Intro aux méthodes agiles

Contenu

Malaise

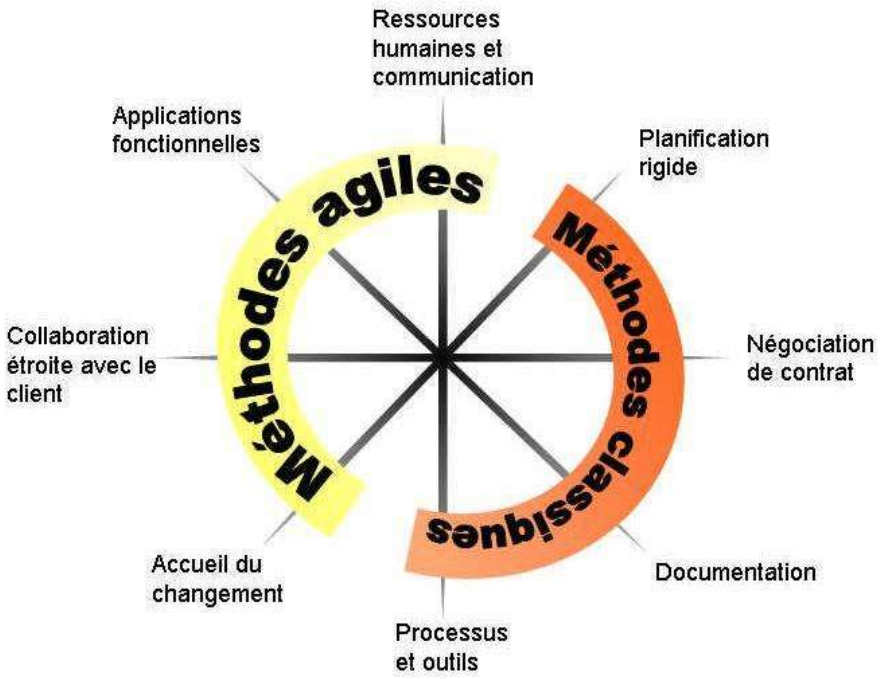
Les méthodes agiles répondent à un malaise.

Les méthodes agiles ont comme caractéristiques les préoccupations suivantes :

- Priorité des personnes et des interactions sur les procédures et les outils;
- Priorité d'applications opérationnelles sur une documentation exhaustive;
- Priorité de la collaboration avec le client sur la négociation de contrat;
- Priorité de l'acceptation du changement sur la planification.

Les principes des méthodes agiles :

- Délivrer rapidement et très fréquemment des versions opérationnelles, pour favoriser un feed-back client permanent;
- Accueillir favorablement le changement;
- Assurer une coopération forte entre client et développeurs;
- Garder un haut niveau de motivation;
- Le fonctionnement de l'application est le premier indicateur de l'avancement du projet;
- Garder un rythme soutenable;
- Viser l'excellence technique et la simplicité;
- Se remettre en cause régulièrement.



Tiré de la présentation « Méthodes agiles » de Jean-Louis Bernard, mai 2002

Différentes méthodes :

Plusieurs méthodes agiles sont disponibles aujourd'hui en voici quelques-unes parmi les plus significatives :

- XP (eXtreme Programming)
- ASD (Adaptive Software Development)
- FDD (Feature Driven Development)
- TDD (Test Driven Development)
- SCRUM

XP (eXtreme Programming)

L'eXtreme Programming est certainement la méthode agile la plus connue. Elle définit des pratiques portant sur la programmation, la collaboration entre les différents acteurs et la gestion de projet. Cette méthode fait apparaître dans ses principes fondateurs l'utilisation systématique du « refactoring » pour garantir une qualité constante aux versions livrées aux utilisateurs. Par rapport aux principes des méthodes agiles, l'eXtreme Programming introduit plusieurs éléments novateurs. Cette méthode sera décrite plus en détails un peu plus loin.

