de négociation durant le processus de développement du logiciel.

11. Cycle de vie d'un logiciel

Le cycle de vie d'un logiciel (software lifecycle) désigne l'ensemble des étapes nécessaires au bon développement d'un logiciel sur le marché informatique. il décrit également, les enchaînements prévus des différentes phases dont le nom t le nombre d'étapes sont déterminés en fonction des besoins du projet, permettant son développement. Chaque phase ou étape de cycle de vie d'un logiciel produit en général des documents qui vont servir à une autre phase ultérieure. De façon générale, le cycle de vie d'un logiciel est la période de temps s'étalant du début à la fin de développement du logiciel. Un bon développement logiciel doit reposer sur une méthode et un cycle de développement bien précis. les objectifs d'un cycle de vie sont :

- Définir des jalons intermédiaires pour la validation du développement logiciel, autrement dit, vérifier la conformité du logiciel avec les besoins exprimés
- Vérifier le processus de développement en termes d'adéquation des méthodes mises en œuvre ;

- Détecter les erreurs au plus tôt afin de minimiser le coût et les délais de réalisation ainsi maîtriser la qualité du logiciel.

11.1. Étapes de cycle de vie d'un logiciel

Il existe de nombreux modèles de cycle de vie du développement d'un logiciel, les plus courants comportent principalement les phases suivantes :

- Étude du préalable (faisabilité) ;
- Définition et analyse de besoins (Spécification) ;
- Organisation du projet ;
- Conception du logiciel ;
- Implémentation ;
- Tests ;
- Livraison ;
- Maintenance.

11.1.1. Étude au préalable (étude de faisabilité)

Le développement est précédé d'une étude préalable, il s'agit d'un processus d'examen de l'existant, dont l'objectif est de faire des études de faisabilité, autrement dit, c'est de déterminer si le développement proposé vaut la peine d'être mis en œuvre sur le marché. Cette phase vise ainsi de dresser comment procéder afin de réaliser ce développement, et par quels moyens faut-il mettre en œuvre ?

Le tableau suivant résume la description, les entrés les sorties de l'étape"étude de faisabilité " :

Activité	Étude préalable
Description	Étudier la faisabilité du projet, ses contraintes techniques (coût, temps, qualité) et les alternatives possibles
Entrées	¨Problème à résoudre, objectifs à atteindre
Sorties	OUI (la décision est prise de réaliser le logiciel) ou NON (le projet est abandonné)

11.1.2. Expression et analyse des besoins (Spécification)

Cette phase est également appelée phase de spécification. Cette phase a comme objet la capture et l'analyse des besoins (exigences) de l'utilisateur pour le logiciel. Il s''agit de déterminer les fonctionnalités que doit posséder le logiciel et ce que le logiciel devra faire sous forme d'un document appelé « cahier des charges du logiciel ou spécification du logiciel ».

Le tableau suivant résume la description, les entrés les sorties de l'étape"Spécification" :

Activité	Spécifier
Description	Décrire ce que doit faire le logiciel (comportement en boite noire). Décrire comment vérifier en boite noire que le logiciel fait bien ce qui est exigé.
Entrées	Client qui a une idée de ce qu'il veut exigences, besoins ... concernant le système permettant de résoudre le problème

Sorties	Cahier des charges du logiciel (spécifications du logiciel). Des procédures de validation. Version provisoire des manuels d'utilisation et d'exploitation de logiciel

11.1.3. Organisation du projet

Cette phase vise à déterminer la manière dont développer le logiciel, elle inclut principalement :

- Analyse des coûts : établir une estimation du prix du projet ;
- Planification : établir un calendrier pour le développement ;
- Assurance qualité du logiciel : déterminer les actions qui permettront de s'assurer de la qualité du produit fini ;
- Répartition des taches : hiérarchiser les taches et les sous taches nécessaires au développement logiciel ;

Le tableau suivant résume la description, les entrés les sorties de l'étape "Organisation du projet" :

Activité	Planifier
Description	Organisation et planification des tâches relatives afin de mener à bien la conception et la réalisation du produit.
Entrées	Cahier des charges du logiciel (spécification du logiciel)
Sorties	Calendrier du projet , formation des équipes selon l'hiérarchisation des tâches, coût de réalisation

11.1.4. Conception du logiciel

Il s'agit d'une phase de définition du futur logiciel et la détermination de la façon dont le logiciel fournit les différentes fonctionnalités recherchées.

Le tableau suivant résume la description, les entrés les sorties de l'étape "Conception" :

Activité	Concevoir
Description	Organiser le logiciel afin qu'il puisse satisfaire les exigences de la spécification. Faire les principaux choix techniques pour satisfaire les exigences de la spécification.
Entrées	Une spécification
Sorties	Une description des décisions de conception. Des procédures de tests qui permettent de vérifier que les décisions de conception sont correctement implémentées en code source et qu'elles contribuent à satisfaire les exigences de la spécification.

