CULTURE DU WEB

Méthodes et Agilité

Une approche agile des méthodes agiles

Smile
OPEN SOURCE SOLUTIONS

www.smile.fr • +33 (0)1 41 40 11 00 • contact@smile.fr
www.smile-oss.com • blog.smile.fr • twitter: #GroupeSmile

PREAMBULE

Smile

Smile est une société d'ingénieurs experts dans la mise en œuvre de solutions open source et l'intégration de systèmes appuyés sur l'open source. Smile est membre de l'APRIL, l'association pour la promotion et la défense du logiciel libre, de Alliance Libre, PLOSS, et PLOSS RA, des associations clusters régionaux d'entreprises du logiciel libre.

Smile compte 290 collaborateurs en France, 330 dans le monde, ce qui en fait la première société en France spécialisée dans l'open source.

Depuis 2000, environ, Smile mène une action active de veille technologique qui lui permet de découvrir les produits les plus prometteurs de l'open source, de les qualifier et de les évaluer, de manière à proposer à ses clients les produits les plus aboutis, les plus robustes et les plus pérennes.

Cette démarche a donné lieu à toute une gamme de livres blancs couvrant différents domaines d'application. La gestion de contenus (2004), les portails (2005), la business intelligence (2006), les frameworks PHP (2007), la virtualisation (2007), et la gestion électronique de documents (2008), ainsi que les PGIs/ERPs (2008). Parmi les ouvrages publiés en 2009, citons également « Les VPN open source », et « Firewall est Contrôle de flux open source », et « Middleware », dans le cadre de la collection « Système et Infrastructure ».

Chacun de ces ouvrages présente une sélection des meilleures solutions open source dans le domaine considéré, leurs qualités respectives, ainsi que des retours d'expérience opérationnels.

Au fur et à mesure que des solutions open source solides gagnent de nouveaux domaines, Smile sera présent pour proposer à ses clients d'en bénéficier sans risque. Smile apparaît dans le paysage informatique français comme le prestataire intégrateur de choix pour accompagner les plus grandes entreprises dans l'adoption des meilleures solutions open source.

Ces dernières années, Smile a également étendu la gamme des services proposés. Depuis 2005, un département consulting accompagne nos clients, tant dans les phases d'avant-projet, en recherche de solutions, qu'en accompagnement de projet. Depuis 2000, Smile dispose d'un studio graphique, devenu en 2007 Smile

Digital – agence interactive, proposant outre la création graphique, une expertise e -marketing, éditoriale et interfaces riches. Smile dispose aussi d'une agence spécialisée dans la TMA (support et l'exploitation des applications) et d'un centre de formation complet, Smile Training. **Enfin, Smile est implanté à Paris, Lille, Lyon, Grenoble, Nantes, Bordeaux, Poitiers, Aix-en-Provence et Montpellier. Et présent également en Espagne, en Suisse, au Benelux, en Ukraine et au Maroc.**

Quelques références de Smile

Intranets et Extranets

Société Générale - Caisse d'Épargne - Bureau Veritas - Commissariat à l'Energie Atomique - Visual - CIRAD - Camif - Lynxial - RATP - Sonacotra - Faceo - CNRS - AmecSpie - INRA - CTIFL - Château de Versailles - Banque PSA Finance - Groupe Moniteur - Vega Finance - Ministère de l'Environnement - Arjowiggins - JCDecaux - Ministère du Tourisme - DIREN PACA - SAS - CIDJ - Institut National de l'Audiovisuel - Cogedim - Diagnostica Stago Ecureuil Gestion - Prolea - IRP-Auto - Conseil Régional Ile de France - Verspieren - Conseil Général de la Côte d'Or - Ipsos - Bouygues Telecom - Prisma Presse - Zodiac - SANEF - ETS Europe - Conseil Régional d'Ile de France - AON Assurances & Courtage - IONIS - Structis (Bouygues Construction) - Degrémont Suez - GS1-France - DxO - Conseil Régional du Centre - Beauté Prestige International - HEC - Veolia

Internet, Portails et e-Commerce

Cadremploi.fr - chocolat.nestle.fr - creditlyonnais.fr - explorimmo.com - meilleurtaux.com - cogedim.fr - capem.fr - Editions-cigale.com - hotels-exclusive.com - souriau.com - pci.fr - odit-france.fr - dsv-cea.fr - egide.asso.fr - Osmoz.com - spie.fr - nec.fr - vizzavi.fr - sogeposte.fr - ecofi.fr - idtgv.com - metro.fr - stein-heurtey-services.fr - bipm.org - buitoni.fr - aviation-register.com - cci.fr - eaufrance.fr - schneider-electric.com - calypso.tm.fr - inra.fr - cnil.fr - longchamp.com - aesn.fr - bloom.com - Dassault Systemes 3ds.com - croix-rouge.fr - worldwatercouncil.org - Projectif - credit-cooperatif.fr - editionsbussiere.com - glamour.com - nmmedical.fr - medistore.fr - fratel.org - tiru.fr - faurecia.com - cidil.fr - prolea.fr - bsv-tourisme.fr - yves.rocher.fr - jcdecaux.com - cg21.fr - veristar.com - Voyages-sncf.com - prismapub.com - eurostar.com - nationalgeographic.fr - eau-seine-normandie.fr - ETS Europe - LPG Systèmes - cnous.fr - meddispar.com - Amnesty International - pompiers.fr - Femme Actuelle - Stanhome-Kiotis - Gîtes de France Bouygues Immobilier - GPdis - DeDietrich - OSEO - AEP - Lagardère Active Média - Comexpo - Reed Midem - UCCIFE - Pagesjaunes Annonces - 1001 listes - UDF - Air Pays de Loire - Jaccede.com - ECE Zodiac - Polytech Savoie - Institut Français du Pétrole - Jeulin - Atoobi.com - Notaires de France - Conseil Régional d'Ile-de-France - AMUE

Applications métier

Renault - Le Figaro - Sucden - Capri - Libération - Société Générale - Ministère de l'Emploi - CNOUS - Neopost - Industries - ARC - Laboratoires Merck - Egide - ATEL-Hotels - Exclusive Hotels - CFRT - Ministère du Tourisme - Groupe Moniteur - Verspieren - Caisse d'Epargne - AFNOR - Souriau - MTV - Capem - Institut Mutualiste Montsouris - Dassault Systèmes - Gaz de France - CAPRI Immobilier - Croix-Rouge Française - Groupama - Crédit Agricole - Groupe Accueil - Eurordis - CDC Arkhineo

Applications décisionnelles

IEDOM – Yves Rocher - Bureau Veritas - Mindscape – Horus Finance – Lafarge – Optimus – CecimObs – ETS Europe – Auchan Ukraine – CDiscount – Maison de la France – Skyrock – Institut National de l'Audiovisuel – Pierre Audouin Consultant – Armée de l'air – Jardiland – Saint-Gobain Recherche – Xinek – Projectif – Companeo – MeilleurMobile.com – CG72 – CoachClub

Introduction

Depuis quelques années, les méthodes Agiles sont en vogue. Elles semblent tout particulièrement répondre aux exigences de rapidité et de flexibilité des projets web.

Pourtant, une méthode Agile ne convient pas à tous les projets, à tous les contextes. Faut-il alors y renoncer totalement ? L'alternative est-elle entre Agilité et Lourdeur ? Doit-il y avoir opposition entre une démarche méthodique et être démarche Agile ? Ne peut-on pas avoir une approche elle-même flexible et adaptative vis-à-vis des méthodes Agiles ?

C'est ce que nous nous proposons d'analyser ici.

Nous n'avons pas tous la chance d'avoir un corps souple, et des élongations brutales ou excessives feraient courir à certains le risque d'un claquage. De la même manière, certaines organisations, certains projets, ne sont pas prêts pour des méthodes Agiles, et il ne serait pas bon de les déployer en force.

Chez Smile nous réalisons des projets dans des contextes variés, caractérisés par des exigences, des contraintes et des niveaux de risques potentiellement élevés.

Nous souhaitons mettre en œuvre des pratiques Agiles afin de donner à nos clients la meilleure visibilité quant à l'avancement de leur projet, tout en leur permettant si possible de réorienter leur stratégie, d'ajuster leur cahier des charges, de changer d'avis.

Mais nous devons également être en mesure de prendre des engagements contractuels forts dans le cadre de projets au forfait. Engagements de qualité, de coûts et de délais. Et nous devons également faire face à une diversité de situation, quant aux

C'est pourquoi nous avons défini une méthode particulière, Smile Foundation, qui est une mise en œuvre elle-même Agile d'une méthode Agile, c'est-à-dire qui vise à introduire dans chaque projet toute l'Agilité qui est possible, en sachant y intégrer toutefois les éléments de méthodologie plus structurés chaque fois qu'il est nécessaire.

C'est cette démarche que nous présentons ici. Pour l'étayer, nous rappellerons les grands principes des approches Agiles et leurs principales mises en œuvre, que nous comparerons aux méthodes plus traditionnelles. Nous expliquerons en quoi certaines

situations réelles courantes rendent difficile une démarche strictement Agile, et nous décrirons la manière d'y faire face.

Des entrainements standards, agiles et moins agiles.

Nous présentons d'abord précisément ce que la littérature et le web offrent comme principales approches agiles : Scrum, XP, lean. Puis d'autres pratiques complémentaires : CMMI, UP. Nous tenons là autant de bonnes pratiques visant à développer l'agilité ou à renforcer une organisation de projets.

Des standards au réel.

Nous exposons ensuite des cas réels, tels que nous en avons environ une centaine par an chez Smile. Il s'agit d'illustrer les obstacles in vivo aux approches agiles et les méthodes mises en œuvre, à partir des pratiques standards exposées précédemment.

Nous décrivons en quelque sorte le programme d'entraînements quotidiens, adapté à nos projets chez Smile.

Devenir adaptable.

Enfin, ayant présenté notre expérience, nous en venons à décrire comment la transmettre ? Car ce programme d'entrainement Smile, même s'il utilise des exercices issus de grands standards internationaux, est surtout adapté à notre profil. Comment se construire son entrainement personnalisé, c'est-à-dire ajusté à sa propre culture d'entreprise, à ses objectifs stratégiques? Selon quelles étapes ?