ASD (Adaptive Software Development)

L'ASD est fondé sur le principe de l'adaptation continue du fait de la nécessité d'accepter les changements continuels qui s'imposent aux logiciels. Ainsi, l'ASD est organisé autour d'un cycle en trois phases (spéculation, collaboration et apprentissage) en remplacement du cycle classique des projets informatiques (planification, conception et construction).

La spéculation

La spéculation comporte les cinq étapes suivantes :

1. Initialisation du projet, définissant la mission affectée au projet.
2. Planification générale du projet limitée dans le temps. Toute l'organisation du projet est centrée sur le respect de cette limite.
3. Définition du nombre d'itérations à effectuer et de leur date limite de livraison afin de respecter la limite globale du projet.
4. Définition du thème ou des objectifs de chaque itération.
5. Définition en concertation par les développeurs et les utilisateurs du contenu fonctionnel de chaque itération.

La collaboration

Pendant que l'équipe technique livre des versions opérationnelles du logiciel, les chefs de projet facilitent la collaboration et les développements en parallèle afin de respecter le planning du projet et les besoins des utilisateurs.

L'apprentissage

À la fin de chaque itération, une phase d'apprentissage est prévue. Cette phase est destinée à obtenir le feed-back le plus exhaustif possible sur la version livrée afin d'améliorer continuellement le processus. Le focus est mis sur la qualité du résultat, à la fois du point de vue des utilisateurs et du point de vue technique, ainsi que sur l'efficacité du mode de fonctionnement de l'équipe.

FDD (Feature Driven Development)

Contrairement aux méthodes précédentes, le FDD débute par une phase de conception générale, qui vise à spécifier un modèle objet du domaine du logiciel en collaboration avec les experts du domaine. Une fois le modèle du domaine spécifié et un premier recueil des besoins des utilisateurs effectué, les développeurs dressent une liste de fonctionnalités à implémenter. La planification et les responsabilités sont alors définies. Le développement du logiciel autour de la liste des fonctionnalités suit une série d'itérations très rapides, au rythme d'une itération toutes les deux semaines au maximum, composées chacune d'une étape de conception et d'une étape de développement.

TDD (Test Driven Development)

Le TDD place les tests au centre du développement logiciel. Il s'agit d'une démarche complémentaire des méthodes plus globales, comme l'XP, mais centrée sur le développement. Chaque développement de code, même le plus petit, est systématiquement précédé du développement de tests unitaires permettant de spécifier et vérifier ce que celui-ci doit faire. Puisque les tests portent sur du code qui n'existe pas encore, ils échouent si nous les exécutons. Nous pouvons dès lors ne développer que le code nécessaire et suffisant pour que les tests réussissent. Un « refactoring » est ensuite effectué pour optimiser à la fois les tests et le code testé, notamment en supprimant la duplication de code. Au fur et à mesure de l'avancée du projet, de plus en plus de tests sont développés, la règle étant que tout nouveau code ajouté ne doive pas les faire échouer. Les itérations du TDD sont beaucoup plus courtes qu'en XP puisqu'elles s'enclenchent à chaque morceau de code significatif, comme une méthode de classe. Leur fréquence varie donc de quelques minutes à une heure environ. En procédant de la sorte, nous garantissons le respect des spécifications et la non-régression à chaque itération.

Tiré de « Refactoring des applications Java/J2EE » de Jean-Philippe Retaillé, Eyrolles 2005.

Scrum

« Scrum » est une autre de ces méthodes agiles conçues pour un usage au sein d'une petite équipe, cherchant à maximiser sa productivité au travers de "règles de vie" facile à adapter à son cycle de développement.

Scrum tire son nom du terme anglais "mêlée", au Rugby. Le nom a été choisi pour l'analogie que constituent les réunions quotidiennes de Scrum avec la mêlée.

Le rugby est un sport de combat collectif, le rugby fait appel à la force et à la vélocité, et nécessite une forte organisation stratégique. Une équipe essaie d'avancer en restant unie en se passant le ballon de main en main.

Conçue en 1993 et formalisée en 1995, cette méthode de développement (orientée gestion de projet) **inclut souvent des pratiques venant de XP**.

