I - Cycle de développement hybride

Cascades en spirale

Un modèle de processus hybride

A. Motivation

Les deux modèles “classiques” de processus de développement logiciel sont la **Cascade** et la **Spirale**. Ces modèles ont été développés à une époque où :

- les langages procéduraux étaient prépondérants. Même l’Assembleur était un langage assez courant !
- les grands systèmes centralisés étaient la norme. Ex : de grandes applications Cobol sur des mainframes MVS;
- les cycles Codage/Débogage/Correction étaient relativement longs : souvent comptés en années;
- les cycles CPU étaient une denrée rare, et fréquemment rationnée;
- la technologie de l’interface utilisateur était dans sa toute petite enfance.

Depuis leur définition, ces modèles de processus ont essuyé plusieurs changements technologiques. Bien que plusieurs alternatives aient été proposées et aient gagné quelque audience dans l’industrie, les modèles Cascade et Spirale constituent toujours la base de la plupart des Systèmes Informatiques de Gestion et des SSII à travers le monde.

Dès la fin des années 80 et le début des années 90, il était reconnu que la technologie des PC faisait que les développements devenaient plus itératifs. Aujourd’hui, toutefois, les progrès technologiques réclament une évolution des modèles classiques Cascade et Spirale. Certains de ces changements sont les suivants :

- Prédominance croissante des méthodes orientées-objet;
- Technologies dédiées Internet/Intranet : Navigateurs, Java, HTML...
- Architectures distribuées : Multi-tier et Client-serveur pour de nombreuses applications;
- Plates-formes matérielles peu onéreuses. MIPS, mémoire et capacité disque sont relativement bon marché;

- Cycles débogage/correction courts;
- Délai de mise sur le marché très court : mesuré en mois;
- Interfaces utilisateur sophistiquées, et grande emphase mise sur la convivialité.

B. La Cascade

Les forces et les faiblesses du Modèle de Processus en **Cascade** sont montrées en figure 1. Il est reconnu dans l'industrie du logiciel que la meilleure qualité que l'on puisse atteindre en une passe de développement est de 90%. Les exemples de l'industrie montre que ce niveau de qualité est rarement atteint. Par exemple, si un produit logiciel est bon seulement à 50 % à son développement initial, il faudra 4 passes pour atteindre 90% (en fait il atteindrait 93,75% à la fin de la quatrième passe : 50% -> 75% -> 87,5% -> 93,75%). Combien de chances ce produit aurait d'être accepté par le marché dès la deuxième passe : probablement aucune.

C. Le processus en Cascade : forces et faiblesses

Points forts	**Points faibles**
Donne une structure à un processus potentiellement non structuré.	Limite la consultation de l'utilisateur final au début (analyse des besoins) et la fin (déploiement du produit).
Correspond bien au processus de développement du hardware.	Plus efficace sur les grands projets comportant davantage de risques et des retards potentiels.
Déterministe pour les modèles de mise sur la marché à date stricte.	L'approche inflexible ne réagit pas bien aux demandes de changement : changements de conception ou changements fonctionnels.
Il se dirige lui-même vers un management de programme par phases.	Provoque des désavantages dans la compétition de date de mise sur le marché
Figure 1	

D. La Spirale

De nombreuses organisations de développement ont été réticentes à utiliser le processus de développement en spirale.

Un tel processus est moins déterministe - du point de vue du management : combien de “spirales” seront nécessaires ? Il est aussi difficile de déterminer l’état du produit - en terme d’aptitude au déploiement par le client - à n’importe quel moment du cycle de vie du projet. Les traditionnels processus de mesures et de métriques - telles que la densité de défauts ou les courbes cumulées d’incidents - ne sont pas vraiment utiles pour le management de projet.
L’implication du client est difficile à obtenir parce qu’il est plus difficile de prédire le niveau de qualité que le produit aura atteint à une date particulière dans le futur.
Toutefois, plusieurs directeurs de services informatiques ont obtenu de grands succès avec le développement en spirale. Le dénominateur commun de leurs succès est de s’en être tenu à des projets de taille réduite : 2-3 développeurs sur une durée de 2-3 mois.
Un modèle de processus peut-il être développé pour aider les développeurs des produits logiciels et les services informatiques à réaliser à la fois le meilleur des deux processus : Spiral et Cascade ? Oui.