Sommaire

LES PRINCIPALES APPROCHES AGILES

Une méthode est une démarche structurée. C'est-à-dire une façon d'avancer par étapes bien identifiées a priori et qui s'articulent de façon ordonnée.

L'intérêt de la méthode est de prévoir, voire de garantir un résultat : si les étapes sont suivies dans l'ordre préconisé et sous les conditions requises, alors le livrable sera au rendez-vous.

L'agilité commence par se définir en opposition à la notion rigide de structure formelle. Son prédicat est que l'essentiel n'est pas de suivre des étapes. L'important est une bonne relation demandeur / fournisseur, pour livrer le produit le plus adapté. C'est une façon de remettre l'accent sur le véritable objectif – livrer le produit apportant de la valeur au demandeur - et de ne plus confondre la fin et les moyens.

L'agilité est ainsi définie par le Manifesto Agile (http://agilemanifesto.org) en 12 principes fondateurs :

1. Satisfaire le client, par des livraisons rapides
2. Accueillir les nouvelles demandes pour maintenir l'avantage métier
3. Livrer des versions fréquemment, toutes les deux semaines
4. Client et développeurs travaillent ensemble quotidiennement
5. Impliquer des acteurs motivés, leur donner les moyens de travailler et leur faire confiance
6. Le meilleur moyen de communiquer ? Le face à face
7. Des versions qui marchent sont les meilleures preuves d'avancement
8. Avoir un rythme de projet soutenable par tous : sponsor, client, développeurs
9. Continuellement être attentif à l'excellence technique et de conception
10. Viser la simplicité, c'est-à-dire minimiser l'effort
11. Les meilleures solutions naissent d'équipes autonomes

12. Régulièrement l'équipe réfléchit à s'améliorer et met en œuvre ses idées

Mais nous verrons dans la suite que l'agilité prend corps dans des méthodes. Nous en résumons les principales : *Scrum* et *Extreme Programming* (XP), apparues dans les années 90.

Lean, adapté au développement logiciel, n'est pas une méthode mais un ensemble de principes empiriquement identifiés et validés par le retour d'expérience. *Lean Software Development* promeut une approche à laquelle les méthodes agiles répondent.

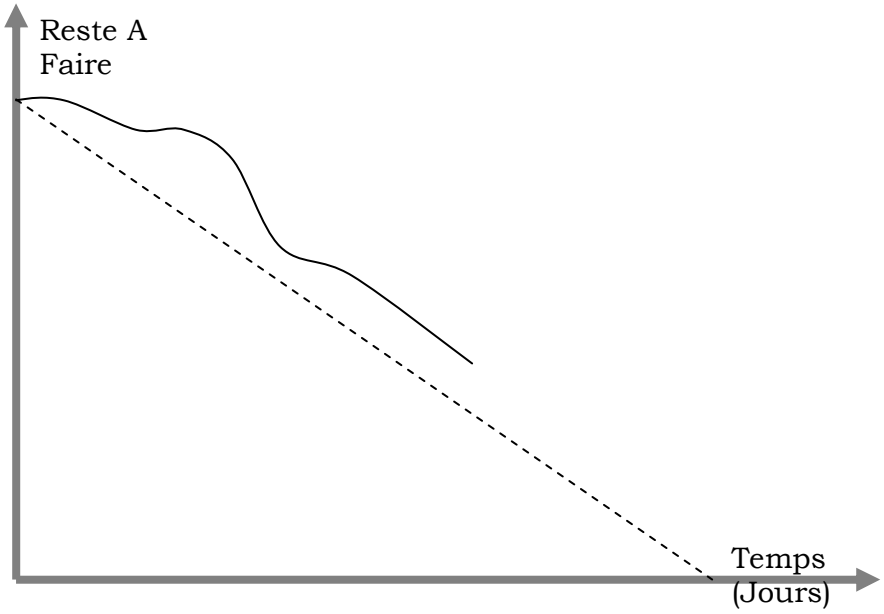
La « méthode », une démarche méthodologique, n'est donc pas incompatible avec l'agilité. Mais l'agilité en réoriente le sens en posant que le réel, le retour d'expérience, prime sur la théorie aussi élaborée soit-elle.

Scrum

C'est une des plus connues des méthodes agiles. Scrum n'est pas une abréviation, mais signifie mêlée (de Rugby). L'image illustre la forte cohésion de tous les acteurs que cette approche invoque.

De façon très synthétique, les étapes de la méthode sont les suivantes :

- Le ***Product Owner*** (l'interlocuteur client) établit une liste de besoins ou ***Product Backlog***. C'est un peu la grande « liste de courses » qui décrit le périmètre.
- Le ***Scrum Master*** (Chef de projet avec une forte inclination de facilitateur), le *Product Owner* et l'équipe priorisent les besoins au cours d'un *Planning Meeting* d'une journée. Le but est de distribuer les besoins dans des laps de temps successifs de 2 à 4 semaines. Ce sont les ***Sprints***. Les choses à faire pendant un *Sprint* sont listées dans la ***Sprint Backlog*** qui est un extrait affiné de la *Product Backlog*.
- En cours de *Sprint*, le *Scrum Master* maintient un indicateur, le ***burndown graph***, qui représente simplement le reste à faire par jour. Les données proviennent des tests : ce qui passe les tests est gagné. Sa charge estimée est déduite du montant de reste à faire global.



Exemple de burndown graph

Si la courbe est très au dessus de la ligne prévue, c'est peut être que l'équipe a du mal à réduire le reste à faire dans les temps.

- A l'issue de chaque *Sprint* (c'est-à-dire toutes les 2, 3 ou 4 semaines), l'équipe projet présente au *Product Owner* une version partielle du produit répondant à la *Sprint Backlog*. C'est la ***Sprint Review***. Le demandeur a donc l'occasion de constater l'avancement réel du produit et de faire des retours.
- Egalement à l'issue du *Sprint*, l'équipe et le *Scrum Master* font une ***Sprint Retrospective*** : un bilan de ce qui a fonctionné et de ce qui doit être amélioré dès le sprint suivant. Le but est de créer des actions concrètes d'améliorations. Ces actions sont insérées dans la *Sprint Backlog* suivante.
- Et le plus connu : le ***Daily Meeting*** qui a lieu tous les matins sur une durée verrouillée de 15 minutes. Ce point d'équipe inclut également le *Product Owner*, voire d'autres interlocuteurs client. Mais ceux-ci ne peuvent s'y exprimer, seulement observer. Chaque membre d'équipe par contre indique à tous : ce qu'il a terminé

dans la *backlog*, ce qu'il va démarrer et ce qui le bloque afin que le *Scrum Master* ou un autre équipier lui apporte assistance.

Ce dernier point, le *Daily Scrum Meeting*, est emblématique de Scrum. C'est précisément la mêlée où tous les acteurs sont impliqués, selon leur rôle.

La première idée force de cette méthode est de **s'appuyer sur des personnes** qui sont très impliquées et responsables : le *Product Owner* et chaque membre d'équipe.

Nul besoin de reporting. Le projet est très transparent grâce aux *Daily Meetings*.

La seconde idée force est le **rythme de livraison soutenu** (et soutenable). Comme Saint-Thomas, le client est invité à ne croire du produit que ce qu'il en voit. S'il ne peut le toucher de ses mains, alors il ne pourra croire que celui-ci a émergé du monde de l'abstrait.

Extreme Programming (XP)

Egalement très connue, la méthode XP partage beaucoup d'étapes avec Scrum. Mais elle insiste beaucoup plus sur l'ingénierie, les bonnes pratiques de programmation.

- Tout d'abord, XP fait l'hypothèse que l'équipe projet est rassemblée dans un ***open space***, sur un même site. L'équipe embarque avec elle chaque jour le représentant client sur son site. Cette proximité permet de mettre en œuvre des pratiques de communication très concrètes (tableau blanc visible de tous à tout instant) et très directe.
-
- Une première phase d'**exploration** courte (de l'ordre du mois) permet au client de fournir une liste de ***User Story***. Ce sont de courtes descriptions des demandes client. Assez courte pour tenir sur un post-it. Cette première phase donnera naissance au tout premier cycle de développement et à la livraison d'une première version partielle.
- La suite du projet est découpée en **itérations**, périodes verrouillées à 1 à 2 semaines. On retrouve ici la notion de *Sprint* de Scrum. Le premier jour de chaque itération

est consacré au ***Planning Game***. Comme dans Scrum, le client et l'équipe identifient par priorités les *User Story* à développer sur l'itération. L'équipe donne son estimation et s'assure que le choix effectué est réalisable dans la durée.

- L'étape suivante est l'organisation des travaux par l'équipe pendant l'itération. Au cours d'un ***Stand Up meeting*** quotidien fixé à 15mn, les membres d'équipe vont s'apparier en **binômes** et se choisir les *User Story* à prendre en charge. Les *User Story* sont matérialisées par des post-it collés sur un tableau blanc. Les post-it passent de la colonne *To do* vers la colonne *In Progress*. Ceux terminés (testés avec succès) sont dans la colonne *Done*. L'hypothèse d'une équipe rassemblée sur un site unique et ouvert prend ici son sens. D'autant que les binômes se redéfinissent chaque jour, de façon à ce que chacun travaille avec tous.
- Chaque binôme va tout d'abord coder les **tests unitaires** des fonctions de la *User Story*. Puis les fonctions elles mêmes. L'un code, tandis que son binôme vérifie en continu la qualité de son code : respect des **normes de codages**, absence d'erreur, etc. Les rôles s'échangent pendant la journée. L'idée est qu'en exécutant le test unitaire il commence par être en échec, puis le code s'étoffant, il arrive à passer. Que les binômes changent chaque jour amène naturellement à ce que tous les membres d'équipe interviennent sur toutes les parties du logiciel. On parle à ce propos de ***Pair Programming***.
- **De manière continue** (plusieurs fois par jour), les codes de tous les binômes sont **intégrés** (*commit* et *build*). Les tests unitaires sont passés. Continument également, le binôme refactorise son code au maximum, c'est-à-dire réutilise le code et supprime les redondances.
- En fin de *Sprint*, la version partielle du produit est livrée au client. Cette version est stable, elle a passé de multiples fois tous les tests unitaires avec succès, a subie sur toutes ses parties des « *4 eyes checks* ». Le client passe alors ses propres tests, fonctionnels. Ces cas de tests détaillent les *User Story* en des exemples concrets et réels.