L'idée de Scrum est de tenir compte de la réalité de la plupart des projets pour lesquels il n'est pas

possible de tout définir dès le début. Les travaux à faire sont ajustés régulièrement au cours du projet, notamment à la fin de chaque itération, appelée le "Sprint".

XP ou Programmation extrême

Introduction

L'Extreme Programming (XP) est une méthode de développement légère (axée sur la réalisation) dédiée à de petites équipes confrontées à un environnement changeant ou à des besoins mal connus. XP est né dans une communauté « *SmallTalk* ». Cette méthode est née sur le terrain, à partir des observations et recherches de développeurs expérimentés soucieux de se concentrer sur les nécessités élémentaires du développement :

- **Développer vite** : s'assurer que les changements restent toujours faciles pour conserver une vitesse de développement soutenue tout au long du projet.
- **Développer juste** : éliminer tout travail inutile en se focalisant sur les besoins réels du client.

XP a fait ses preuves pour des équipes de moins de 10 développeurs réalisant un produit sur mesure pour un client suffisamment accessible.

La mise au point de XP dure depuis quelques années, mais XP a réellement démarré en octobre 1999 avec la parution du livre :

Extreme Programming Explained : Embrace Change, Kent Beck, Addison-Wesley, 1999.

D'autres livres consacrés à XP sont parus depuis, dont un en Français:

L'Extreme Programming (avec deux études de cas), Jean-Louis Bénard, Laurent Bossavit, Régis Medina, Dominic Williams, Eyrolles, 2002.

Par ailleurs, le phénomène continue à prendre de l'ampleur :

Créée en janvier 2000, la mailing list spécialisée compte en janvier 2003 près de 3800 abonnés. XP est défendu par de grands noms du développement objet, en particulier Kent Beck, Martin Fowler et Robert C. Martin.

XP est déjà mis en oeuvre dans des domaines aussi variés que la banque (Bayerische Landesbank, Credit Swiss Life, First Union National Bank), l'industrie automobile (DaimlerChrysler, Ford Motor Company), le transport (CSEE Transport), les télécoms (Nortel), le commerce électronique (Evant).
Dans la pratique, XP se définit à trois niveaux :

- Code : clair et simple grâce au remaniement et aux tests.
- Équipe : travail en commun sur toutes les parties de l'application.
- Gestion du projet : démarche itérative basée sur l'expérience acquise en cours de route.

Selon une étude en 1994 aux États-Unis sur un échantillon de 8 000 projets, seuls 16 % sont finalisés dans le temps et le budget initial. Pire, 32% sont interrompus en cours de route.

4 valeurs de la pratique XP et des agiles…

Communication	discussion, échange.
Simplicité	choisir le design le plus simple, les algorithmes et les techniques les plus simples (ne pas choisir XLST parce que c'est à la mode).
Feedback	livraison et tests fréquents donc retours immédiats.
Courage	jeter le code non performant et inefficace ou incompréhensible et le réécrire.

12 règles XP

1. Planification itérative	Plan, livraison, priorités, scénarios, durée Client assisté par les développeurs.
2. Petites livraisons fréquentes	Produire le plus de valeurs avec le moins d'efforts.
3. Design/conception simple	Garder le code simple, on n'essaie de résoudre des problèmes futurs.
4. Tester	C'est la clé. Tests automatisés et toujours avoir le courage de changer le code. Tests de recette et tests unitaires.
5. Intégration continue	Crucial. Automatiser la construction, la distribution et le déploiement. (ANT)
6. Remaniement permanent	Ajouter des fonctionnalités et rester simple. Améliorer, dégraisser ou ajouter pour un meilleur fonctionnement et/ou plus de simplicité et de clarté.
7. Binôme	Révolutionnaire. Toujours par paire. Gestionnaire sceptique sur les coûts? Double l'éventail des connaissances, permet le « refactoring » immédiat et la vérification du respect des normes. Équipe interchangeable.
8. Propriétaire collectif	Tous contribuent, le code appartient à tous. N'importe qui peut apporter un changement en ayant les tests appropriés.
9. 40 heures / semaine	Pas de temps supplémentaire. Si le travail ne peut être fait en 40 heures, il y a un problème. Trop d'erreurs surviennent par des programmeurs brûlés. Pas de longues réunions, présence aux heures du client.
10. Travail chez le client (client sur le site)	Plus pratique pour la consultation, le client voit rapidement les résultats et peut préciser ou modifier la stratégie de développement.