E. Le processus en Spirale : forces et faiblesses

Points forts	Points faibles
Processus flexible - très adaptable aux modifications de conception et modifications d'exigences utilisateurs.	Moins déterministe - plus complexe à gérer.
Peut résulter en une mise sur le marché plus rapide.	Difficile pour le manager de déterminer l'état du projet à un moment donné.
Peut améliorer la qualité du produit fabriqué.	Processus plus risqué que le processus en cascade.
Le résultat final peut correspondre de plus près aux exigences utilisateur.	Plus difficile d'obtenir l'implication du client pour les livrables du produit.
Figure 2	

F. Les deux processus dans la nouvelle réalité

L'expérience a montré que les paramètres déterminant l'adéquation d'un modèle de processus sont la durée du cycle codage/déboguage/correction, et la complexité des composants à mettre en œuvre *(voir figure 3).*

Les modules logiciels de complexité faible ou moyenne, et qui sont mis en œuvre avec des technologies ayant un cycle codage/déboguage/correction court, sont plus adaptés au modèle de processus en **Spirale**.

Les modules impliquant une complexité élevée ou qui utilisent des technologies ayant un cycle codage/déboguage/correction long, sont plus adaptés au modèle de processus en **Cascade**.

		Durée du cycle Codage/Débogage/Correction		
		Courte	**Moyenne**	**Longue**
Complexité	**Simple**	Spirale	Spirale	Cascade
	Moyenne	Spirale	Spirale	Cascade
	Complexe	Cascade	Cascade	Cascade
	Figure 3			

Les nouvelles technologies telles que Java et HTML ont considérablement réduit les cycles codage/déboguage/correction. Les approches Orientés Objet peuvent accroître la réutilisation et réduire les cycles de développement. Les applications Client-Serveur peuvent être développées plus rapidement que les applications centralisées. L'utilisation de ces technologies rend possible, pour beaucoup de projets, l'utilisation du processus de développement en **Spirale**.

En réalité, toutefois, de nombreux projets logiciels utiliseront un mélange des diverses technologies. Les modèle de processus traditionnels et alternatifs ont ignoré la mixité de complexité et de technologies dans beaucoup de projets.

Le modèle de processus de développement **Hybride**, montré en figure 4, permet à un projet de mettre à profit les bénéfices du processus de développement en **Spirale** et de celui en **Cascade**, plus quelques bénéfices supplémentaires :

- Ajoute de la structuration pour une gestion de projet plus déterministe;
- Permet de fréquentes interventions de l'utilisateur pendant le développement;
- Plus grande flexibilité pour s'adapter aux modifications des exigences et de la conception;
- Correspond bien aux stratégies de développement basées sur le métier du client;
- Le produit final sera de meilleure qualité et plus près des besoins de l'utilisateur final;
- Peut être mis en œuvre avec les modèles de développement par phases.

Le modèle de processus **Hybride** présente trois sections clés :

1. Architecture et planning;
2. Développement itératif et déploiement itératif de la plupart des composants majeurs;
3. Développement linéaire des composants qui ne peuvent pas être développés de façon itérative.

L'architecture et le planning (incluant l'analyse des besoins) doivent être effectués en premier. Ils constituent les fondements des activités subséquentes dans le projet.

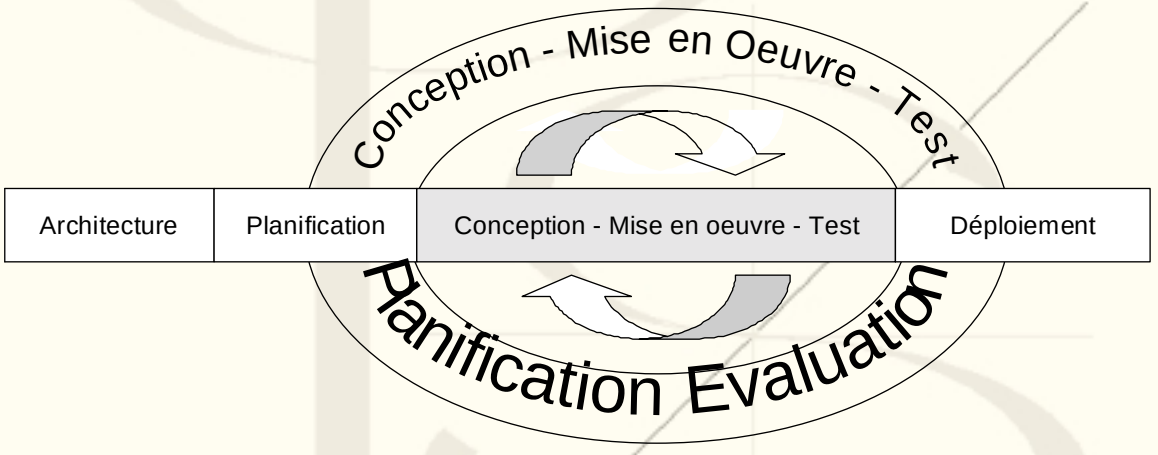
Le développement itératif, littéralement, "tourne en cercles" autour du composant à développement linéaire. L'exécution en parallèle de ces activités minimise le temps de mise sur le marché.