On l'a compris, XP et Scrum se volent l'un l'autre les bonnes idées. Aux idées forces que nous citions sur *Scrum*, implication des

personnes et livraisons fréquentes, **XP ajoute celle d'un code continument vérifié**.

Lean Software Development

Lean est un paradigme issu de l'expérience de Toshiba pour optimiser ces processus de fabrication. Il a été adapté aux processus de création logicielle.

Il en ressort 7 principes qui ne constituent pas une méthode, mais auxquels Scrum et XP répondent.

1. Eliminer le gaspillage. C'est le credo numéro 1 des méthodes agiles. Tout ce qui ne produit pas de valeur pour le client (le demandeur) est à éliminer.

Ainsi toutes les tâches habituellement requises par une méthode lourde ou par l'habitude : modèles de spécifications détaillés que personne ne lit, reporting que personne ne consulte, étape obligée dont on a oublié l'objectif. Comme me le disait un jour un client : « je ne suis pas payé pour faire des spécifications. » Et de fait, son patron attendait de lui une application qui tourne.

Gaspillage également les fonctions décrites et développées par l'équipe, mais non demandées, non testées par le demandeur et non utilisées au final.

➔ La priorisation des demandes sur laquelle Scrum et XP insistent permet déjà de ne pas se perdre en détails sur des choses qui risquent d'être abandonnées à terme.

Energie perdue dans le fonctionnement même de l'équipe, lorsque certains se dispersent sur plusieurs sujets en parallèle. Il y a un « effet Joule », de la dissipation d'énergie, dans l'effort à se remettre sur un sujet en sortant d'un autre. Temps perdu à attendre les retours de tel ou tel sur un document ou sur une question…Temps et énergie perdus dans les transitions entre rôles sur un projet lorsque ceux-ci sont très éclatés entre différentes personnes : de l'analyste vers le développeur, vers le testeur, etc.

➔ Rassembler l'équipe, de taille réduite, sur un site commun vise à ne plus disperser la communication et les échanges dans du texte, des emails à répétition, etc.

Perte inutile enfin dans la production et la réparation de défauts. Le coût inutile lié aux corrections étant d'autant plus élevé que le défaut est détecté tardivement.

➔ L'intégration continue, c'est-à-dire un *build* toutes les heures, voire toutes les demi-heures, les tests unitaires automatisés d'XP ou l'accent mis par Scrum sur des tests à chaque *sprint* cherchent et traquent le défaut au plus tôt. Et le suppriment dans la foulée. Dans une approche « Zéro tolérance » du bug.

2. Maximiser l'apprentissage. L'équipe projet doit être apprenante, pour mieux réagir aux imprévus, mieux identifier les demandes à haute valeur. Abandonner vite les mauvais réflexes, comme répondre immédiatement à un email toutes affaires cessantes…

Il s'agit de multiplier les occasions de retours (*feedbacks*) sur ce qui est développé.

Cela passe par des techniques telles que des revues fréquentes et régulières de ce qui est produit par le demandeur. Y compris sur les produits intermédiaires qui prendront rapidement la forme de maquettes ou de prototypes.

L'autre solution est bien sûr le test de ce qui est créé ou plus généralement la vérification. Ainsi mettra-t-on en œuvre des revues par les pairs (institutionnalisées en pratique de base par XP), des tests unitaires que l'on voudra automatiser, des tests fonctionnels fréquents de versions partielles…

3. Fixer la décision le plus tard possible. Vouloir être prêt trop tôt ou obtenir une certitude précoce fixe la solution alors que les options sont encore nombreuses et les informations fluctuantes.

L'idée est de favoriser une approche ouverte à toutes les options tant que celles-ci sont possibles.

Deux solutions sont proposées par les méthodes agiles :

➔ Prototyper, essayer différentes possibilités, sans aller trop loin dans le détail (de la spécification, du code).

➔ Maintenir une capacité de réaction forte en livrant fréquemment.

Les deux approches se ressemblent. Un tâtonnement large qui sollicite des validations rapides (go / no go) pour identifier le chemin optimum. Empirique, pragmatique, pas théorique.

4. Livrer au plus vite. Ce principe fait écho aux précédents. Il s'agit de rechercher une validation du chemin emprunté avant de s'y enterrer. Il s'agit aussi de multiplier les embranchements : à chaque livraison, à chaque démonstration, de nouvelles options apparaissent.

Les méthodes agiles sont complètement dans cette approche, à l'extrême même avec XP, qui recommande des itérations d'une semaine. L'agilité est créée par la multiplication des articulations que sont les *Sprint Reviews*.

Attention bien sûr à la faisabilité de la chose. XP comme Scrum plaident pour des rythmes de livraisons soutenables par l'équipe comme par le client.

5. Responsabiliser l'équipe. L'expérience Lean montre qu'il faut faire confiance aux acteurs du projet pour profiter au maximum de leurs talents.

Ou autrement dit : sans leur implication motivée, le projet piétine. Et pour les motiver, il faut leur ménager une marge de décision.

Dans cette optique, on évitera la chaîne de commandement *top / down*. Au contraire, le manager sollicite le collaborateur pour qu'il lui expose sa vision du chemin à suivre. Etant entendu que l'objectif est donné par le manager.

Chacun est ainsi responsabilisé et motivé à apporter des solutions. Plutôt que de demeurer passivement dans l'attente de l'ordre - voire du contre-ordre - ou de développer des tactiques d'inertie.

L'exercice du bilan d'itération, au cours de duquel chacun est invité à identifier ce qui marche bien et comment corriger ce qui marche mal, permet de cadrer l'équipe.

Le management confronte l'équipe à son résultat. Pas de menace, pas de sanction ici. Mais le souci, encore une fois, de profiter du talent de chacun pour qu'il énonce lui-même la correction utile.

Des événements permettant de célébrer les succès, les résultats positifs, viennent compléter l'approche.

6. Construire la cohérence du système. Ce principe se reflète dans les pratiques agiles des tests continus, des revues de pairs, de *refactoring*. Nous citons là particulièrement XP, qui descend davantage dans l'intimité du système produit, dans ses modes de production.

Dans le développement classique, surtout lorsqu'il répond à l'urgence, la cohérence des strates de codes ajoutées s'efface peu à peu. Et le système se fragilise de plus en plus.

Le gaspillage nait de cette entropie croissante et les coûts de réparations deviennent indécents. La démotivation de l'équipe suit, avec l'impression pénible de maintenir à flot une passoire.

Le *refactoring* quotidien du code, amélioré par les revues croisées de binômes renouvelés, empêche l'accumulation de redondances, de codes morts et autres nids de contradictions à terme.

7. Voir global. Il y a deux manières de comprendre ce principe. Tout d'abord, si une optimisation est faite dans un coin du code, elle doit y être diffusée, standardisée.

Cette action rejoint le principe précédent de cohésion du système. Mais va plus loin, considérant que l'amélioration d'un point particulier ne doit pas s'arrêter à sa particularité. Ce serait prendre le risque de traiter le symptôme et d'ignorer la cause. Il faut chercher les liens, les raisons éventuelles derrière les circonstances pour rejoindre le cas général à traiter. En améliorant bien plus que localement on s'assure aussi de ne pas accumuler les rustines, génératrices de chaos.

L'autre interprétation est de s'astreindre à agréger l'information. Souvent, en cherchant à mesurer ou à décrire une situation, nous cherchons à être exhaustifs.

Ce faisant se multiplient les mesures ou les contrôles sur des détails. Le coût augmente. L'information récupérée est faible.

Un exemple : nous cherchons à réduire le nombre de bugs dans une livraison. Nous souhaitons donc mesurer et contrôler ce nombre de bugs. Souvent, en première approche cette mesure se décompose en décomptes de bugs par origines (client, membre d'équipe), par catégories du bug, par priorités…Le décompte devient complexe. Il est par ailleurs peu fiable : la catégorisation ou l'origine d'un bug n'obéissant pas toujours aux règles.

Mais à quoi servent tous ces décomptes ? Encore une fois, le but est de réduire le nombre de bugs.

Si par la suite nous souhaitons pister un type précis d'anomalies, alors un indicateur pourra peut-être être suivi un temps. Mais pas forcément de façon continue ou systématique.

D'AUTRES APPROCHES

Nous venons de balayer les principales approches agiles. Et nous avons vu qu'apparaissent en filigrane certaines conditions. Un peu comme un programme d'assouplissement corporel intensif présuppose que votre corps dispose d'un minimum de flexibilité.

Arrivez-vous à toucher vos orteils sans plier les genoux ? Pouvez-vous réunir en un même lieu, en un même temps et en une même action (à la manière d'une pièce classique) interlocuteurs clients et professionnels expérimentés ?

Si ce n'est pas le cas, ne désespérez pas. L'agilité et ses méthodes sont tout de même accessibles. Mais leurs présupposés non réunis devront être compensés par d'autres outils.

Les limites de l'agile

Le but de cette section n'est pas de faire le grognon face à l'agile, pour mieux s'agripper à ses bons vieux rituels.

Il s'agit d'être lucide, conscient des conditions d'application des approches agiles. Comme nous l'indiquions à l'instant, vous pouvez être confronté à des contraintes qui empêchent partiellement ou totalement d'être agile.

Il faut alors suivre d'autres approches qui permettent de survivre sous de telles contraintes, tout en apportant un maximum de souplesse. Ce sont celles-ci que nous présentons maintenant.

Mais auparavant, voyons quelles seraient ces limites ou ces contraintes qui nous empêcheraient de nous déclarer agile comme bon nous semble.

Pour commencer, comment gérer un périmètre flexible, ouvert aux nouvelles demandes, lorsqu'on est sous contrat au forfait ?

