Introduction aux méthodes de développement (Rational Unifed Process et Méthodes Agile

1. Introduction

Les méthodes d'analyse orientées objet sont initialement issues des milieux industriels. La préoccupation dominante de leurs auteurs est le génie logiciel, c'est-à dire les principes et les techniques permettant d'augmenter la rigueur et la qualité quand on construit une application informatique.
Initialement, UML(Unified Modeling Language) est le résultat de la fusion de trois méthodes orientées objet(La méthode OOD, object oriented design, de **grady booch,** La méthode OMT object modeling technique de **James Rumbaugh,** La méthode OOSE, object oriented software engineering d'**IVAR Jacobson**.
Il existe plusieurs méthodes de développement logiciel construites sur UML comme la méthode : UP, RUP, UP agile, XP, …..
Parmi ses méthodes notre choix est basé sur la méthode RUP (Rational Unified Process) et XP.

2. Rational Unifed Process (RUP)

2.1. Définition

RUP est une démarche de développement qui est souvent utilisé conjointement au langage UML

- RUP est Piloté par les cas d'utilisation
- Centré sur l'architecture Itératif et incrémental
- Chaque itération prend en compte un certain nombre de cas d'utilisation
- Les risques majeurs sont traités en priorité
- Chaque itération donne lieu à un incrément et produit une nouvelle version exécutable
- Adéquation du service rendu par le logiciel avec les besoins des utilisateurs
- Le développement d'un logiciel doit être centré sur l'utilisateur
- Les cas d'utilisation permettent d'exprimer ces besoins
- Détection et description des besoins fonctionnels
- Modélisation de différentes perspectives indépendantes et complémentaires
- Organisation des besoins fonctionnels RUP est centrée sur l'architecture

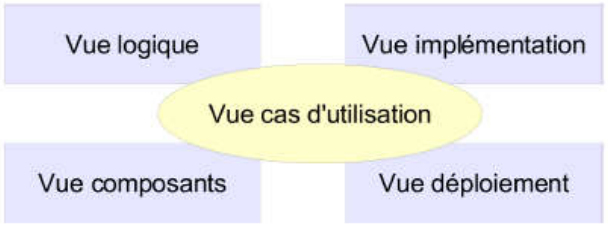

Figure 1 différente vue du système.

Vues du système

- **Vue cas d'utilisation**

Description du système comme un ensemble de transactions du point de vue de l'utilisateur

- **Vue logique**

✓ Créée lors de la phase d'élaboration et raffinée lors de la phase de construction

✓ Utilisation de diagrammes de classes, de séquences...

- **Vue composants**
  Description de l'architecture logicielle
- **Vue déploiement**
  Description de l'architecture matérielle du système
- **Vue implémentation**
  Description des algorithmes, code source

2.2. **Phases du cycle de développement:**

a) Initialisation : Définition du problème

b) Analyse : Planification des activités, Affectation des ressources, Cahier des Charges, Analyse

c) Développement : Développer le logiciel au travers d'une série d'itérations incrémentales (Conception Générale, Conception détaillée, Implantation, Tests)

d) Recettage et déploiement

a) **Phase Initialisation**

- Définition de l'étendue du projet
- Poser le problème : Quel est le système dont l'utilisateur a besoin ?
- Identifier les acteurs : utilisateurs, gestionnaires du système, plates-formes, interfaces avec autres applications
- Identifier les cas d'utilisations Préciser les cas d'utilisation les plus importants, en fonction du risque, sous forme de scénarios (itération sur les scénarios)

b) **Phase Analyse**

- Analyse du domaine : Modèle Objet statique Scénarios (Diagramme de séquences) Définition de l'Architecture. L'architecture est la façon d'organiser les objets pour qu'ils réalisent les fonctionnalités de l'application par leurs collaborations.

b 1) Planification du Développement

1. Identifier les risques principaux
2. Sélectionner un petit nombre de scénarios couvrant ces risques. Les scénarios sont classés par ordre de priorité en fonction des risques, de leur importance pour le client et du point de vue fonctionnel.
3. Sélectionner les classes et leurs relations à implanter en analysant les scénarios
4. implanter les classes et relations sélectionnées
5. Définir le plan de test
6. Tests : vérifier que les fonctionnalités des scénarios sont correctement réalisées
7. Tests d'intégration avec le résultat des itérations précédentes
8. Bilan de l'itération et définition des révisions

c) **Phase Développement**

1. Identifier les classes et relations à implanter
2. Compléter les classes et relations sélectionnées: Typage des attributs Méthodes et leurs signatures Ajout de classes d'implantation (conteneurs, contrôleurs) Choix de conception pour les relations (navigation, ...)
3. Choix de conception concernant l'héritage
4. Codage
5. Création / Mise à jour de la documentation