La phase de conception est généralement décomposée en deux phases successives :

a) Conception architecturale

lors de cette phase, il faut décrire et détailler l'architecture de la solution ainsi que les différentes interconnexions. c'est-à-dire la décomposition et organisation de l'application en composants plus simples (modules), définis par leurs interfaces (fonctions et services offerts) et les interactions entre ces modules. Le résultat de cette démarche est « un document de conception ».

b) Conception détaillée

La conception détaillée affine la conception architecturale. Elle décrit la conception des modules de bas niveau et des objets. Autrement dit, Elle décompose les entités en entités plus élémentaires jusqu'au niveau où les entités sont faciles à implémenter et à tester. Ensuite, chaque composant logiciel est décrit en détail : son interface, les algorithmes utilisés, le traitement des erreurs, etc. L'ensemble de ces descriptions constitue le « document de conception détaillée ».

11.1.5. Implémentation

Après la phase de la conception détaillée, la phase d'implémentation s'enchaîne, cette dernière est également appelée phase de codage ou phase de réalisation. Cette phase vise à traduire la conception détaillée dans un langage de programmation cible.

Le tableau suivant résume la description, les entrés les sorties de l'étape "Implémentation" :

Activité	Coder
Description	Écrire le code source du logiciel. Tester le comportement du code source afin de vérifier qu'il réalise les responsabilités qui lui sont allouées.
Entrées	Spécification, conception.
Sorties	Code source. Tests unitaire. Documentation.

11.1.6. Tests

La phase d'implémentation est suivie de la phase de test. Cette phase vise à essayer le logiciel sur des données d'exemple pour s'assurer d'une part qu'il fonctionne correctement et d'une autre part qu'il satisfait la spécification élaborée lors de la phase d'analyse et possède bien le comportement attendu. Le tableau suivant résume la description, les entrés les sorties de l'étape "Tests" :

Activité	tester et valider
Description	Description Construire le logiciel complet exécutable. Dérouler les tests de validation sur le logiciel complet exécutable.
Entrées	Logiciel complet exécutable à valider. Tests de validation.
Sorties	Rapport de tests de validation.

les tests logiciels peuvent être présentés en différentes catégories:

a) Tests unitaires

Chaque composant est testé individuellement pour s'assurer qu'il est conforme à la conception détaillée. S'il s'avère qu'un composant comporte des erreurs, il est renvoyé à son développeur, qui devra diagnostiquer la cause de l'erreur puis corriger le composant. Le test unitaire de ce composant est alors à reprendre et les résultats de tests obtenus sont consignés dans le document des tests unitaires.

b) Tests d'intégration

l'intégration est de fait d'assembler progressivement plusieurs composants (ou modules) élémentaires en un composant de plus haut niveau. Un test d'intégration valide les résultats des interactions entre plusieurs composants permet de vérifier que leur assemblage s'est produit sans défaut et s'assurer de la bonne facture des composants assemblés.

c) Tests système

Test en vraie grandeur du système complet dans un environnement similaire à l'environnement de production

d) Tests de validation

Ces tests sont réalisés lors de la recette (validation) par un client d'un projet livré par l'un de ses fournisseurs. Souvent écrits par le client lui-même, ils portent sur l'ensemble du logiciel et permet de vérifier son comportement global en situation. De par leur spectre large et leur complexité, les tests de validation sont le plus souvent manuels. Les procédures à suivre sont regroupées dans un document associé au projet, fréquemment nommé plan de validation.

11.1.7. Livraison (déploiement)

L'objectif de cette phase est de fournir au client le logiciel qui fonctionne correctement. Cette phase inclut principalement :

- Installation : mise en fonctionnement le logiciel opérationnel sur le site du client
- Formation : enseigner aux utilisateurs à se servir du logiciel
- Assistance : accompagnement et réponse aux questions des utilisateurs

Le tableau suivant résume la description, les entrés les sorties de l'étape "Livraison" :

Activité	Livrer et déployer
Description	Fournir au client son produit demandé
Entrées	Logiciel complet.
Sorties	Utilisation du produit par le client et propositions d'éventuelles améliorations

11.1.8. Maintenance du logiciel

Une fois le logiciel est employé dans son environnement opérationnel, la phase de maintenance suit comme activité dans le but de surveiller le comportement et, si nécessaire. Il peut être nécessaire de modifier le logiciel pour corriger des défauts, pour améliorer ses performances ou autres caractéristiques, pour adapter le logiciel à un nouvel environnement, etc. Le tableau suivant résume la description, les entrés les sorties de l'étape "Maintenance" :

Activité	Corriger
Description	Correction des erreurs du système et demande d'évolution (modification de l'environnement technique, nouvelle fonctionnalité)
Entrées	Logiciel incomplet

Sorties	Logiciel corrigé ; Mises à jour ; Documents corrigés.

la maintenance du logiciel peut être de type :

a) - Maintenance corrective

c'est une maintenance qui Corrige les erreurs et les Défauts et Identifie également les défaillances en localisant la partie du code responsable. La maintenance corrective donne lieu à de nouvelles livraisons

b) Maintenance adaptative

c'est une maintenance qui ajuste le logiciel pour qu'il continue à remplir son rôle compte tenu du l'évolution des Environnements d'exécution, des Fonctions à satisfaire et des conditions d'utilisation

c) Maintenance perfective et d'extension

c'est une maintenance qui sert à accroître et améliorer les possibilités du logiciel afin de donner lieu à de nouvelles versions.

