Applications – Gestion de projets de SI

Application 1 – Méthode de développement

Document 1

Dans les grandes entreprises, il est nécessaire d'aller de plus en plus vite pour offrir aux utilisateurs les outils dont ils ont un besoin vital. De même, la mise en production d'une application développée « entre informaticiens » n'est plus concevable : les risques de rejet pur et simple par les utilisateurs de l'application produite sont trop importants. Il s'agit de trouver un moyen de développer des applications selon une méthodologie permettant de répondre à ces besoins cruciaux pour l'entreprise : il faut aller vite (un marché se gagne plus facilement quand l'entreprise y est présente rapidement et efficacement) ; il faut produire des logiciels correspondant exactement aux besoins des utilisateurs ; il faut enfin garantir une réactivité importante face aux évolutions des marchés concurrentiels.

La remise en cause de la méthode employée pour produire les logiciels de l'entreprise est une obligation. Les méthodes « anciennes », trop linéaires et souvent « mal » appliquées, qui amènent à produire une documentation volumineuse, redondante, jamais à jour et que de toutes façons « personne ne lit vraiment », ne répondent pas à ces nouveaux besoins. Il est alors tentant d'examiner de nouvelles solutions. Comme souvent, celles-ci sont nées de l'autre côté de l'Atlantique. Le développement rapide d'applications (Rapid Application Development ou RAD) est une réponse possible. Inventée par l'Américain James Martin, cette méthode offre des avantages importants :

- la forte implication des futurs utilisateurs de l'application permet de garantir l'adéquation entre les besoins exprimés et le logiciel produit ;
- la logique économique qu'elle implique interdit le développement de fonctionnalités « inutiles » ;
- l'utilisation judicieuse des outils informatiques disponibles oriente vers la production d'une documentation nécessaire et suffisante ;
- le respect strict de l'enveloppe budgétaire et des délais permet d'avoir une vision stratégique efficace.

Le principe fondamental du RAD est le suivant : il s'agit de fixer, dès l'initialisation du projet, une enveloppe temps/argent dans laquelle le projet doit impérativement s'inscrire. Le projet étant construit intégralement avec les utilisateurs, c'est à eux, avec l'aide d'un animateur RAD, qu'incombe la tâche de faire cadrer le projet avec le budget défini. L'ensemble des phases du projet est couvert par le RAD et réalisé avec les futurs utilisateurs du système : de la phase de conception de la base de données jusqu'à la mise au point des écrans et des états produits, par itérations successives de prototypage. Il en résulte alors l'obligation de ne développer que des fonctionnalités « utiles », en éliminant les développements particuliers n'emportant pas l'adhésion générale. Il est souvent demandé par des utilisateurs des versions multiples d'une restitution (qu'elle soit imprimable ou consultable à l'écran) : dans le cas d'un projet RAD, on cherche à produire une restitution unique, rassemblant l'ensemble des informations et permettant d'emporter les suffrages de chacun des participants au projet.

Le RAD ne permet pas de traiter des projets dont la charge prévisible est trop importante (supérieure à trois années/homme). Dans ce cas, un lotissement est nécessaire afin de découper le projet en autant de sous-projets compatibles avec les exigences de la méthode. Il est en effet extrêmement délicat d'animer des réunions RAD mettant en cause un trop grand nombre d'utilisateurs différents sur un même projet (les risques de « dérive » des réunions sont alors importants).

La RAD fait partie des méthodes Agiles. Les **méthodes Agiles** sont des procédures de conception de logiciel qui se veulent plus pragmatiques que les méthodes traditionnelles. En impliquant au maximum le demandeur (client), ces méthodes permettent une grande réactivité à ses demandes, visent la satisfaction réelle du besoin du client, et non des termes du contrat de développement. La notion de méthode agile a été officialisée en 2001 par un document Manifeste Agile (*Agile Manifesto*) signé par 17 personnalités impliquées dans l'évolution du génie logiciel et généralement auteurs de leur propre méthode.