6. Tests des créés/modifiés par l'itération
7. Tests d'intégration des éléments créés/modifiés par l'itération

2.3. Organisation en activités de développement

Chaque phase comprend plusieurs itérations

Pour chacune des itérations, on se livre à plusieurs activités

- ✓ Modélisation métier
- ✓ Expression des besoins
- ✓ Analyse
- ✓ Conception
- ✓ Implémentation
- ✓ Test
- ✓ Déploiement

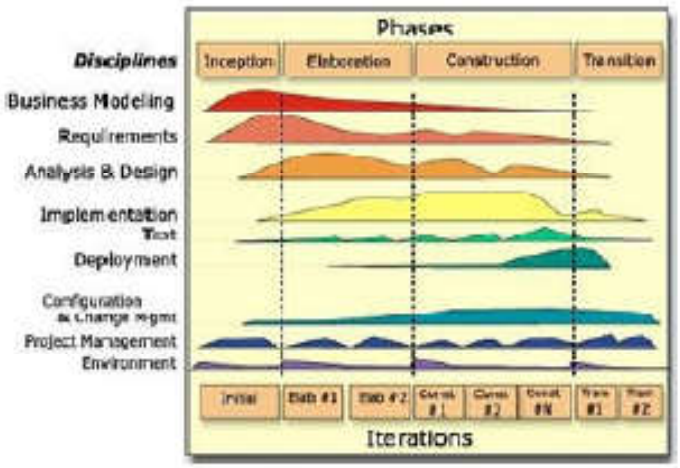

Figure 2 : Organisation en activités de développement

Les activités sont des étapes dans le développement d'un logiciel, mais à un niveau de granularité beaucoup plus fin que les phases. Chaque activité est répétée autant de fois qu'il y a d'itérations.

Remarque :

Dans la démarche RUP :

- ✓ Les objets (et donc leur classes), comme les méthodes, sont découverts en documentant les scénarios
- ✓ C'est une démarche guidée par les scénarios
- ✓ Cette démarche est préférable lorsque l'étude des traitements doit guider l'analyse Dans les autres cas, la production du modèle de classes doit être faite avant l'étude des scenarios

3. Méthodes agiles

3.1. Introduction

Une méthode agile est une méthode de développement informatique permettant de concevoir des logiciels en impliquant au maximum le demandeur (client) :

- plus grande réactivité à ses demandes

- plus pragmatiques que les méthodes traditionnelles
- recherche de la satisfaction réelle du besoin du client

3.2. Exemples de méthodes agiles

XP (EXtreme Programming), DSDM (Dynamic Software Development Method), ASD (Adaptative Software Development), CCM (Crystal Clear Methodologies), SCRUM,FDD (Feature Driven development)

3.3. Priorités des méthodes agiles

- Priorité aux personnes et aux interactions sur les procédures et les outils
- Priorité aux applications fonctionnelles sur une documentation pléthorique
- Priorité à la collaboration avec le client sur la négociation de contrat
- Priorité à l'acceptation du changement sur la planification

3.4. EXtreme Programming (XP)

XP est une méthode basée sur des pratiques qui sont autant de boutons poussés au maximum

- Méthode qui peut sembler naturelle mais concrètement difficile à appliquer et à maîtriser
- Réclame beaucoup de discipline et de communication
- Aller vite mais sans perdre de vue la rigueur du codage et les fonctions finales de l'application.
- Force de XP : sa simplicité et le fait qu'on va droit à l'essentiel, selon un rythme qui doit rester constant.

3.5. Valeurs d'XP

Communication

- XP favorise la communication directe, plutôt que le cloisonnement des activités et les échanges de documents formels.
- Les développeurs travaillent directement avec la maîtrise d'ouvrage
- Feedback
- Les pratiques XP sont conçus pour donner un maximum de feedback sur le déroulement du projet afin de corriger la trajectoire au plus tôt.
- Simplicité : Du processus, Du code,
- Courage : d'honorer les autres valeurs, de maintenir une communication franche et ouverte, d'accepter et de traiter de front les mauvaises nouvelles.