11.2. Documents courants

Document	**Description**
Cahier des charges	description initiale des fonctionnalités désirées, généralement écrite par l'utilisateur.
Calendrier du projet	décrit l'ordre des différentes taches ainsi que les délais et les ressources qu'elles demandent.
Plan de test du logiciel	décrit les procédures de test appliquées au logiciel pour contrôler son bon fonctionnement.
Conception du logiciel	décrit la structure du logiciel.
Plan d'assurance qualité du logiciel	décrit les activités mises en œuvre pour garantir la qualité du logiciel.
Manuel utilisateur	mode d'emploi pour le logiciel dans sa version finale.
Code source	code complet fournit du produit fini.
Rapport des tests	décrit les tests effectués et les réactions du système
Rapport des défauts	décrit le comportement du système qui n'ont pas satisfait le client ; il s'agit de plus souvent de défaillances du logiciel ou d'erreurs.

Exemple : Modèle de cahier de charges

- Positionnement et objectifs : dans quel contexte le cahier de charges est-il établi et quels sont ses objectif
- Spécifications applicatives ou fonctionnelles : descriptions des données, traitements, présentations, interfaces.
- Spécifications techniques : informations plus spécifiques sur le contexte.

- Spécifications administratives : budget, délais, propriété, clauses légales.
- Spécifications d'évaluation : les éléments qui permettront de valider l'offre faite par le futur contractant.
- Contraintes de réalisation.

12. Méthodes de développement classiques

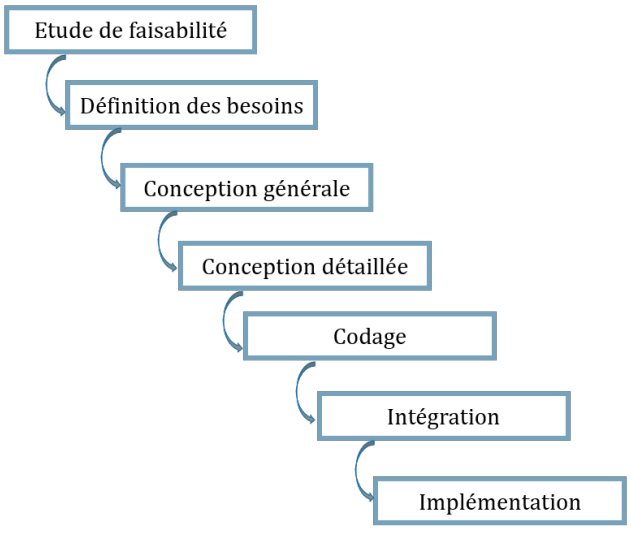
Un modèle de cycle de vie d'un logiciel est une représentation abstraite d'un processus, il présente une description d'un modèle d'une certaine perspective particulière. Un cycle de vie décrit généralement sur les activités , l'ordre des ces activités, les produits, qui sont les résultats d'une activité de modèle, Rôles qui reflètent les responsabilités des personnes impliquées dans le processus. il décrit également les Pré et post conditions qui sont des déclarations qui sont vraies avant et après une activité de modèle a été adoptée ou un produit fabriqué. Il existe différents types de cycles de développement entrant dans la réalisation d'un logiciel. Parmi les modèles de cycles de vie on peut citer :

12.1. Modèle séquentiel linéaire (cascade)

Le modèle séquentiel linéaire est également appelé « *modèle en cascade*», décrit en 1970 par Royce. Ce modèle est une succession de phases dont l'objectif de présenter rigoureusement le processus de développement et de définir de façon précise les rôles respectifs du fournisseur qui produit un livrable et du client qui accepte ou refuse le résultat.

Comme le montre la figure 1.2 ; le projet est découpé en phases successives dans le temps et à chaque phase correspond une activité principale bien précise produisant un certain nombre de livrables. Ce cycle de vie possède les caractéristiques suivantes :

- Sans aucune évaluation entre le début du projet et la validation ;
- Le passage d'une étape à un autre est favorable que si les résultats de l'étape précédente sont jugés satisfaisants ;
- L'activité d'une étape se réalise avec les résultats fournis par l'étape précédente ;
- Chaque étape sert de contrôle du travail effectué lors de l'étape précédente ;
- Chaque phase ne peut remettre en cause que la phase précédente ;



Modèle en cascade

Il existe de nombreuses versions de ce modèle, qui comprennent pour la plupart les mêmes étapes spécifiques du développement, mais qui les regroupent en phases différemment.