11. Métaphore	Langage commun et ensemble commun de termes utilisés pour visionner les fonctionnalités du projet. Client et développeurs parlent la même langue.
12. Normes de programmation	Lisibilité collective, utilisation juste des exceptions, « threads », construction correcte du langage, ne pas dupliquer la logique du code.

Scrum

Le processus Scrum repose sur 2 journaux ou "backlog" :

« backlog » de produit :

Une liste des fonctionnalités (spécification ou user story) pour le produit, définie par le directeur de produit (client), exigences, priorités.

	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

« backlog » de Sprint :

Recense les tâches du Sprint en cours (une itération) :

Tasks	Mon	Tues	Wed	Thurs	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	4	
Test the middle tier	8	16	16	11	8
Write online help	12				
Write the foo class	8	8	8	8	8
Add error logging			8	4	

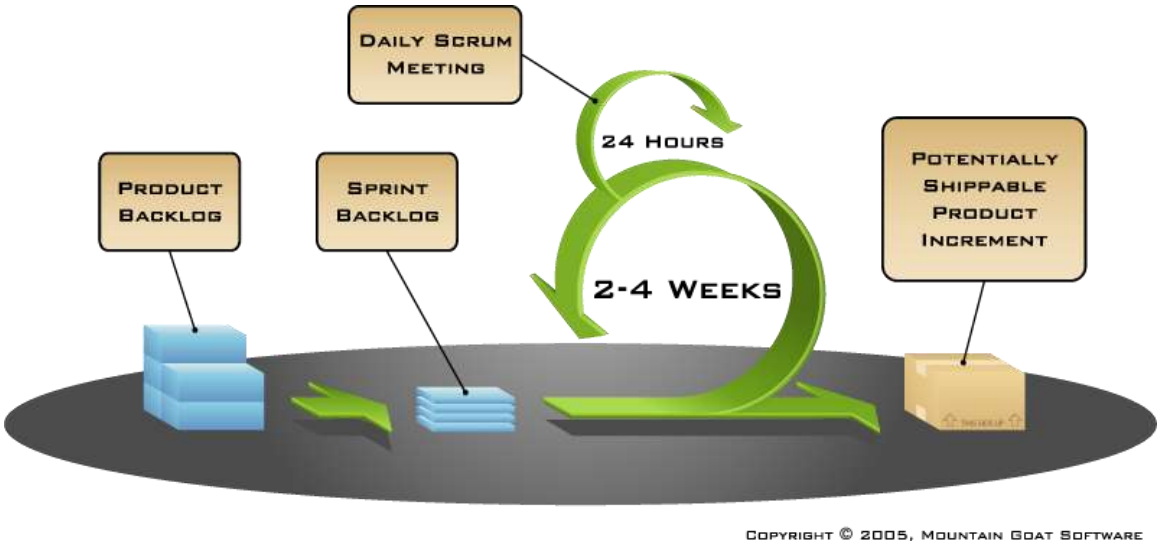
Un projet utilisant Scrum a son cycle de vie composé de Sprints successifs (itération).

Un Sprint dure au plus 4 semaines.

Pendant un Sprint, des réunions quotidiennes de moins de 15 minutes (appelées Daily Scrum) permettent à toute l'équipe de faire le point sur le travail accompli par chacun depuis la dernière réunion.

Pendant un Sprint l'équipe développe un produit partiel, Il s'y déroule (passe par) toutes les activités nécessaires pour cela : analyser, concevoir, développer, tester, documenter, intégrer.