Le contrat au forfait fixe un périmètre, un budget et un délai. Ses modifications font l'objet d'avenants. Ce n'est pas souple. Et

surtout, c'est en contradiction avec l'intention du client qui est de fixer les limites au départ pour éviter les surprises dans la suite.

Certains proposent des « contrats agiles » qui prévoient des conditions souples de réagencement ou d'extension du périmètre. Et en effet, il est concevable que se basant sur un barème d'unités d'œuvre ou de points de complexités bien définis, client et fournisseur redéfinissent le périmètre par échanges de fonctionnalités équivalentes dans ce barème.

Mais on entre là sur un territoire moins agile, de définition précise des unités d'œuvre et d'évaluation de leur taille. On se rapproche déjà plus des préconisations classiques présentées dans la suite.

De plus, que faire si notre client ne peut, pour des raisons légales, se départir du très classique forfait ? Qui plus est, sans avenants possibles, comme c'est le cas pour certains marchés publics par exemple.

Comment par ailleurs vivre agile avec un client peu disponible ? Supposons deux brillants ingénieurs qui montent une *start up* et vous confient la création de leur site web. Ils sont naturellement les interlocuteurs et décideurs du projet de site. Mais il y a fort à parier que leurs journées de 24h se passent en *business développements* ou en recherche de financements. Certainement les *daily stand up meeting* auront toute leur sympathie, mais pas leur présence. Et les *sprint reviews* ? Comment démontrer une version intermédiaire à des absents ?

Autre contrainte, le projet doit être lancé en août. Vos ressources sont éparpillées sur les plages de sable blanc. Vous devez monter une équipe compacte, compétente et mature dans les 2 jours. Vous allez peut être plutôt créer une équipe multi-sites, faite des meilleures compétences qui ne sont pas forcément réunies au même endroit. Pour compléter ce noyau compétent et senior, vous allez adjoindre quelques jeunes embauchés, motivés mais pas matures. C'est là une formidable occasion de montée en compétence pour cette équipe et il faut bien une première fois. Mais entre la distance des sites et le nécessaire suivi des jeunes, il sera peut plus dur d'organiser du *pair programming* et de laisser en confiance l'équipe s'auto-organiser.

Dernier cas que nous soumettons à la réflexion, afin de ne pas casser l'engouement, comment être complètement agile lorsque votre interlocuteur client n'est pas dans cette culture là ?

Pour certains, la communication passe par un reporting précis et formel. Assister en observateur à une réunion quotidienne et matinale où des développeurs listent le détail des tâches ne leur convient pas. Ces interlocuteurs souhaitent un résumé concis des difficultés, être alerté. Sans quoi, pas de nouvelles, bonnes nouvelles. Et que faire également lorsqu'en début de *sprint* l'interlocuteur vous dépêche un représentant qui débute sur les questions fonctionnelles (« non sachant »)? Que vaudront les choix ?

Nous voyons que le niveau de risque à faire de l'agile tout seul, sans le client (Maitrise d'Ouvrage interne ou externe), est intolérable. D'ailleurs, la formulation « être agile sans le client » est intrinsèquement absurde.

Voyons donc ces autres approches qui permettent de traiter ces contraintes, en réduisant l'agilité mais sans nécessairement y renoncer.

CMMI

Le CMMI, *Capability Maturity Model Integration*, édité par le SEI (Software Engineering Institute, http://www.sei.cmu.edu/) apporte une batterie de pratiques qui ont pour but de garantir un résultat.

Le résultat en question est un produit, développé sous contraintes d'engagements (forfait ou TMA avec niveaux de services définis).

Le modèle CMMI est apparu dans les années 80. A l'époque il s'appelait CMM et distinguait les pratiques adaptées au développement de produit logiciel (« Software CMM ») de celles de systèmes électroniques ou mécaniques. Dans la réalité des projets de la défense américaine et de l'industrie, systèmes et logiciels s'intègrent et sont développés simultanément pour répondre à des exigences communes. CMMI en apporte une modélisation commune.

Le CMMI est d'abord un outil de mesure comparative, de benchmark. L'ensemble des pratiques qui y sont décrites fonctionne comme une sorte de cahier des charges plus ou moins satisfait par les fournisseurs logiciels. Ceci est évalué par des niveaux notés de 1

à 5 ou de 2 à 5, selon que la représentation du modèle est continue ou étagée.

Ce n'est pas cette utilisation qui nous intéresse ici. Ce que nous souhaitons, c'est gagner en flexibilité, tout en étant capables de vivre avec nos contraintes et sans perdre en sécurité.

Si nous reprenons la comparaison avec la souplesse du corps, nous voulons pouvoir se rapprocher du grand écart, sans risquer le claquage.

CMMI et agilité

Or il se trouve que pour établir les critères comparatifs du CMMI, le SEI a récolté, synthétisé et décrit les meilleures pratiques observées depuis les années 80 sur différents projets des industriels américains, mais aussi européens ou asiatiques.

Nous y reconnaissons l'approche empirique et pragmatique évoquée avec Lean. Car tout comme Lean, CMMI n'est pas une méthode mais trouvera sa réalisation dans différentes méthodes.

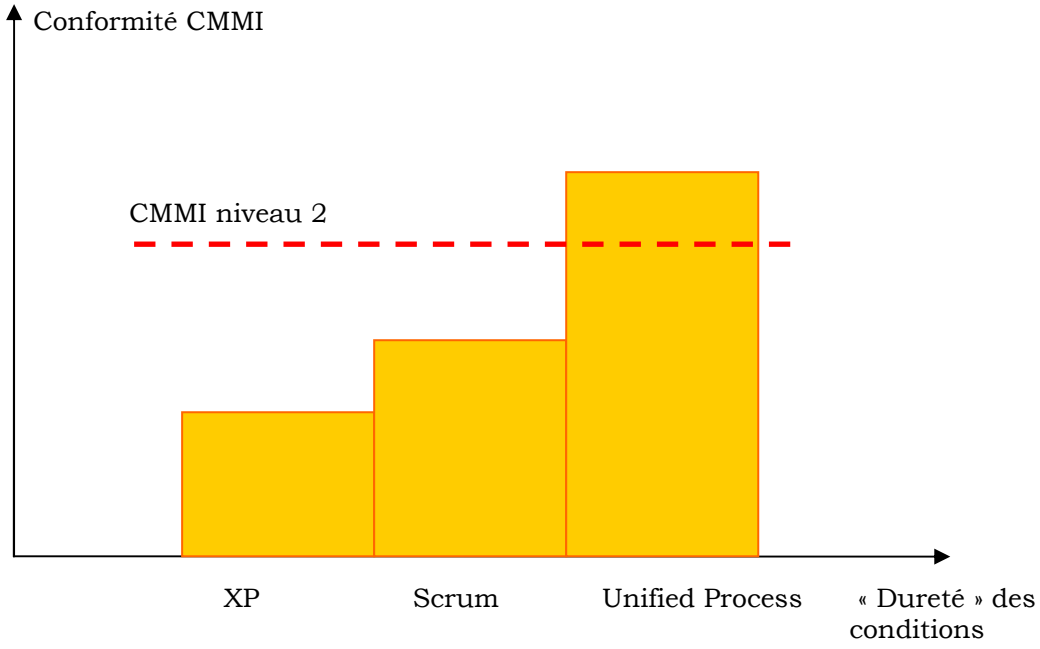
Bien sûr, l'opposition agilité / méthode devient parfois une opposition Lean / CMMI. Débats d'experts qui tout pragmatiques qu'ils soient et pétris de retours d'expériences, aboutissent à des croyances ennemies.

C'est que les expériences ne doivent pas être les mêmes…Et les croyances peut être pas si antinomiques, même s'il faut bien distinguer celui qui croit qu'un protocole bien étalonné peut garantir un résultat (CMMI), de celui qui ne croit qu'à l'expérience immédiate (Lean).

C'est là que CMMI répond à notre objectif. Il répond à des expériences différentes, peut être plus contraintes et plus risquées. A des conditions moins idéales, car précisément moins enclines à fournir une information immédiate. Ainsi les réponses de CMMI, et certaines de ses pratiques, pourront utilement compléter nos choix agiles.

Enfin, comme nous le disions plus haut, CMMI n'est pas une méthode mais s'implémente dans différentes méthodes qui répondent à ses exigences.

Une méthode agile ou itérative pourra répondre à CMMI de façon plus ou moins complète : de Scrum et XP, partiellement, à Unified Process qui est plus itératif qu'agile et que nous présentons plus loin.



Ce graphe illustre que plus les engagements et les conditions sont « durs », dans les deux sens de risqué et de rigide (peu favorable à l'agilité), plus il devient utile de piocher des solutions dans le CMMI.

Mais comprenons nous bien : nous ne prétendons pas à un « CMMI agile » . Il est clair qu'en se conformant point par point aux exigences CMMI on perd en agilité, dans le sens que Lean, XP et *Scrum* défendent. C'est le sens de la barre « CMMI niveau 2 » que seul Unified Process atteint. Et UP n'est plus très agile…

En fait, le niveau 2 – et a fortiori les niveaux supérieurs – de CMMI ne nous intéressent pas ici. Ce qui nous intéresse, c'est d'être agile au sens de *très proche de la demande réelle*, tout en compensant les lourdeurs d'environnement par des leviers issus du CMMI.

Structure et contenu de CMMI

Le modèle est consultable librement sur le site du SEI. Nous en résumons les points qui nous paraissent important pour notre sujet (d'après le modèle CMMI-DEV v1.2).

Le modèle *CMMI for development* (CMMI-DEV) est composé de 22 Process Area (PA). Un PA est en quelque sorte un type d'activité qui regroupe des bonnes pratiques. Par exemple la planification du projet, le suivi du projet, la gestion du fournisseur, etc.

Les PA sont eux-mêmes regroupés en catégories :

- Pratiques de gestion de processus
- Pratiques de gestion de projet
- Pratiques d'ingénierie
- Pratiques support

Afin de lever un peu le voile sur les termes techniques du modèle, nous nous essayons à donner pour chaque catégorie un très bref descriptif des activités couvertes.