Le modèle en cascade présente comme avantages :

- Planification simple de ce qu'il faut faire ;
- Facile à comprendre et à mettre en place ;
- un procédé structuré pour une équipe inexpérimentée ;
- idéal pour la gestion et le suivi des projets.

Le modèle en cascade présente comme inconvénients :

- les besoins des client sont très rarement stables et clairement définis ;
- sensibilité aux nouveaux besoins: nécessite de refaire tout le procédé ;.
- Les erreurs de spécifications sont généralement détectées au moment des tests, voire au moment de la livraison du logiciel à l'utilisateur.
- les résultats intermédiaires obtenus ne reflètent pas la façon dont le code est réellement développé.
- Trop rigide, manque de flexibilité pour imprévus et très faible implication du client ;
- Aucune validation intermédiaire et pas de «feedback» avant la livraison au client, ce qui augmente les risques,

Le modèle en cascade est utilisé quand:

- les besoins sont connus et stables ;
- la technologie à utiliser est maîtrisée ;
- lors de la création d'une nouvelle version d'un produit existant ;
- lors du portage d'un produit sur une autre plate forme .

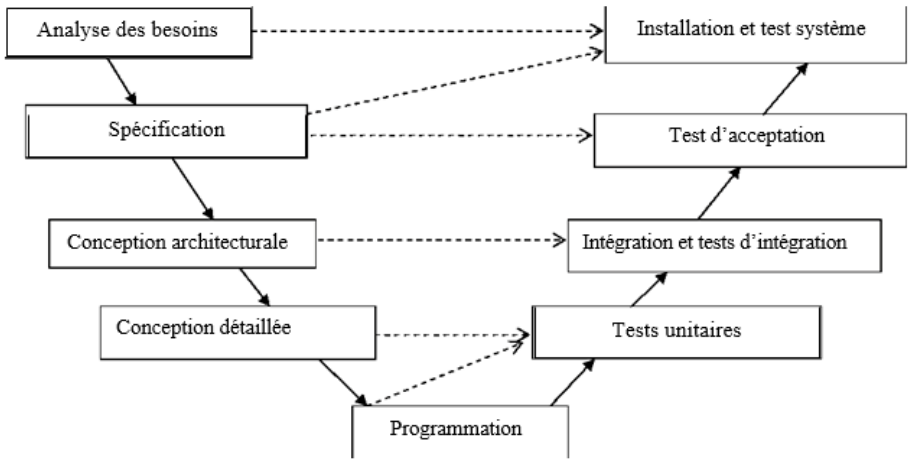
12.2. Modèle en V

Le modèle en V du cycle de vie du logiciel est dérivé du modèle de la cascade montrant non seulement l'enchaînement des phases successives (enchaînement séquentiel), mais aussi les relations logiques entre phases plus éloignées. Comme l'illustre la figure 1.3, la première branche correspond à un modèle en cascade classique quant à la seconde branche intègre des phases de tests effectués sur des composants réalisés:

les tests système sont préparés à partir de la spécification ;

les tests d'intégrations sont préparés à partir de la conception architecturale ;

les tests unitaires sont préparés à partir de la conception détaillée des composants).



Les flèches discontinues représentent le fait qu'une partie des résultats de l'étape de départ est utilisée directement par l'étape d'arrivée, Avec les jeux de tests préparés dans la première branche. Le cycle en V est le cycle qui a été normalisé, il est largement utilisé, notamment en informatique industrielle et en télécommunication. Ce modèle fait également apparaitre les documents qui sont produits à chaque étape qui permettent de valider les différents produits.

Complément

Le modèle en V présente comme avantages :

- Facile à planifier dans une gestion de projets ;
- Validations intermédiaires ;
- Bon suivi de projet : points de mesure concrets de l'avancement du travail avec étapes clés
- Limitations des risques en cascade par validation de chaque étape ;
- Existence d'outils supports.

Le modèle en V présente comme inconvénients :

- Ne gère pas les activités parallèles
- Souffre toujours du problème de la vérification trop tardive du bon fonctionnement du système ;
- Ne gère pas explicitement les changement de spécifications ;
- Prédominance de la documentation sur l'intégration validation tardive du système global ;
- Les validations intermédiaires n'empêchent pas la transmission des insuffisances.

Le modèle en V est utilisé quand:

- le produit à développer à de très hautes exigences de qualité ;
- les besoins sont connus à l'avance ;
- les technologies à utiliser sont connues à l'avance.

12.3. Modèle par prototypage

Le prototypage permet de contourner la difficulté de la validation liée à l'imprécision des besoins et caractéristiques du système à développer. Autrement dit, lorsqu'il est difficile d'établir une spécification détaillée, on a recours au prototypage qui est considéré, dans ce cas, comme un modèle de développement de logiciels.