Les méthodes Agiles étaient antérieures au Manifeste Agile. Le manifeste Agile n'est donc pas l'acte de naissance des méthodes Agiles ou du mouvement Agile, ce n'est que la formalisation consensuelle par les auteurs de ces méthodes, toutes nées dans la deuxième partie de la décennie 90, du fait qu'elles avaient des valeurs communes, une structure (cycle de développement) commune (itérative, incrémentale et adaptative) et une base de pratiques, soit communes, soit complémentaires. Parmi ces méthodes on trouve DSDM (1995) la version Anglaise du RAD (Développement rapide d'applications) de James Martin et plusieurs autres méthodes comme ASD ou FDD reconnaissant leur parenté directe avec RAD la première méthode Agile publiée (1991). Les deux méthodes Agiles les plus connues en France sont: la méthode Scrum (1996) et la méthode XP pour Extreme programming (1999).

La notion de méthode agile a émergé avec des pratiques ciblant uniquement le développement d'une application informatique. Mais un mouvement managérial plus large (Management Agile) commence à coupler les valeurs Agiles aux techniques de l'amélioration continue de la qualité (MTQS ou Lean).

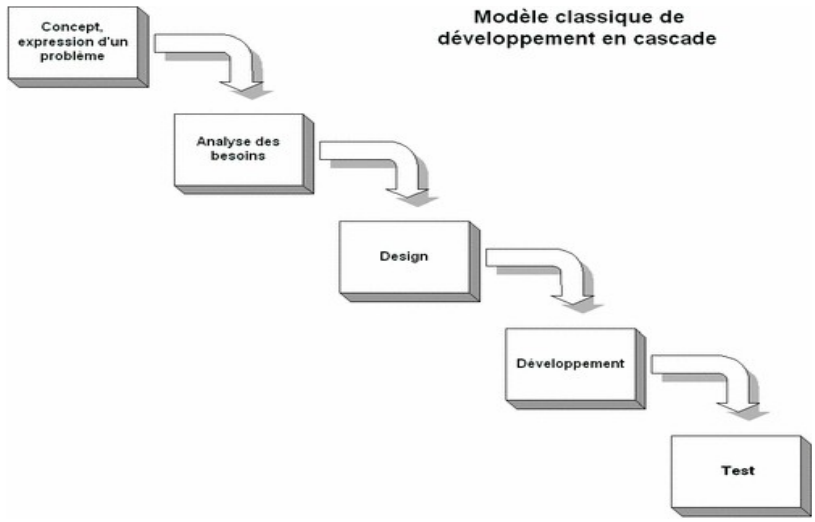
Il sera question dans les sections qui suivent de vous présenter de manière claire la méthode RAD: phases, avantages limites.

Document 2

Depuis les débuts de l'informatique commerciale dans les années '60, plusieurs méthodologies de développement de logiciel ont vu le jour. Le modèle en cascade et ses dérivés ont connu un grand succès, mais leur lourdeur et rigidité sont de sérieux handicaps. Extreme Programming propose de remplacer certaines notions acquises par des idées révolutionnaires, et de rendre le développement de logiciel efficace et agile.

Le développement de logiciels a été pendant longtemps entièrement libre de directives ou de contraintes, et il n'y avait aucune méthode pour analyser les besoins, produire une solution, ou même évaluer le succès. En 1968, une conférence de l'OTAN a introduit le terme « génie logiciel » et suggéré d'utiliser les méthodes rigoureuses et éprouvées du génie civil au chaos du développement de logiciels. L'idée était louable, mais une différence majeure existe: contrairement à la physique et aux mathématiques, l'informatique ne repose sur aucune loi et ne peut être vérifiée scientifiquement.

Des méthodologies de développement sont apparues à différents moments durant la révolution informatique. Le modèle en cascade, inventé par la US Navy, est sans aucun doute le modèle qui a eu le plus d'influence, et cette influence peut encore être ressentie aujourd'hui. Le modèle est très strict: les étapes de concept, analyse, design, programmation et test doivent être exécutées dans l'ordre, et le retour en arrière n'est pas permis. La notion que le coût du changement augmente à mesure que le projet progresse est dérivée de ce modèle. L'emphase sur la documentation est très importante, et chaque étape doit être approuvée avant que l'étape suivante débute. Très bureaucratique, très lourd, mais un grand bond dans la bonne direction.