Plusieurs utilisateurs : Microsoft, Google, BBC, Intuit, etc.



Sprints

- Les projets Scrum progressent par une série de sprints (itérations)
- Equivalents aux itérations XP
- La durée d'un sprint est de 2 à 4 semaines
- Une durée constante apporte un meilleur rythme
- Le produit (partiel) est conçu, codé et testé pendant le sprint
- Une version fonctionnelle est livrée à la fin du Sprint

Scrum quotidien (daily scrum)

- Tous les jours

- 15 minutes
- Debout
- Pas fait pour résoudre les problèmes
- Tout le monde est invité
- Seuls les membres de l'équipe peuvent parler
- Permet d'éviter l'organisation d'autres réunions

- Chacun répond à trois questions :
  1. Qu'as-tu fait hier?
  2. Que vas-tu faire aujourd'hui?
  3. Y a-t-il un truc qui cloche?

Mettre les réponses dans le journal de bord.

À regarder : Vidéo *: http://www.dailymotion.com/video/x56sex_planification-et-gestion-de-projet_tech?start=14#from=embediframe*

Mountain Goat Software, traduction de Claude Aubry , JDN Développeurs, Xavier Borderie, SCRUM – Design Pattern, Syracuse University

Gros plan sur les tests

Écrire les tests avant tout et les automatiser

XP définit deux types de tests

Les premiers sont les tests de **recette** ou tests fonctionnels qui sont rédigés par le client, à partir d'un langage formel si possible. Comme leur nom l'indique, ces tests (Acceptance Tests) servent à valider le travail du prestataire, comme on peut le voir dans les projets classiques avec un cahier de recette. Ainsi, il s'agit tout simplement de vérifier que tout fonctionne correctement et est conforme à la demande, avant de donner le dernier chèque... Mais, avec XP, le client définissant au fur et à mesure ses besoins aux développeurs, il n'y a généralement pas de grandes surprises lors de ces tests...

Le deuxième type de test que recense XP est le test **unitaire** (Unit Testing). On appelle

généralement test unitaire, tout test écrit par l'un des programmeurs dans le but de s'assurer du bon fonctionnement de son programme. Avec XP, c'est la même signification sauf qu'ils sont plus nombreux et plus fréquents. Ainsi, au contraire du test de recette, son but n'est pas de satisfaire le client directement mais permet de vérifier la qualité du développement. Ces tests sont, bien sûr, écrits comme les autres avant le codage (+ ou - TDD) et sont exécutés après chaque phase de codage.

Ils doivent tester généralement une itération et leur durée d'exécution est comprise entre 1 et 10 minutes. Au-delà, ils doivent être découpés.

Une modification du code (refactoring) ne devra en aucun cas faire échouer les tests qui garantissent donc, dès le début, la conformité du développement. Par contre, un changement dans le caractère fonctionnel du code, rappellera au développeur que ce changement est peut-être inutile car non demandé par le client.

Les tests montrent l'avancement

Les tests systématiques assurent que le système en cours de création est constamment opérationnel puisqu'un dysfonctionnement provoqué par une évolution du code source est tout de suite détecté et donc rapidement corrigé.

De ce fait, le client peut toujours connaître l'avancement exact du projet et surtout obtenir des livraisons régulièrement.

Aussi, la livraison, qui peut se faire donc dès la fin du développement, est donc celle d'un produit totalement fonctionnel et sans aucun bug, ouf...

Des tests optimaux pour une meilleure conception

On en arrive à se demander si ce n'est pas grâce aux tests que l'on arrive à programmer mieux et plus rapidement. Pour cause, on possède un garde-fou : si on fait quelque chose de faux, le filet de sécurité des tests intégrés au langage prévient qu'il y a un problème et montre même où il réside...

Les tests documentent la conception

L'écriture des tests conduit à décrire de façon formelle ce que fait chaque partie du code. Un test est une façon d'exprimer des contraintes telles que les données reçues en paramètres (chaîne avec des nombres traitée différemment, etc.). On peut aussi rapprocher le rôle des tests unitaires à des notions de « contrat » prônés par Bertrand Meyer et que nous verrons ultérieurement.