Ce modèle consiste à fournir le plus rapidement possible une version d'essai du logiciel, Cela veut dire d'écrire une première spécification et de réaliser un sous-ensemble du produit logiciel final. Le modèle par prototypage sert également à montrer aux clients les fonctionnements que l'on se propose de mettre en œuvre. Après que le client a donné son accord, le développement du logiciel suit généralement les phases du modèle en cascade.On en distinguera deux types de prototypage :

1. Prototype jetable : Maquette exploratoire développée pour mieux comprendre les besoins du client, évaluer différentes solutions, etc. Développé rapidement et jeté ensuite.
2. Prototype évolutif (ou réutilisable): Maquette destinée à être complétée/optimisée dans les prochaines étapes du développement jusqu'à l'obtention du produit final.

Complément

Le modèle par prototypage présente comme avantages :

Implication active du client : améliore la compréhension mutuelle du problème entre client, développeur et utilisateur ;

S'adapte rapidement aux changement des besoins ;

Validation concrète et non sur document ;

Limitation du risque à chaque itération ;

Le modèle par prototypage présente comme inconvénients :

Tentation de se contenter d'un prototype incomplet.

- implique un code faiblement structuré ;
- Degré très faible de maintenabilité
- Processus peut ne jamais s'arrêter
- très difficile d'établir un planning

le modèle par prototypage est utilisé quand :

- Pour de très petits projets impliquant très peu de personnes ;
- Quand les besoins sont instables et/ ou nécessitent des clarifications ;
- Peut être utilisé avec le modèle en cascade pour la clarification des besoins ;
- Quand des livraison rapides sont exigées.

12.4. Modèle par incréments

Le modèle par incréments est un modèle itératif, Son principe est de concevoir et de livrer au client un sous-ensemble minimal du système global qui soit fonctionnel. Le processus se répète durant l'ensemble de cycle de vie par l'ajout d'incréments minimaux (nouvelles fonctionnalités) jusqu'à l'obtention d'un produit logiciel final.

Dans un premier temps un logiciel noyau est développés, puis successivement, les incréments sont développés et intégrés. Chaque incrément est développé selon l'un des modèles précédents. Dans ce modèle, les intégrations sont progressives et il peut y avoir des livraisons et des mises en service après chaque intégration d'incrément.

Complément

Le modèle par incréments présente comme avantages :

- Développement de fonctionnalités à risque en premier;
- Chaque incrément donne un produit fonctionnel ;
- Le client intervient à la fin de chaque incrément ;
- intégrations sont progressives ;
- Meilleur lissage du temps et de l'effort de développement à cause de la possibilité de recouvrement des différentes phases.

Le modèle par incréments présente comme inconvénients :

- Exige une bonne planification et une bonne conception ;
- Exige une vision sur le produit fini pour pouvoir le diviser en incréments ;
- Remise en cause du noyau de départ, des incréments précédents;
- L'impossibilité d'intégrer un nouvel incrément

Le modèle par incréments est utilisé quand:

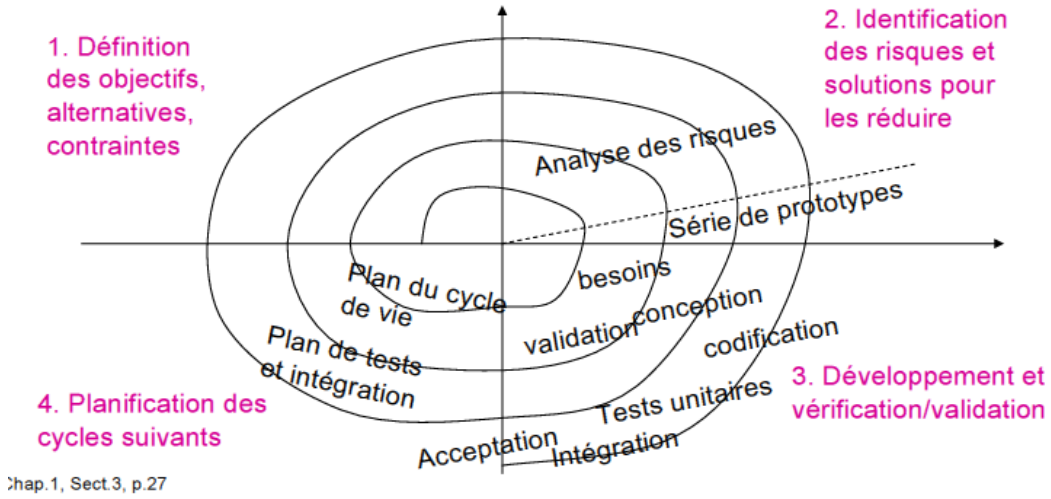
- Quand la plupart des spécifications sont connues à l'avance et vont être sujettes à de faibles évolutions;

- Des projets impliquant de nouvelles technologies ;
- Besoin rapide d'un produit fonctionnel pour des projets de longues durées.