Figure 4

Architecture et planification

La conception architecturale fournit les fondations techniques pour tous les modules du projet. Elle s'assure que les modules vont bien s'assembler lorsqu'ils seront intégrés dans le projet final.

Les modules logiciels développés **itérativement** peuvent être intégrés en tant que pièces de chaque itération, tandis que les modules développés **linéairement** ne seront disponibles que lorsqu'ils seront terminés. Pour cette raison, une architecture commune bien décrite aidera à



assurer que les modules développés itérativement s'intégreront bien avec les modules développés de façon linéaire, et aussi avec les autres composants du projet.

L'analyse des besoins s'effectue parallèlement aux activités d'Architecture et de Planning.

Le développement logiciel est une entreprise complexe par nature. Les plannings techniques et de projet sont essentiels pour optimiser la qualité, les délais et les coûts du projet.

Certaines décisions essentielles de planification spécifiques au modèle de processus hybride sont les suivantes :

- Quels sont les composants qui peuvent être développés de façon itérative et non pas de façon linéaire;
- Quelle fonctionnalité ou caractéristique cible sera implémentée à chaque itération;
- A quel degré les modules seront-ils déployés lors de la conclusion de chaque itération.

Plus nombreux seront les composants pouvant être mis en œuvre de façon itérative, plus grands seront les bénéfices pour le projet. Une architecture bien conçue peut grandement

réduire la complexité de nombreux modules logiciels - accroissant ainsi le nombre de modules pouvant être implémentés de façon itérative.

Le nombre d'itérations à effectuer peut être basé sur deux facteurs :

- La durée d'une itération;
- La durée du composant à développement linéaire.

Il est probable que trois itérations moyennement efficaces fourniront un produit de haute qualité qui satisfera de très près les exigences de l'utilisateur final.
Par exemple, si chaque itération satisfait 75% des exigences de l'utilisateur final, le produit aura satisfait plus de 98,4% de ces exigences à la fin de la troisième itération[1]. Ceci comparé aux 90% de la limite théorique d'une itération simple !
L'exécution d'un projet sera plus efficace si la durée d'une itération est fixée à un tiers de la durée du développement linéaire.

Développement itératif

Chaque itération devra suivre les cinq étapes générales suivantes :

1. Conception
2. Mise en œuvre
3. Test
4. Évaluation
5. Planification

Premièrement, la **conception** détaillée de chaque module logiciel doit être menée à bien. Noter que l'effort de conception doit, dans la mesure du possible, être adapté à la complexité de la fonctionnalité ou caractéristique implémentée en tant que pièce de l'itération en cours.
La conception est suivie par la **mise en œuvre** de la fonctionnalité ou caractéristique cible pour cette itération. Après mise en œuvre, elle doit être **testée** - à la fois en test unitaire et en test d'intégration avec les autres composants développés itérativement.

[1] 75% + 18,75% + 4,687% = 98,437%

Après les tests, **l'évaluation** devra être effectuée pour collecter les retours d'information de l'utilisateur final. Noter que l'évaluation peut être effectuée par un déploiement contrôlé du logiciel. Les contrôles devront permettre le retour d'information de façon structurée, afin de permettre son incorporation dans la prochaine itération. Après la collecte des informations en retour, celles-ci seront utilisée en conjonction avec le plan global du projet pour planifier la prochaine itération. En résumer, il s'agit de mettre en place un planning de projet, focalisé sur la prochaine itération.