3.6. Cycle de vie XP

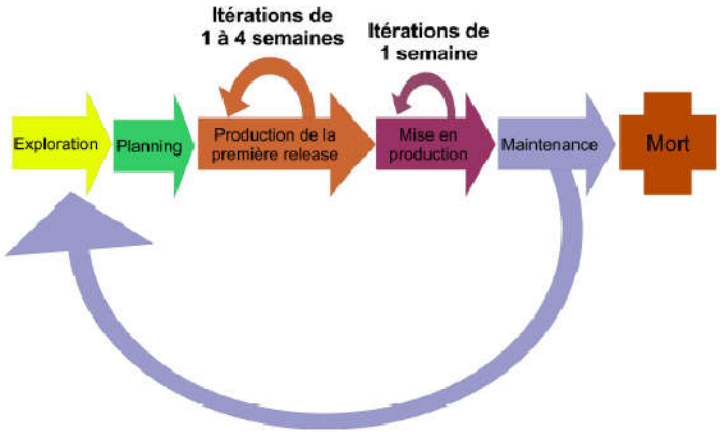

Figure 3 Cycle de vie XP.

a) Exploration

-Les développeurs se penchent sur des questions techniques
-Explorer les différentes possibilités d'architecture pour le système
-Etudier par exemple les limites au niveau des performances présentées par chacune des solutions possibles
-Le client s'habitue à exprimer ses besoins sous forme de user strories (proches de diagrammes de cas illustrés par des diagrammes de séquences)
-Les développeurs estiment les temps de développement

b) Planning

-Planning de la première release :
-Uniquement les fonctionnalités essentielles
- Première release à enrichir par la suite
- Durée du planning : 1 ou 2 jours
- Première version (release) au bout de 2 à 6 mois

c) Itérations jusqu'à la première release

-Développement de la première version de l'application
-Itérations de une à quatre semaines
-Chaque itération produit un sous ensemble des fonctionnalités principales
-Le produit de chaque itération subit des tests fonctionnels
-Itérations courtes pour identifier très tôt des déviations par rapport au planning
-Brèves réunions quotidiennes réunissant toute l'équipe, pour mettre chacun au courant de l'avancement du projet

d) Mise en production

-La mise en production produit un logiciel
-Offrant toutes les fonctionnalités indispensables
-Parfaitement fonctionnel
-Mis à disposition des utilisateurs
-Itérations très courtes
-Tests constants en parallèle du développement
-Les développeurs procèdent à des réglages affinés pour améliorer les performances du logiciel.

e) Maintenance

-Continuer à faire fonctionner le système
-Adjonction de nouvelles fonctionnalités secondaires
-Pour les fonctionnalités secondaires, on recommence par une rapide exploration
-L'ajout de fonctionnalités secondaires donne lieu à de nouvelles releases

f) Mort

-Quand le client ne parvient plus à spécifier de nouveaux besoins, le projet est dit _ mort _
-Soit que tous les besoins possibles sont remplis
-Soit que le système ne supporte plus de nouvelles modifications en restant rentable

3.7 Equipe XP

Pour un travail en équipe, on distingue 6 rôles principaux au sein d'une équipe XP

-Développeur (Conception et programmation, même combat, Participe aux séances de planification, évalue les tâches et leur difficulté, Définition des tests unitaires, Implémentation des fonctionnalités et des tests unitaires).

-Client (Écrit, explique et maîtrise les scénarios, Spécifier les tests fonctionnels de recette, Définit les priorités)
-Testeur (Écriture des tests de recette automatiques pour valider les scénarios clients, Peut influer sur les choix du clients en fonction de la _ testatibilité _ des scénarios)

_ Tracker (Suivre le planning pour chaque itération, Comprendre les estimations produites par les développeurs concernant leur charges, Interagir avec les développeurs pour le respect du planning de l'itération courante, Détection des éventuels retards et rectifications si besoin Manager, Supérieur hiérarchique des développeurs)

_ Manager (Responsable du projet, Vérification de la satisfaction du client, Contrôle le planning, Gestion des ressources)

_ Coach (Garant du processus XP, Organise et anime les séances de planifications, Favorise la créativité du groupe, n'impose pas ses solutions techniques, Pas de documents d'analyse ou de spécifications détaillées, Les tests de recette remplacent les spécifications, Emergence de l'architecture au fur et à mesure du développement.

3.8. Comparaison entre XP et RUP

-XP pour les petits projets en équipe de 12 max
-RUP pour les gros projets qui génèrent beaucoup de documentation

3.8.1. Inconvénients de XP

-Focalisation sur l'aspect individuel du développement, au détriment d'une vue globale et des pratiques de management ou de formalisation.
-Manquer de contrôle et de structuration, risques de dérive

3.8.2. Inconvénients de RUP

-Fait tout, mais lourd, (usine à gaz _
-Parfois difficile à mettre en oeuvre de façon spécifique.

Bibliographie

1. Grady Booch, James Rumbaugh et Yvar Jacobson,"Le guide de l'utilisateur UML", Eyrolles 2000.
2. Craig Larman, "Applying UML and Patterns", Addison Wesley/Pearson Education, 1999.
3. Kent Beck,"Extreme Programming Explained: Embrace Change", Addison Wesley/Pearson Education, 2000.
4. Craig Larman,"Agile and Iterative Development: A Manager's Guide", Addison Wesley/Pearson Education, 2003.