12.5. Modèle en spirale

B.boehm a introduit un modèle cyclique appelé « modèle en spirale ». Il est représenté par une spirale commençant au centre et repassant continuellement par les taches de base d'un cycle. chaque cycle de la spirale comporte les phases suivantes:

- La spécification des besoins,
- La planification,
- L'analyse des risques,
- La conception,
- Le développement d'un prototype
- Simulation et essais du prototype ;
- Détermination des besoins à partir des résultats des essais ;
- Validation des besoins par un comité de pilotage ;
- Planification de l'itération suivante.



Dans le modèle de la spirale :

- chaque cycle donne lieu à une contractualisation préalable, s'appuyant sur les besoins exprimés lors du cycle précédent et Le dernier cycle permet de développer la version finale et d'implémenter le logiciel.
- Le nombre de cycles est variable selon que le développement est classique ou incrémental.
- Le modèle en spirale met l'accent sur l'analyse et traitement des risques au travers de bouclages successifs
- Chaque cycle/spire confirme et affine les spires précédents en menant des activités de même nature successivement
- l'analyse ou la conception ne sont plus effectuées dans une seule phase mais sont conduites en tant qu'activités qui se déroulent sur de multiples phases
- A chaque étape, après avoir défini les objectifs et les alternatives, celles-ci sont évaluées par différentes techniques, l'étape est réalisée et la suite est planifiée.
- Ce modèle est utilisé pour des projets dont les enjeux sont importants.
- le modèle en spirale englobe tous les autres modèles ; il n'est pas nécessaire d'adopter un seul modèle à chaque cycle de la spirale ou même pour l'ensemble d'un système.

le modèle en spirale est un modèle itératif, ou la planification de la version se fait selon une analyse de risques. l'idée est de s'attaquer aux risques les plus importants assez tôt. de façon générale, les risques liés au développement logiciels peuvent être répartis en quatre catégories :

- les risques commerciaux (placement du produit sur le marché, concurrence) ;
- les risques financiers (capacités financières suffisantes pour réaliser le produit) ;
- les risques techniques (la technologie employée est-elle éprouvée) ;
- les risques de développement (l'équipe est-elle suffisamment expérimentée ?).

Complément

Le modèle en spirale présente comme avantages :

- Identification rapide des risques ;
- Planification très attentive et rigoureuse ;
- Fonctions critiques développées en premier ;
- Feed-back rapide du client ;
  Une évaluation continue du procédé ;
- Permet d'établir le modèle de développement le plus adéquat en regard du risque.
- Méta-modèle: tous les autres modèles de développement constituent une variante de la spirale!

Le modèle en spirale présente comme inconvénients :

- l'évaluation des risques peut prendre beaucoup de temps ;
- la spirale peut s'éterniser ;
- les développeurs doivent être réaffectés pendant les phases de non-développement ;
- les objectifs ne pas souvent faciles à formuler.

Le modèle en spirale est utilisé quand :

- Le prototypage est exigé ;
- Le risque du projet est considérable ;
- Les spécifications ne sont pas stables ;

13. Méthodes de développement agiles

le développement axé sur le plan est essentiel pour certain types de systèmes néanmoins, ne répond pas à la totalité de besoins commerciaux. Des méthodes de développement dites méthodes agiles ont émergé à la fin des années 1990, qui sont des groupes de pratiques de pilotage et de réalisation de projets. L'origine des méthodes agiles est liée à l'instabilité de l'environnement technologique et au fait que le client est souvent dans l'incapacité de définir ses besoins de manière exhaustive dès le début du projet.

les méthodes agiles ont pour origine le manifeste Agile, rédigé en 2001 (http://www.agilemanifesto.org/) . le Manifeste agile : Réunion organisée par Kent Beck en février 2001 : *« nous devrons les meilleures façons pour développer un logiciel en le faisant et en aidant les autres à le faire. A travers ce travail, nous en sommes venus à valoriser »* :

1. **Personnes et interaction** plutôt que processus et outils ;
2. **Logiciel fonctionnel** plutôt que documentation complète ;
3. **Collaboration avec client** plutôt que négociation de contrat ;
4. **Réagir au changement** plutôt que suivre un plan.

Le terme « **agile** » fait ainsi référence à la capacité d'adaptation aux changements de contexte et aux modifications de spécifications intervenant pendant le processus de développement. Grâce aux méthodes du développement agile, le client est pilote à part entière de son projet et obtient très vite une première mise en production de son logiciel.

Les méthodes de développement agile vise à :

- Réduire le cycle de vie du logiciel (réduire radicalement le délai de livraison) en développant une version minimale, puis en intégrant les fonctionnalités par un processus itératif basé sur une écoute client et des tests tout au long du cycle de développement ;

- Impliquer au maximum le demandeur (client) ;

- Permettre une grande réactivité aux demandes des clients ;

-Se reposer sur un cycle de développement itératif, incrémental et adaptatif.