Disons à l'avance – afin de distinguer un peu mieux ces notions de processus, types de processus (PA) et pratiques – que :

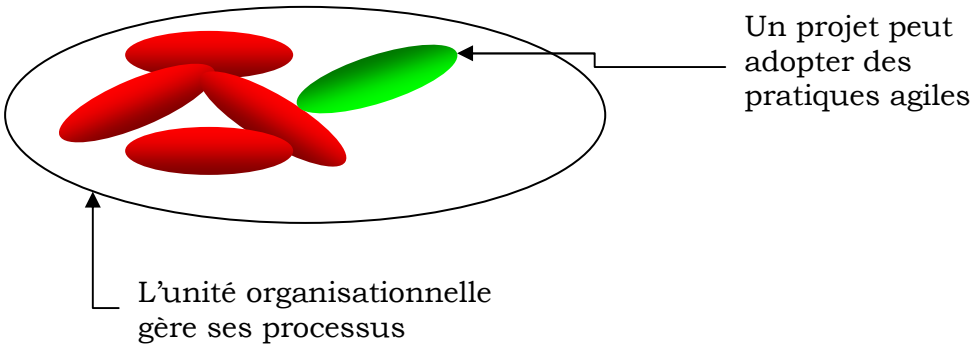
- Une **pratique** peut être vue comme une préconisation du modèle. Le descriptif de la pratique annonce ce qu'il faut pouvoir constater (exemple : client et équipe projet s'accordent sur les exigences), mais ne donne pas d'étapes ou de solutions concrètes pour y arriver.
- Le **PA** regroupe les pratiques – ou préconisations – par types. Chaque PA est caractérisé par des objectifs (***Specific Goals***) c'est-à-dire l'annonce de résultats qui doivent être observés si les préconisations relatives sont mises en œuvre.
- Un **processus** est une façon précise de mettre en œuvre les préconisations du modèle dans un contexte donné. Ainsi, plusieurs processus peuvent répondre à un même PA, chacun caractérisé par l'identité ou les choix de l'entreprise. Basé ou non sur un outil intégré et partagé, sur des documents bureautiques, sur tel ou tel type de rôles, etc.

Les PA de gestion de processus :

- **Organizational Process Focus (OPF)**. Ce qui permet au sein d'une organisation (par exemple une business unit) de repérer les façons de faire les plus adaptées du moment.
- **Organizational Process Definition (OPD)**. Ce sont les activités de formalisation de processus adaptées à l'organisation. (D'après les retours de OPF).
- **Organizational Training (OT)**. Organisation et entretien des programmes de formations.
- **Organizational Process Performance (OPP)**. Ce qui a pour but de déterminer et de suivre des objectifs quantitatifs pour certains processus critiques. (Un processus correctement suivi doit garantir un résultat quantitativement établi.)
- **Organizational Innovation and Deployment (OID)**. Ce sont les pratiques favorisant et facilitant l'introduction de nouveautés significatives dans l'organisation, par exemple une nouvelle technologie.

Nous sommes avec cette première catégorie d'activités en dehors du périmètre de jeu de l'agile. Ces préconisations concernent l'organisation qui fait naitre et aboutir plusieurs projets, alors que l'agile met le focus sur le projet lui-même.

Ce n'est pas contradictoire : les pratiques agiles choisies par un projet précis peuvent parfaitement répondre aux consignes dont s'est dotée l'organisation.



Périmètre CMMI (l'organisation) et Agile (le projet)

Les PA de gestion de projet :

- **Project Planning (PP)**. Les préconisations de planifications.
- **Project Monitoring and Control (PMC)**. Celles de suivi du projet, par rapport à la planification.
- **Supplier Agreement Management (SAM)**. Les pratiques de sélection, de contractualisation et de suivi du fournisseur.
- **Integrated Project Management (IPM)**. Au sein d'une organisation déjà dotée de ses propres processus, il s'agit là d'ajuster ces processus au contexte d'un projet et d'en déduire la gestion à mettre en œuvre.
- **Risk Management (RSKM)**. Identification, analyse et suivi des risques et des actions associées.
- **Quantitative Project Management (QPM)**. Comme nous avons un Organizational Process Performance pour mesurer l'efficacité des processus, voici le pendant pour les projets. Il s'agit là de pratiques ayant pour but de donner et de suivre les objectifs quantitatifs d'un projet.

Cette catégorie de préconisations s'intéresse au projet proprement dit mais aussi à son intégration dans une organisation.
L'organisation apprend de ses projets (c'est le propos de OPF évoqué plus haut) et un projet démarre à partir du capital de connaissance de l'organisation qui l'aide à se mettre en place vite et efficacement.

Les PA d'ingénierie :

- **Requirements Development (RD)**. Les exigences concernant la façon de décrire les demandes client.
- **Requirements Management (REQM)**. Les préconisations relatives à la gestion des demandes client : collecte, enregistrement, modifications.
- **Technical Solution (TS)**. Les activités concernant la conception de la solution répondant aux demandes. Les pratiques d'architecture.
- **Product Integration (PI)**. Les activités de développement et d'intégration, conformément à la description des demandes et à l'architecture choisie.
- **Verification (VER)**. Ce sont les pratiques de tests, au sens large de vérification ce qui inclut également les

revues par les pairs, à réaliser par l'équipe projet. Le but à atteindre est de savoir démontrer que la livraison répond à la demande.

- **Validation (VAL)**. Le but des pratiques de validation est de permettre au demandeur de s'assurer que la livraison couvre son besoin réel. Il s'agit de traiter l'écart demande / besoin.

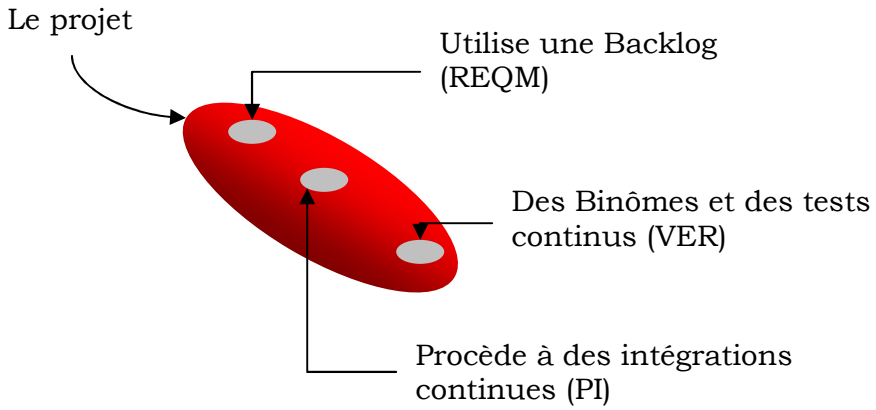
Nous arrivons là avec les PA d'ingénierie sur des groupes de préconisations que des méthodes comme Scrum et XP peuvent réaliser avec force.

Par exemple, les revues par les pairs de VER sont très présentes dans XP et son approche par binômes.

De même, l'intégration de versions rejoint les préoccupations de PI.

De façon originale et importante : l'implication centrale du *Product Owner* de Scrum et les *Sprints Reviews* (démos) fréquentes répondent à VAL sur toute la durée du projet. Et plus profondément que les classiques Vérification d'Aptitude au Bon Fonctionnement (VABF) et Vérification de Service Régulier (VSR), car en continu et précocement.

Et encore : la gestion des demandes de REQM est matérialisée de façon simple par les *backlog* de Scrum et le *Planning Meeting* de début de Sprint.



Le CMMI à l'intérieur d'un projet rejoint l'agile

Les PA de support :

- **Configuration Management (CM)**. Les pratiques de gestion de configurations, identification et description des versions, report des changements, etc. Particulièrement important dans un rythme de livraisons soutenu.
- **Process and Product Quality Assurance (PPQA)**. La vérification objective et l'aide à l'application des processus et à la qualification des poduits.
- **Measurement and Analysis (MA)**. Les préconisations de bases qui définissent ce que le modèle entend par 'mesures'. Qu'est ce qu'une mesure ou indicateur ? Comment le définir complètement ?
- **Decision Analysis and Resolution (DAR)**. CMMI préconise une analyse rationnelle de situation pour décider entre plusieurs options.
- **Causal Analysis and Resolution (CAR)**. Les préconisations d'identification et de plan d'actions pour améliorer plus fondamentalement les processus de l'organisation.

Unified Process (UP), Agile UP, Open UP

Unified Process (UP) ou 'processus unifié' est une méthode de projet itérative et incrémentale. Elle définit un ensemble complet d'étapes, de livrables et de rôles. La méthode est basée sur les cas d'utilisations (Use Cases) et sur UML (Unified Modeling Language).

Itérative et incrémentale, donc agile ? Pas tout à fait. C'est pourquoi nous ne l'avons pas présenté dans la première partie. UP ajoute aux méthodes Scrum et XP un formalisme précis, des modèles documentaires et promeut un outillage significatif.

Toutes choses que l'agilité contourne.

Car le détail scrupuleux du processus – documents du processus, chapitres dans les documents, schémas dans les chapitres - suggère que de son application rigoureuse découlera le résultat souhaité. Ce qui n'est pas le credo agile, lequel ne veut qu'interroger le réel et le retour utilisateur pour valider son produit.

Toutefois UP est adaptatif en ce sens qu'il y est explicitement demandé de l'ajuster au contexte. Selon les difficultés à adresser, l'équipe projet devra sélectionner les étapes et documents à faire ou non. Ainsi, si le contexte s'y prête, UP peut devenir agile.

Ce sont les déclinaisons AUP (Agile UP) et Open UP. Nous présentons d'abord UP standard, puis ces déclinaisons.