Le modèle en cascade a donné naissance à de nombreuses autres méthodologies: prototypage, modèle en spirale, implantation en étape, et même RAD (Rapid Application Development). Ces méthodologies sont des variations de l'original, où des possibilités de retour-arrière à différents points ont été ajoutées. Au milieu des années '90, l'industrie informatique utilisait le terme RAD à toutes les sauces, et promettait des résultats fantastiques. Les fabricants de langages de programmation, dont Borland et Microsoft, ont lancé des produits de développement qui ont effectivement accéléré le processus de programmation, mais n'avaient rien à voir avec une méthodologie de travail. En fait, dans la grande majorité des cas, les outils contribuaient à diminuer la qualité.

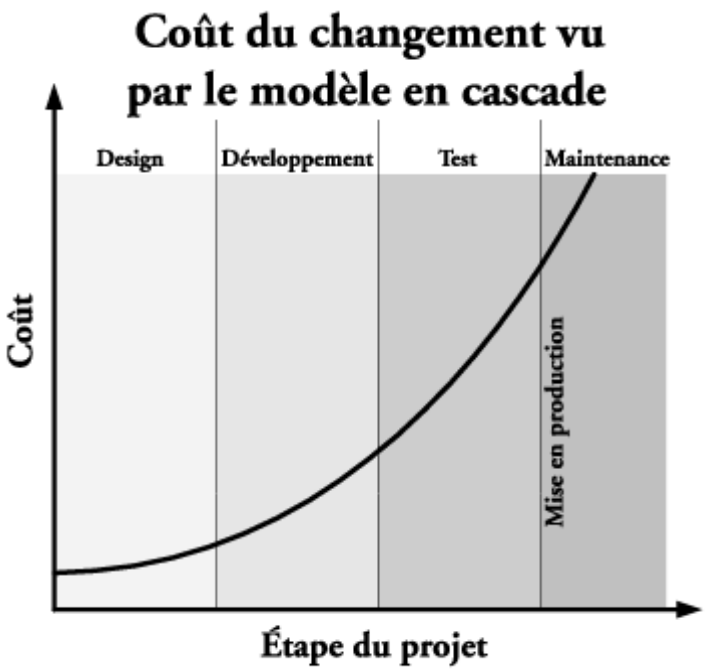
À travers toutes ces méthodes et tous ces modèles, de nombreuses bonnes idées sont apparues. Malheureusement, elles sont souvent mal utilisées, ou perdues au coeur de la rigueur et de la paperasse générée par le modèle. Un mouvement est apparu au milieu des années '90 qui avait pour but de rendre le processus de développement plus efficace, en utilisant des concepts très simples et reconnus. Ce mouvement s'appelle Agile Modeling. Il n'est directement relié à aucune méthodologie, mais propose une série de recommandations, valeurs, et principes qui sont appliquées dans d'autres méthodes. C'est de là que Extreme Programming est né.