13.1. Fondements et principes des méthodes agiles

Les méthodes agiles prônent **quatre valeurs fondamentales citées précédemment** et et douze principes généraux :

1. Satisfaire le client en priorité ;
2. Accueillir favorablement les demandes de changement ;
3. Livrer fréquemment de nouvelles versions opérationnelles pour évaluation ;
4. Assurer une coopération permanente entre le client et l'équipe projet ;
5. Construire des projets autour d'individus motivés ;
6. Privilégier la conversation en face à face ;
7. Mesurer l'avancement du projet en termes de fonctionnalités de l'application ;
8. Faire avancer le projet à un rythme soutenable et constant ;
9. Porter une attention continue à l'excellence technique et à la conception ;
10. Faire simple ;
11. Responsabiliser les équipes ;
12. Ajuster à intervalles réguliers son comportement et ses processus pour être plus efficace ;

13.2. Panorama des méthodes agiles

Les méthodes pouvant être qualifiées d'agiles, depuis la publication du manifeste Agile, sont :

- *Rapid application development RAD* (1991) ;
- *Extreme programming XP* (1999) ;
- *Dynamic software developpement method DSDM* () ;
- *Adaptive software development ASD* (2000) ;
- *Crystal clear* (2004) ;
- *SCRUM* (1995), publié ensuite en (2010) ;
- *Feature driven development FDD* (2003) ;
- *Behavior-driven development BDD* (2003).

13.3. eXtreme Programming

L'extreme programming (XP) est une méthode agile inventée par Kent Beck, en octobre 1999. Elle est plus particulièrement orientée sur l'aspect réalisation d'une application, sans pour autant négliger l'aspect gestion de projet. XP définit un certain nombre de bonnes pratiques permettant de développer un logiciel dans des conditions optimales en plaçant le client au cœur du processus de développement, en relation étroite avec le client. De plus elle est adapté aux équipes réduites avec des besoins changeants.

Dans le livre ***Extreme Programming Explained***, la méthode est définie comme :

- une tentative de réconcilier l'humain avec la productivité ;
- un mécanisme pour faciliter le changement social ;
- une voie d'amélioration ;
- un style de développement ;
- une discipline de développement d'applications informatiques.

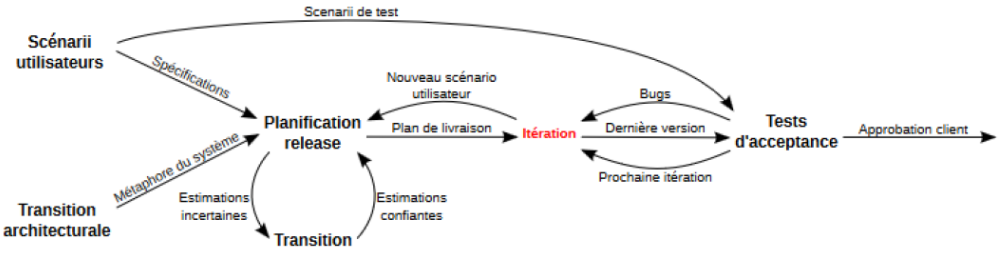
Son but principal est de réduire les coûts du changement. XP s'attache à rendre le projet plus flexible et ouvert au changement en introduisant des valeurs de base, des principes et des pratiques. L'originalité de la méthode est de les pousser à l'extrême puisque:

- la revue de code est une bonne pratique, elle sera faite en permanence (par un binôme)
- les tests sont utiles, ils seront faits systématiquement avant chaque mise en oeuvre
- la conception est importante, elle sera faite tout au long du projet (refactoring)
- la simplicité permet d'avancer plus vite, nous choisirons toujours la solution la plus simple
- la compréhension est importante, nous définirons et ferons évoluer ensemble des métaphores
- l'intégration des modifications est cruciale, nous l'effectuerons plusieurs fois par jour
- les besoins évoluent vite, nous ferons des cycles de développement très rapides pour nous adapter au changement

13.3.1. Cycle de développement

L'extreme programming repose sur des cycles rapides de développement (des itérations de quelques semaines) dont les étapes sont les suivantes :

- Une phase d'exploration détermine les scénarios « client » qui seront fournis pendant cette itération ;
- l'équipe transforme les scénarios en tâches à réaliser et en tests fonctionnels ;
- Chaque développeur s'attribue des tâches et les réalise avec un binôme ;
- Lorsque tous les tests fonctionnels passent, le produit est livré.



Le cycle se répète tant que le client peut fournir des scénarios à livrer. Généralement le cycle de la première livraison se caractérise par sa durée et le volume important de fonctionnalités embarquées. Après la première mise en production, les itérations peuvent devenir plus courtes (une semaine par

exemple).

13.4. Scrum

Scrum (www.scrum.org) est un schéma d'organisation de développement de produits complexes. Il est défini par ses créateurs comme un « *« cadre de travail holistique itératif qui se concentre sur les buts communs en livrant de manière productive et créative des produits de la plus grande valeur possible »* ».