Pratiques clés d'UP

UP promeut les pratiques clés suivantes :

- **Itératif et incrémental, piloté par les risques**. Comme les sprints de Scrum, UP rythme le projet sur des itérations successives de 2 à 6 semaines. Chacune a pour but de livrer une version intermédiaire du produit : un incrément. Cette pratique fait d'UP un cousin très proche des méthodes agiles. L'identification et l'évaluation des risques est fondamentale dans UP. De cette analyse découlera l'ordre des Cas d'Utilisations implémentés dans les itérations : les plus risqués seront traités en premier.
- **Gérer la demande par les Cas d'Utilisations**. Le processus prend la description des besoins sous l'angle des Cas d'Utilisations. Cette description part du point de vue de l'utilisateur du produit, de ce qu'il veut voir et obtenir. La description est en boite noire : des scénarios listent les échanges sollicitations utilisateur / réponses du système. Qu'importe comment le système produit sa réponse.
- **Définir une architecture par composants**. Le but est de créer une solution qui soit flexible et évolutive. Il s'agit également de promouvoir la réutilisation de composants existants.
- **Concevoir visuellement.** L'activité de conception doit permettre de mettre en évidence une solution clairement compréhensible par tous les acteurs amenés à la valider. UP préconise de s'éloigner des ambiguïtés du texte pour privilégier les schémas. Le système est décrit à l'aide du formalisme UML, en 5 vues :
  - La vue **logique** définit la statique du produit (sa structure).
  - La vue **implémentation** définit les solutions de codage.

- La vue **comportement** définit la dynamique du futur système, ses interactions.
- La vue **déploiement** définit les nœuds et réseaux d'exploitation du produit.
- La vue **utilisateur** définit les fonctionnalités ou utilisations du produit.

- **Tester et vérifier la qualité.** Le rythme de livraisons fréquentes pousse à s'assurer plus souvent de la qualité des travaux. (C'est d'ailleurs aussi une difficulté d'adoption). Des pratiques communes avec XP ou Scrum sont portées par UP pour impliquer tous les acteurs du projet dans la vérification de ce qui est produit : revue d'itération, revue de code ou de documents, tests automatisés.
- **Contrôler les changements.** C'est un peu le but en définitive : de permettre, voire de favoriser, les changements en contrôlant leur introduction dans le produit en construction. La gestion des besoins par scénarios (cas d'utilisation), la traçabilité dans les tests et dans les versions formées, la mise en œuvre de tests de non régression automatisés travaillent dans ce sens.

Cycle de vie UP : disciplines

UP décrit le déroulement précis des étapes par lesquelles passer pour mettre en œuvre ces pratiques. Il est donc plutôt directif.

Les étapes en questions sont distribuées au sein de quatre phases de projet :

L' « inception » (ou initialisation en français.)

Les parties prenantes établissent la vision du projet. Le document de Vision est un des livrables types d'UP et il en fournit un modèle. Il s'agit d'expliciter les enjeux et objectifs du projet, l'intérêt à faire. Il s'agit également de proposer différents scénarios de projets : développements spécifiques ? Intégration ? Achat de progiciel ? Etc.

L'élaboration

Cette phase est a priori déroulée sur 2 itérations. Mais ce n'est qu'indicatif et ce n'est pas parce que 2 itérations sont passées que la phase est complète. Le but de l'élaboration est de prouver

la faisabilité du scénario de projet retenu. La solution est décrite et prototypée.

La construction

Cette phase déroule plusieurs itérations et implique toute l'équipe. Chaque itération est un mini cycle – répété séquentiellement - d'affinage des cas d'utilisations à implémenter, d'implémentation de ces cas, de tests unitaires et d'intégration. Elle aboutit à une livraison intermédiaire qui, comme un sprint, fait le point sur la vélocité de l'équipe c'est à dire sa capacité à implémenter tout ou partie des scénarios visés (~ Nombre de scénarios / itération). C'est aussi le moment de collecter et de prioriser les demandes de changements.

La transition

Une ou deux itérations finales pour transférer aux utilisateurs le produit construit: création des manuels d'utilisation (support de formation) et document d'exploitation, formations des administrateurs et des formateurs, déploiement physique sur les sites : sites pilotes, puis par vagues successives.

Au cours de chaque phase successive, toutes les activités typiques d'un projet de développement informatiques sont mises en œuvre. UP parle de « disciplines » :

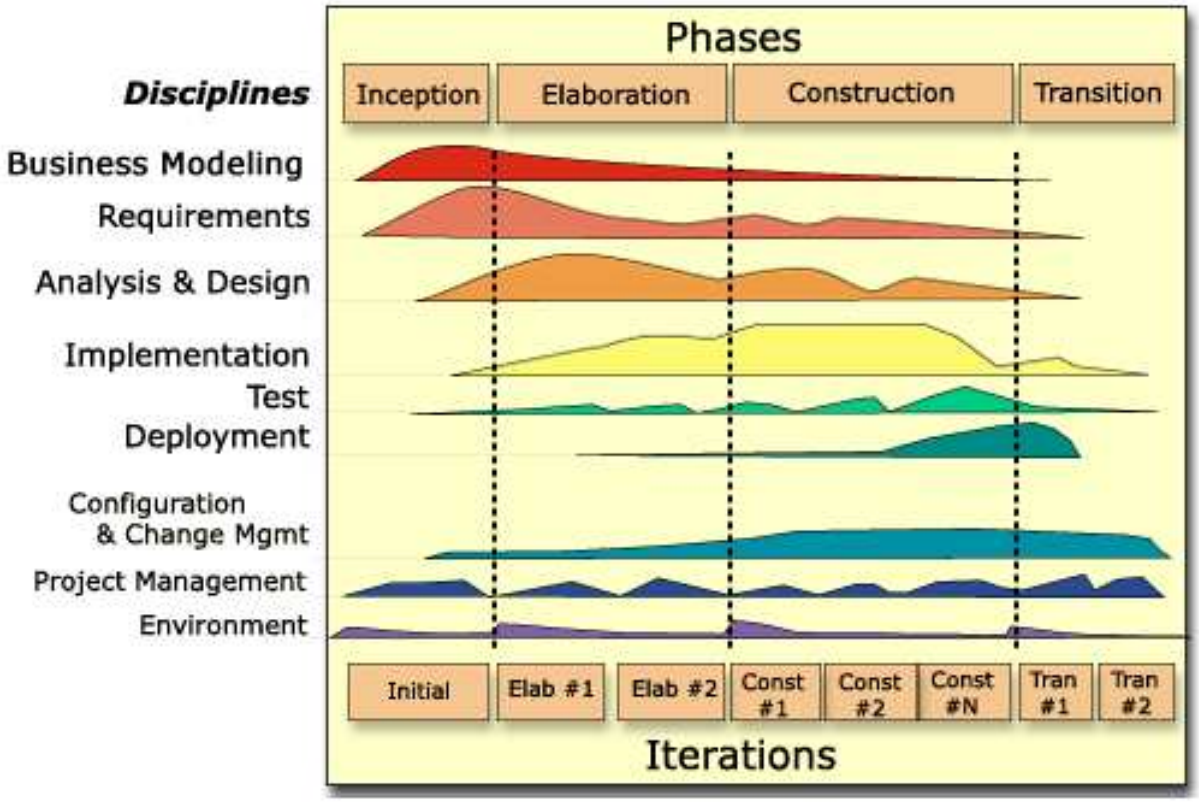
- Modélisation métier
- Collecte et développement des exigences (la demande)
- Analyse et conception
- Implémentation
- Tests
- Déploiement

Auxquels s'ajoutent des disciplines dites de support, en ce qu'elles supportent les précédentes :

- Gestion de projet
- Gestion de configurations
- Gestion des environnements

Le schéma ci-dessous représente la mobilisation des disciplines au cours des phases successives. Ce schéma bien connu est extrait de la présentation du Rational Unified Process (IBM).

Il a pour objectif de montrer que toutes les disciplines sont sollicitées en parallèle et non séquentiellement. Ainsi, l'implémentation n'attend pas la fin de l'analyse pour démarrer. Ce qui ne signifie pas pour autant que le développement anticipe sur l'analyse, mais que les travaux sont fragmentés.



Agile UP (AUP)

Agile Unified Process (AUP) est une simplification d'UP (par A.Scott) afin de « l'agiliser ».

En quelques mots, l'idée est d'être moins directif et plus ouvert qu'UP.

Par exemple AUP ne considère pas les Cas d'Utilisations comme l'outil principal de collecte des besoins. Il reprendra ce qu'un processus comme XP, par exemple, invite à utiliser : des choses plus simples comme les *User Story*.

Et un outil comme le *Storyboard* permet de mieux représenter le besoin IHM, plus visuellement, qu'un Cas d'Utilisation.

De même, AUP étend les possibilités de conception au-delà du seul usage d'UML. L'*Agile Model Driven Development* (AMDD) mis en avant par AUP introduit dans chaque itération une spécification détaillée de la solution en 2 temps :

- Un *Model Storm* : session orale courte (< 30mn) entre développeurs (2 ou 3) qui réfléchissent ensemble sur la façon d'implémenter au mieux telle ou telle question. Le fruit de leur réflexion (le « livrable ») recouvre tout simplement un tableau blanc, puis formalise des tests.
- Un *Test Driven Development* : une fois que les développeurs ont établis ensemble comment s'y prendre pour coder la fonctionnalité, ils formalisent d'abord les cas de tests. Des outils tels que JUnit et PHPUnit permettent de coder et d'automatiser ces cas. Le code sera ensuite progressivement enrichi afin de réussir les cas.

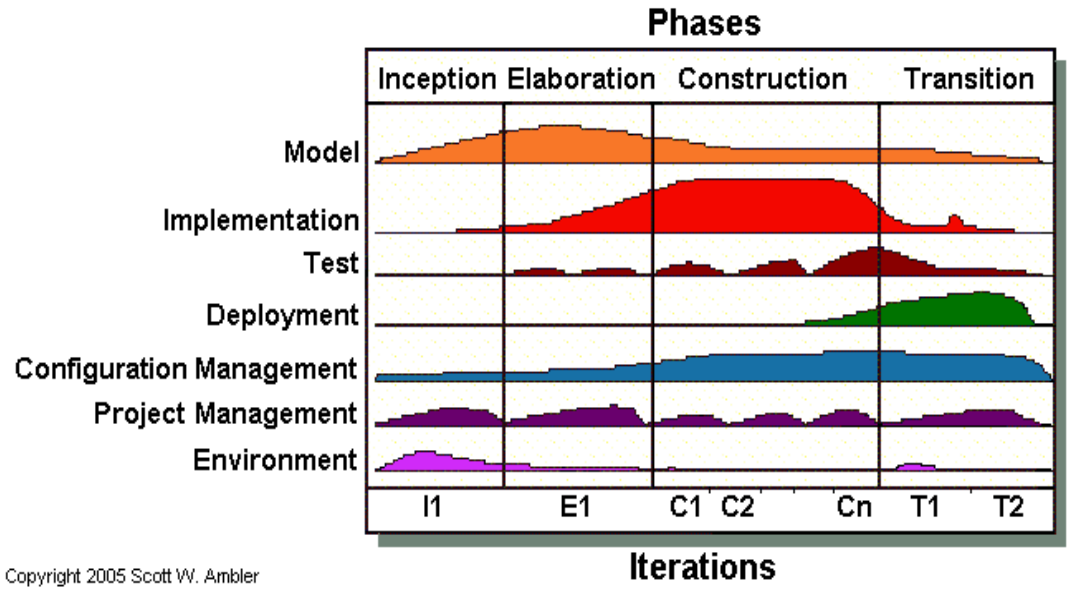
Nous voyons que la conception (le design) AUP est passablement moins formelle ou documentée qu'en UP natif. Nous nous rapprochons d'XP.