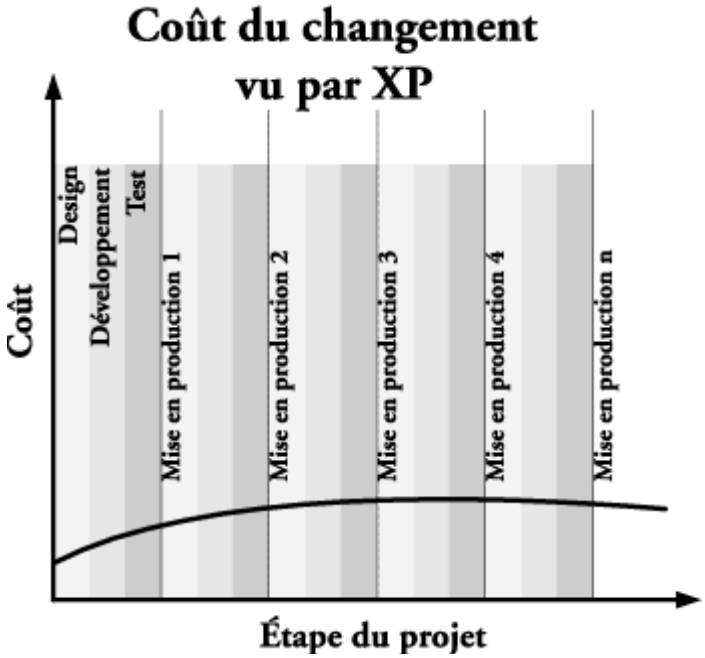
Extreme Programming, ou XP, est basé sur des principes très simples, mais souvent ignorés par l'industrie. Une des idées révolutionnaires est que le coût du changement n'est pas variable mais plutôt constant; XP accepte donc le changement comme une réalité et l'intègre dans le processus de développement. Aussi, la programmation en paire, où deux programmeurs travaillent ensemble sur le même ordinateur, permet d'augmenter la qualité à des niveaux encore jamais vus. Une autre idée consiste à mettre l'emphase sur la communication constante entre l'équipe de développement et le client, par le biais de boucles rapides développement-test-feedback. Certaines de ces idées ne sont pas nouvelles, mais elles sont assemblées dans XP pour former une méthodologie efficace et qui présente des résultats concrets, plutôt que des résultats sur papier.

Les cours de génie logiciel, la littérature, et presque toutes les méthodologies de développement jusqu'au milieu des années ‘90 suggéraient que le coût relié à des changements dans les spécifications d'un problème ou le design d'une solution augmente de façon exponentielle à mesure que le projet avance. D'innombrables références et études sont disponibles dans des livres et articles, et l'idée est une notion acquise pour plusieurs générations de gestionnaires de projet. En fait, certaines méthodologies basées sur le modèle en cascade reposent entièrement sur cette prémisse et ignorent complètement les requêtes de changement, ou encore imposent des processus très stricts pour gérer ces requêtes (possiblement dans le but de les éliminer).

Dans la réalité, quelqu'un a-t-il déjà travaillé sur un projet d'informatique (ou n'importe quel projet)

où il n'y a pas eu de changement? Est-ce même possible? Peu importe la taille du projet, peu importe le contexte, peu importe l'industrie, le changement est inévitable. Depuis qu'il y a des projets d'informatique, il y a des requêtes de changement; pourquoi les ignorer? Pourquoi supposer que ce projet sera différent?
Le court cycle de développement utilisé dans XP règle le rythme du projet, et a un effet direct sur la qualité du produit fini. En ayant des mises en production fréquentes, les éléments les plus importants sont produits en premier; les mises en production suivantes permettent d'ajouter des fonctionnalités tout en testant le résultat à chaque nouvelle étape.





Le graphique présente le coût du changement d'après XP. On peut voir que le coût du changement augmente légèrement durant les premières étapes, alors que le design devient plus complexe et que l'effet d'un changement a de plus en plus de répercussions. Après un certain point, cependant, le coût atteint un plateau et diminue légèrement; ceci est causé par la « factorisation » du programme (expression pompeuse pour dire « simplification »).

Une des grandes forces de XP est d'accepter le changement comme une réalité, et de l'inclure comme partie intégrale du processus de développement. Les boucles rapides de développement-test-feedback permettent d'intégrer le changement de façon contrôlée, sans avoir à “revenir en arrière”. En permettant au changement de prendre place à tout moment, il est possible de prédire le temps et le coût associé, et de rapidement présenter au client la conséquence de ce changement. Le coût du changement devient donc une constante plutôt qu'une variable.
Les caractéristiques de XP :

- Programmation en paire. La Programmation en paire est l'arme secrète de Extreme Programming. En permettant aux développeurs d'échanger des idées et de se corriger mutuellement, la qualité des programmes atteint des niveaux très élevés tout en diminuant les délais. Mais le coût de développer un projet en utilisant 2 programmeurs est plus élevé, et les gestionnaires auront tendance à rejeter

l'idée. Un des points distinctifs de Extreme Programming est la programmation en paire, où deux programmeurs travaillent ensemble à la même station de travail. La programmation en paire est tranquillement en train de prendre sa place dans l'industrie. C'est une méthode efficace et reconnue d'augmenter la qualité et la vitesse du processus de développement.