Quelques chiffres

Il n'est pas du tout exceptionnel, pour une application non triviale, d'écrire par mois et par programmeur une quantité de tests unitaires de l'ordre de la centaine.

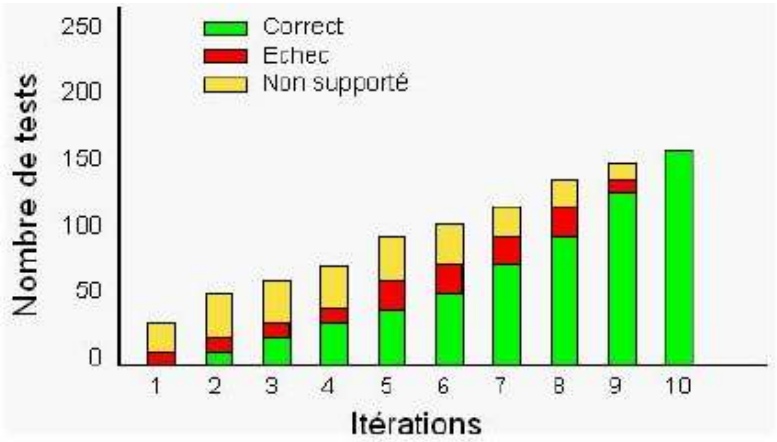
Le nombre de tests unitaires peut varier considérablement selon qu'ils regroupent plus ou moins d'assertions. Une « densité » typique semble être de 4 ou 5 assertions par test, mais on peut atteindre jusqu'à 20 assertions par test.

Dans le cadre d'une application nouvelle, la stricte application des principes du développement par les tests conduit assez naturellement à un volume de lignes de code équivalent entre code de tests et code principal.

L'évolution des tests est un indicateur concret de l'avancement des développements :

- Le nombre de tests écrits indique l'avancement des spécifications,
- Le nombre de tests validés indique l'avancement des développements,
- Le nombre de tests en échec indique le degré de qualité du logiciel.

Le chef de projet peut ainsi représenter de manière très synthétique l'évolution du projet :



Les méthodes agiles ont permis de développer :

- Patron de conception
- Revue de code ou « refactoring » (renommer, modifier, éliminer code mort, ajout d'assertions, retravailler le code source)
- Réorganisation de code (menu Source)
- Tests automatisés

En résumé

Les principes agiles :

- Priorité aux personnes et aux interactions plutôt que sur les processus et les outils
- Des applications fonctionnelles opérationnelles plutôt qu'une documentation pléthorique
- Collaboration avec le client plutôt que négocier un contrat
- Réactivité au changement plutôt que suivre un plan

Les mêmes valeurs que XP :

- Communication
- Simplicité
- Feedback
- Courage
- Humilité

UML et les méthodes agiles

XP

- XP n'interdit pas les documents, mais pose dessus un regard critique …
- XP n'interdit pas la modélisation, mais rappelle que seul un code opérationnel est garant de qualité…
- Si la création d'un modèle est une demande Client, alors il s'agit d'une « user story » à intégrer dans une ou plusieurs itérations, comme les autres …

Scrum

Scrum ne préconise aucune technique particulière d'ingénierie, mais :

- La modélisation agile est encouragée, au lieu d'une modélisation UML extensive
- Les modèles aident à comprendre, ils ne sont pas considérés comme une documentation obligatoire
- Documentez le produit quand il est stable…
- Si une documentation est requise, il vaut mieux la construire après que le code ait été testé (reverse engineering)

Certains experts Scrum préconisent :

- Un peu de modélisation au début de chaque itération pour mieux communiquer le but à tous les intervenants
- Un modèle métier lors de la réunion de planification du début améliore la discussion sur les « user stories »

Tous encouragent :

- Une modélisation participative
- Les modèles sont dessinés sur les murs (post-it géants, etc.), pendant des sessions de modélisation collaboratives.