Cette approche itérative des spécifications détaillées par cas de tests est dite *Just In Time*, puisqu'elle est produite au moment même de s'atteler au développement de la fonctionnalité. Nous y entendons l'écho de Lean qui se refuse à fixer un choix tant que toutes les options sont vivantes.

Ainsi Agile UP simplifie UP. AUP insiste sur cette approche moins formelle, moins documentaire, des méthodes agiles. Les livrables UP sont des possibilités. AUP demande de n'instancier que celui qui sert dans un contexte précis et encore, sans nécessairement lui donner une forme écrite très riche.

Les 'disciplines' que nous présentions plus haut se retrouvent de façon très proche. La discipline Modélisation réunit la modélisation métier, la gestion des exigences et les analyses et conceptions. Le but étant toujours d'identifier une solution viable pour la demande.

Le schéma des disciplines par phases qui caractérise UP est transformé de la façon suivante dans AUP (par Scott W. Ambler).



Open UP

Forme simplifiée, open source et agile d'Unified Process (http://www.eclipse.org/epf/downloads/openup/openup_downloads.php).

Open UP a pour prérequis une petite équipe (5 personnes ou moins), située en un même lieu et sur une courte durée (moins de 6 mois).

Open UP est typiquement une convergence Scrum / UP. Nous y retrouvons les *daily stand up meetings*, la *backlog* sous l'appellation *Work Items List*, etc.

Mais Open UP garde le vocabulaire et les orientations directives principales d'UP : conduit par les cas d'utilisations, des rôles et des livrables documentaires précis.

CAS ET REPONSES ADAPTEES

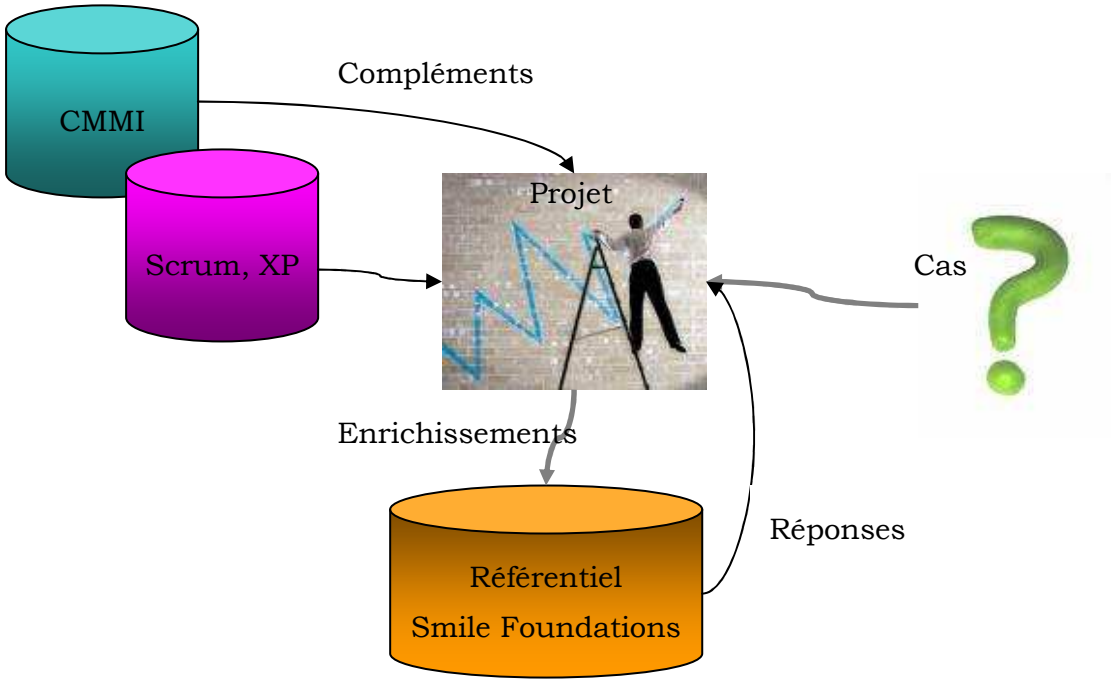
Smile Foundations : la démarche

Les outils que nous venons de passer en revue ne constituent pas tous les choix possibles du marché. D'autres matériaux existent que nous n'avons pas présentés : Cobit, ITIL, PMI…Mais leurs finalités sont différentes.

Quoi qu'il en soit, nous disposons maintenant d'une boite à outil, d'un catalogue d'exercices, assez riche pour adresser des cas réels. C'est l'objet de ce chapitre où nous exposons différents cas qui nous permettent d'illustrer concrètement l'exploitation de cette boite à outil.

Nous essayons de dégager, au travers de ces cas, une démarche opportuniste d'utilisation de ce qui se fait de mieux, plutôt qu'une méthode fixée.

Smile Foundations, notre approche interne pour les forfaits, est l'empreinte laissée par nos expériences. Mais c'est surtout une démarche de capitalisation, une façon que cette empreinte a de s'élargir à l'occasion des nouveaux cas ou de nouvelles contraintes.



Smile Foundations

Ce que montre ce schéma, c'est que notre référentiel est construit empiriquement, à partir de standards adaptés aux cas vécus par nos projets. Notre démarche interne est ainsi un effort structuré pour identifier dans les standards les réponses aux cas nouveaux.

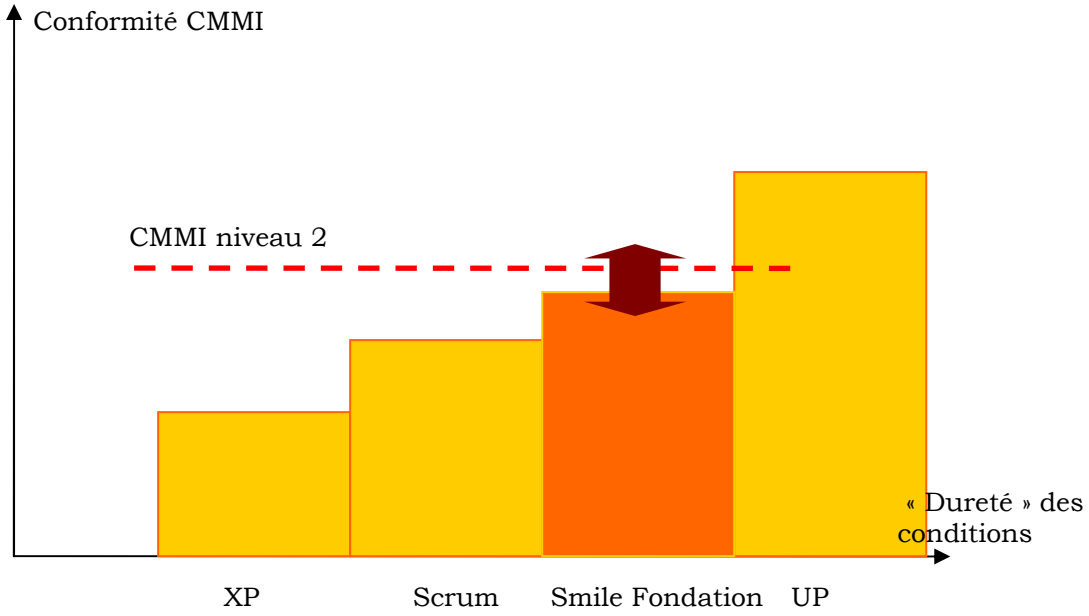
La boucle de capitalisation est assez classique. C'est d'ailleurs une préconisation du CMMI (*process focus* et definition).

Ci-dessous, nous reprenons le graphe que nous utilisions pour comparer les différentes approches par rapport à l'usage qu'elles font du CMMI. Nous positionnons Smile Foundations comme moins agile que les méthodes XP et Scrum. Pourquoi ? Parce que nos conditions naturelles de projets ne sont pas toujours « *Agile compatibles* ». Par expérience, nous injectons davantage de CMMI dans nos structures projet pour les rendre plus solides et peut être moins souples.

Mais attention, comme nous l'indiquons plus haut, Smile Foundations est avant tout une démarche adaptative qui utilise des standards.

Si les conditions sont Agile-compatibles, la démarche Smile sera automatiquement Agile.

Si elles ne le sont pas, elle utilise les ressources CMMI. Le niveau 2 CMMI n'est donc pas un but. C'est une réaction adaptative à l'environnement.



Etudes de cas

Cadre forfaitaire de marché public, avec périmètre variable

Contexte. Prenons par exemple un projet en 3 lots de 820 jh au total en Java/J2EE, sur 9 mois, au forfait et répondant à un cahier des charges qui prime sur tout autre document ultérieur. Le cas n'est pas rare.

Enjeux. Le projet doit créer l'interface web d'un outil de requêtes et le moteur de requêtes lui-même. C'est l'une des composantes d'un programme de refonte. Il doit pouvoir s'adapter aux influences et dépendances des autres projets du programme.

De plus, le périmètre doit pouvoir évoluer d'un lot à l'autre en fonction de l'avancement de chaque composante.