- Le cycle Developpement-test-feedback. En contraste avec les étapes de design, développement, test et mise en production strictement indépendantes les unes des autres utilisées dans le modèle en cascade et ses derivés, Extreme Programming propose d'utiliser un cycle de développement très court avec des mises en productions fréquentes. L'objectif est d'augmenter la flexibilité, diminuer le risque associé à de longs délais, et créer de la valeur le plus rapidement et le plus souvent possible pour le client. Extreme Programming propose d'utiliser un cycle très court de design, développement et test, avec des mises en productions fréquentes.

Autre document à consulter :
http://www.journaldunet.com/solutions/dsi/dossier/gestion-de-projets-informatiques-les-secrets-de-la-reussite/les-methodes-agiles-remettent-le-client-au-coeur-du-projet.shtml

1- Qu'est ce que le cycle de développement d'un logiciel ? En quoi est ce que la réflexion sur la méthode de développement est nécessaire ?

2- Expliquer la notion de méthode agile. A quoi s'oppose t-elle ? Quelles sont les principales méthodes agiles décrites dans le texte ?

3- Quel est le grand changement qu'apportent les méthodes agiles dans la démarche de développement ?

4- Expliquez la notion de réduction des coûts induite par l'eXtreme Programming

5- Donnez deux méthodes de modélisation possibles. Ces méthodes sont elles plus générales ou moins générales que les méthodes XP et RAD ?

Application 2 – Méthode de développement

A partir du texte suivant :
http://www.journaldunet.com/solutions/dsi/lenny-descamps-apmg-international-showcase-france.shtml

1- A quoi correspond la notion de gestion de projet IT ?

2- Quels sont les problématiques actuelles de cette gestion de projet ?

Application 3 – Les échecs dans la gestion de projet en SI

http://www.silicon.fr/echec-projet-sap-coute-345-millions-euros-dhl-131307.html
http://www.athena-vostok.com/la-verite-sur-un-veritable-scandale-louvois
https://fr.wikipedia.org/wiki/Logiciel_unique_%C3%A0_vocation_interarm%C3%A9es_de_la_solde
http://www.silicon.fr/louvois-ministere-defense-choisit-sopra-hr-access-112691.html
http://bfmbusiness.bfmtv.com/france/l-armee-commande-un-nouveau-logiciel-de-paie-pour-faire-oublier-le-fiasco-de-louvois-880255.html

http://www.zdnet.fr/actualites/externalisation-de-l-informatique-de-la-sncf-aupres-d-ibm-l-echec-et-le-divorce-39766338.htm
http://www.alliancy.fr/juridique/entreprises/2015/03/10/ibm-condamne-pour-lechec-dun-projet-dintegration
http://bfmbusiness.bfmtv.com/01-business-forum/externalisation-des-rates-chez-renault-315768.htmls
http://www.piloter.org/projet/facteur-echec/index.htm
https://pmiquebec.qc.ca/index.php/articles-du-mois/242-pourquoi-les-grands-projets-en-ti-connaissent-ils-une-si-mauvaise-performance
http://www.finyear.com/Echec-des-projets-informatiques-4-causes-majeures_a12238.html

A partir des textes précédents, expliquer les facteurs qui expliquent les échecs dans les projets en SI. Est-il possible de catégoriser les échecs ?

Application 4 – L'organisation de la gestion projet dans les entreprises.

http://www.journaldunet.com/solutions/dsi/bnp-paribas-personal-investors-et-la-gestion-de-projets-it.shtml

Expliciter la spécificité du management de projet décrit dans ce document.