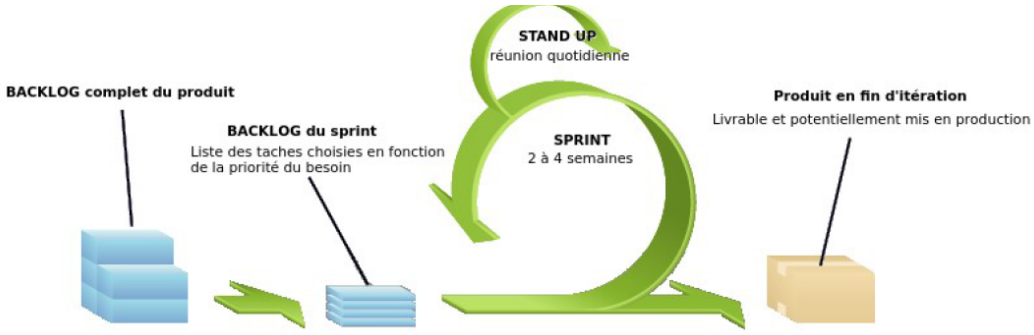
- Le framework s'appuie sur le découpage d'un projet en boîtes de temps, nommées « sprints ».
- Les sprints peuvent durer entre quelques heures et un mois (avec une préférence pour deux semaines).
- Chaque sprint commence par une estimation suivie d'une planification opérationnelle.
- Le sprint se termine par une démonstration de ce qui a été achevé.
- Avant de démarrer un nouveau sprint, l'équipe réalise une rétrospective. Cette technique analyse le déroulement du sprint achevé, afin d'améliorer
- ses pratiques.
- Le flot de travail de l'équipe de développement est facilité par son auto-organisation,
- il n'y aura donc pas de gestionnaire de projet.

La méthode scrum est fondée sur la conviction que le développement logiciel est une activité par nature non-déterministe et que l'ensemble des activités de réalisation d'un projet complexe ne peut être anticipé et planifié. C'est en cela que le scrum s'oppose aux démarches prédictives. Pour répondre à cette imprédictibilité, la méthodologie scrum propose un modèle de contrôle de processus fondée sur l'empirisme, via l'adaptation continue aux conditions réelles de l'activité et une réaction rapide aux changements.

Avant le démarrage du sprint 1, les objectifs sont définis lors d'un sprint 0. La mêlée (scrum meeting) a lieu quotidiennement et des réunions spécifiques permettent de lever les obstacles bloquants. Le Sprint a une durée variable (idéalement deux semaines). Après chaque sprint, une démonstration suivie d'une rétrospective ont lieu. Le propriétaire du produit peut à tout moment compléter ou modifier la liste des fonctionnalités à produire pour les prochains sprints. Sans modifier le but du sprint en cours, celuici peut être affiné et faire l'objet d'une renégociation entre le propriétaire du produit et l'équipe de

développement à la suite de nouvelles connaissances. Si le but du sprint devient obsolète, le propriétaire du produit a la capacité d'annuler un sprint en cours.

Chaque sprint constitue donc un incrément, facilitant le pilotage du projet. La notion d'itération couvre l'adaptabilité au quotidien. Cette adaptabilité est limitée par le but immuable d'un sprint.



une itération de la méthode scrum

14. Conclusion

Le génie logiciel est la science des bonnes pratiques de développement de logiciel. Cette science étudie en particulier la répartition des phases dans le temps, les bonnes pratiques concernant les documents clés que sont le cahier des charges, le diagramme d'architecture ou le diagramme de classes. Le but recherché est d'obtenir des logiciels de grande ampleur qui soient fiables, de qualité, et correspondent aux attentes de l'usager. Par contre, Le développement de logiciel consiste à étudier, concevoir, construire, transformer, mettre au point, maintenir et améliorer des logiciels. Le développement d'une application exige de :

- Procéder par étapes, en prenant connaissance des besoins de l'utilisateur; effectuer l'analyse; trouver une solution informatique; réaliser; tester; installer et assurer le suivi ;
- Procéder avec méthode, en partant du général aux détails et aux techniques; fournir une documentation; s'aider de méthodes appropriées; et Savoir se remettre en question ;
- Choisir une bonne équipe, en trouvant les compétences adéquates ; définir les missions de chacun et coordonner les différentes actions pour la construction du logiciel ;
- Contrôler les coûts et délais, en considérant l'aspect économique; maîtriser de la conduite du projet; et investissements aux bons moments ;
- Garantir le succès du logiciel, en répondant à la demande et assurer la qualité du logiciel.

15. Bibliographie

1. L. Audibert, Cours UML 2.0, site http://www.developpez.com.
2. J. Gabay, et D.Gabay, UML 2 Analyse et conception, Mise en oeuvre guidée avec études de cas, Dunod, 2008.
3. D. Gustafson, Génie Logiciel, Dunod, Paris, 2003.
4. L. BOUDAA, Cours de Génie logiciel, support de cours, Université IBN KHALDOUN-Tiaret, https, 2020.
5. R. Yende. Support de cours de génie logiciel. Licence. RDC (BÉNI), CongoKinshasa. ffcel-01988734, 2019.