Développement linéaire

Les modules développés linéairement sont mis en œuvre en utilisant la séquence traditionnelle en Cascade : conception, mise en œuvre, test des fonctionnalités requises. Puisque les autres modules sont développés de façon itérative, il faut toutefois prendre quelques considérations particulières. Par exemple, une simulation ou des modules "souche" devront peut-être être développés pour permettre aux modules développés itérativement de fonctionner à chaque itération.

Facteurs de succès

Le succès de l'utilisation du modèle de processus **Hybride** met davantage d'emphase sur les trois points suivants :

- Architecture
- Gestion du projet
- Gestion des risques

a. Architecture

Une architecture bien conçue, unifiée, est essentielle pour la mise en œuvre avec succès de modèle de processus **Hybride**.
A cause d'un nombre de développements parallèles supérieur, il y a un plus grand risque de problèmes sérieux lorsque les modules seront intégrés. Une architecture unifiée réduira ce risque de façon significative.
Une architecture conçue de façon adéquate réduira aussi la complexité de chaque module logiciel - ce qui permettra que davantage de modules soient développés de façon itérative, et accroîtra le niveau de parallèlisation des développements. Enfin, elle permettra une réduction du délai de mise sur le marché.

a. Gestion de projet

La gestion du projet est la seconde clé du succès pour la mise en œuvre du processus **Hybride**.
A tout moment dans le projet, il y aura un certain nombre d'activités diverses menées en parallèle, concurremment. Par exemple, développement et validation par l'utilisateur final. Un management de projet fort est essentiel pour gérer ces activités et conserver le contrôle du projet.
La première activité clé est la mise en place du planning initial, en priorité sur le démarrage du développement itératif. Le pilotage et des ajustements continus sont requis à la fois dans la portion itérative du projet, et dans sa partie linéaire. De plus, à la conclusion de chaque itération, la planification de la prochaine itération devra être bien effectuée - et devra être liée à la partie linéaire du projet.

a. Gestion des risques

La gestion des risques est la clé finale du succès du processus de développement **Hybride**. De même que l'architecture et la gestion du projet, la gestion des risques est un facteur clé du succès de n'importe quel projet logiciel. Une gestion des risques pro-active, agressive, préviendra les problèmes avant qu'ils ne surviennent, réduira le temps nécessaire à lutter contre les incidents, et réduira également le délai de mise sur le marché. Une gestion des risques efficace aidera les équipes-projet à optimiser le modèle de processus **Hybride** en fonction de leur projet, et à minimiser l'impact du traitement des demandes de changement.
Plusieurs outils de gestion des risques sont bien adaptés au modèle de processus **Hybride**. Par exemple le "Project Self-Assessment Kit" de Kulik & Lazarus Consulting, Inc., qui peut être utilisé à la fois sur les composants en développement itératif et linéaire.

Conclusion

Les progrès technologiques relativement récents provoquent des changements fondamentaux dans le développement logiciel. Java, méthodes OO , Internet, les intranets... Modifient les fondations économiques et fondamentales des modes de développement. Les deux modes de développement prééminents, **Cascade** et **Spirale**, sont généralement bien compris par de nombreuses organisations. Toutes les deux ont leurs forces et leurs faiblesses, mais aucune ne donne des performances optimales dans la réalité de l'environnement technologique et du monde des affaires d'aujourd'hui.
Les clés du succès pour la mise en œuvre du processus **Hybride** sont :

- Conception architecturale unifiée;

- Management de projet efficace;
- Gestion des risques agressive, pro-active.

Une attention appropriée à ces facteurs de succès assurera une mise en œuvre optimale du processus **Hybride**, réduisant le délai de mise sur le marché, améliorant la qualité, et améliorant l'acceptation du produit par les utilisateurs finaux.

Références

1. Boehm, Barry W., "A Spiral Model of Software Development and Enhancement", IEEE Computer, May 1988.
2. Royce, W.W., "Managing the Development of Large Software Systems: Concepts and Techniques", Proc. ICSE 9, Computer Society Press, 1987.
3. Parnas discusses an Incremental Software Development Model in IEEE Transactions on Software Engineering, March 1979.
4. Mills, H.D., Dyer, M., and Linger, R.C.,"Cleanroom Software Engineering", IEEE Software, September 1987.
5. Hanna, Mary Alice, "Beyond the Waterfall Lies a Brave New World", Software Magazine, June 1991.
6. Campbell, Ian, "The Intranet: Slashing the Cost of Business", International Data Corporation, 1997.
7. Phil Neches, Teradata Corporation, on or about 1985.