Contraintes

- Le budget, la date de livraison et les grandes lignes du périmètre sont fixes.
- L'administration exige un reporting d'alertes et des indicateurs précis.
- Des périodes de recettes (VABF et de VSR) sont imposées pour chaque lot.

Solutions

Dans l'énoncé du cas on peut s'étonner de ce que le périmètre est à la fois fixe et doit pouvoir évoluer d'un lot à l'autre. Les grandes lignes décrites dans le cahier des charges (CCTP, Cahier des Clauses Techniques Particulières) sont fixes. Mais leurs détails sont variables. (Or le diable est dans les détails…)

Nous utilisons d'abord le principe de *Product Backlog* de Scrum, associé à un chiffrage en taille. Chaque demande majeure du cahier des charges est listée dans la *backlog*, évaluée en unités d'œuvres et assignée sur un lot par priorité : 1, 2 ou 3. Nous disposons alors d'enveloppes précises.

Les spécifications n'entrent pas dans le détail, mais identifient ce qui est connu. Les cas de tests viennent compléter les spécifications par les détails établis au fur et à mesure de l'avancement de tout le programme. (Approches Scrum et XP).

Le chef de projet s'assure en continu :

1) que les cas de tests qui décrivent les détails d'une enveloppe ne la débordent pas en taille, c'est-à-dire que le nombre d'éléments à vérifier par les tests (rubriques, pages, etc.) n'excède pas la taille de l'enveloppe,

2) que les cas de tests sont bien reliés à des *backlog items* et donc ne sortent pas du périmètre global initial.

Ces deux points de contrôle sont préconisés par *Requirements Management* de CMMI sur la traçabilité bidirectionnelle besoins / tests et *Project Monitoring and Control* (PMC) sur l'approche d'estimation de taille.

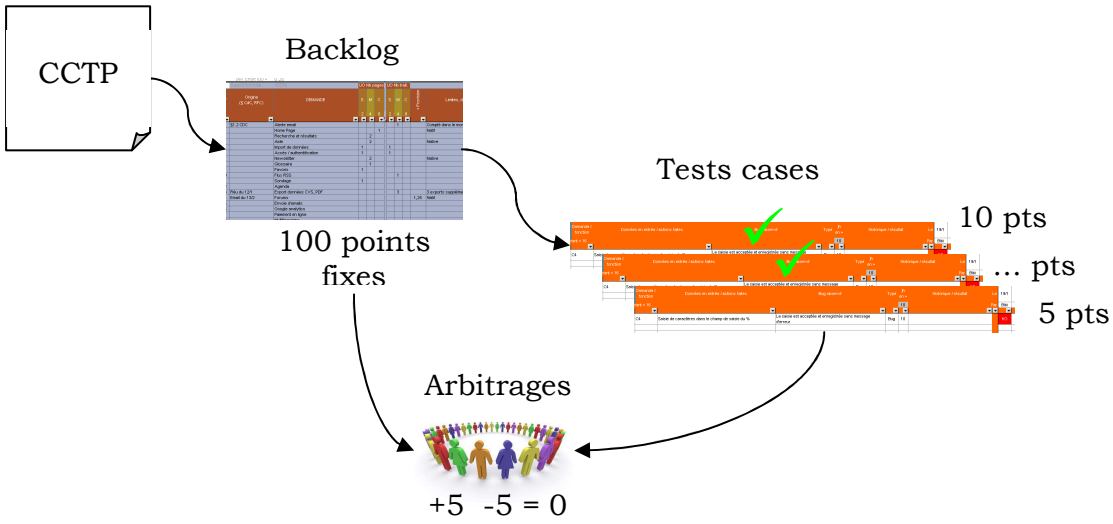
En cas de dépassement du périmètre (taille ou absence de besoin initial), le Chef de projet remonte une alerte argumentée et chiffrée. Un comité de pilotage, équivalent au *Change Control Board* CMMI

ou au *Planning Meeting* de Scrum, arbitre par ajouts / retraits de telle manière que l'enveloppe demeure fixe.

L'ajout pur, sans compensation par retraits, est évité afin de ne pas grossir le budget.

Le schéma suivant montre la boucle de traçabilité : les demandes sont extraites du CCTP et listées dans la *Backlog*. Elles y sont priorisées et estimées en taille (nombre d'unités d'œuvres). Le lien CCTP / Backlog vérifie que rien n'est oublié.

Les cas de tests sont reliés à un item de la *Backlog* : ils précisent ce qui concrètement va matérialiser la demande. Un arbitrage doit donc avoir lieu si les éléments observables décrits par les cas de tests sont plus nombreux ou étendus que les dimensions de la demande.



Les recettes de chaque lot jouent les cas de tests reliés aux *backlog items*. A ce stade, il n'est pas possible d'identifier un « trou » de périmètre du fait de la mise à jour de la *backlog*. Les dysfonctionnements sont donc des bugs ou des ajouts. Les ajouts sont identifiés et traités comme précédemment. Les bugs sont rares car ayant été identifiés par l'équipe en intégration continue, mais également parce qu'un rôle spécifique de testeur est assigné à un membre de l'équipe, en plus des vérifications unitaires et automatiques. Les versions intermédiaires sont intégrées et testées fréquemment.

Nous voyons dans ce cas que nous sommes contraints d'utiliser des outils non agiles (voir déconseillés par l'Agile) : la traçabilité, les tests de recette finaux. Ne serait-ce que parce que ce sont des exigences client ou, pour le moins, des leviers utiles pour remplir notre part du contrat.

Projet multi sites

Contexte. Un de nos projets parisiens doit intégrer et développer une solution autour de l'outil de GED open source Alfresco.

Enjeux. La solution s'éloigne un peu du natif Alfresco et nécessite des développements spécifiques.

Contraintes. L'expert senior sur Alfresco disponible pour le projet est dans notre agence de Nantes alors que le client et l'équipe initiale est sur Paris. Le projet doit donc être scindé en deux équipes, car le déplacement et le rapprochement physique n'est pas possible sur la durée.

Solutions. La contrainte multi-sites rend certaines pratiques agiles difficiles : *daily meeting*, binôme de programmation, formalisation et documentation aussi faible que possible.

La distance impose un formalisme que l'agilité évacue en général pour simplifier : le tableau blanc dans la pièce de projet n'est plus suffisant.

En fait le *daily meeting* peut être conservé assez facilement en utilisant Skype ou tout simplement le téléphone. C'est même la possibilité pour l'équipe scindée de reconstituer une unité quotidienne. Les tâches réparties dans l'équipe sont listées dans une *Sprint Backlog* déduite de la *Product Backlog* du projet, elle-même dérivée du cahier des charges client.

Chaque jour, par email, les 2 équipes échangent la *backlog* en y ayant noté ce qui est à faire, en cours par l'un ou l'autre, terminé.

Le travail en binôme d'XP est nécessairement atténué et s'aligne sur la revue de code par les pairs de CMMI/Verification. Quoique le formalisme de la revue soit léger : les retours sont notés et échangés par email, et non sur une grille de remarques avec suivi de chacune d'elles. Les revues peuvent avoir lieu sur les items marqués terminés de la *Sprint Backlog*.

La *Sprint Backlog* peut être un simple fichier échangé par email ou partagé sur réseau. C'est le moyen le plus souple. Un fichier partagé sur Google docs ou sur un gestionnaire de tâches gratuit serait également possible (fait sur d'autres projets).

Nous voyons bien que la distance empêche le confort de la communication immédiate et volatile (post-it déplacés sur un tableau pour matérialiser les tâches) et nous impose un formalisme minimum.

Interlocuteurs client multiples

Contexte. Une PME souhaite renouveler complètement son site internet. C'est un projet à la fois critique pour la visibilité de l'entreprise et en dehors de son cœur de métier. Le patron de la PME confie le projet à un stagiaire mobilisé pour cela.

Enjeux. Il nous faut créer un site qui reflète l'identité et le message de l'entreprise tout en y apportant les bonnes pratiques du web.

Contraintes. Même une fois établi le storyboard et la création graphique, le client est très sollicité pour donner ses retours sur le site en construction. Il doit être tout à la fois disponible et pertinent sur son besoin par rapport à ce que nous lui montrons. Or cela correspond à deux acteurs distincts : le stagiaire est très disponible mais ne peut prendre toutes les décisions à la place du patron.

Il n'est pas rare qu'un nouvel acteur soit invité par le client en cours de projet, par exemple un consultant en communication.

Solutions. Dans ce cas précis, il nous manque un prérequis important de l'Agile. L'implication d'un *Product Owner* précis, sachant et décisionnaire.

En l'occurrence, deviennent fort probables et impactant des risques tels que : 1) des réunions de travail consommatrices en temps parce que le stagiaire profite de sa disponibilité pour bien détailler chaque élément, 2) de la reprise de travaux (*rework*) ou des divergences par rapport au besoin car le décisionnaire n'est au contraire pas assez présent et remet en question les validations du stagiaire. 3) Remises en questions accentuées par la survenue d'un acteur tardif, le consultant.

Nous nous tournons dans ce cas vers les gardes fous préconisés par CMMI. Le but n'est pas d'alourdir par un formalisme excessif, mais de pouvoir alerter et corriger lorsque l'un de ces risques se révèle.

Pour commencer, nous identifions les acteurs initiaux du projet dans un Plan Projet validé par le client. Il a valeur contractuelle. Ce n'est pas très agile, mais nécessaire pour pouvoir réagir lorsque l'implication de chacun n'est pas au rendez-vous.

Le Plan est également l'endroit où nous pouvons préciser nos besoins : quelle validation par l'acteur décisionnaire ? Sur quelle durée (*'time boxée'*, en allers/retours limités, etc.) et quelle fréquence ? Que valent les décisions de l'acteur stagiaire ?

Bien sûr ces vœux ne seront pas forcément suivis des faits. Mais si la règle du jeu n'est pas écrite et acceptée, il ne nous sera pas possible d'alerter, de gérer les surcoûts ou de proposer les aménagements d'équipe et de disponibilités permettant au moins de limiter les coûts.

Le suivi de ces valeurs : nombre ou durée cumulée des réunions de travail, nombre de re-livraisons, dépassement des délais de validation, nombre de retours (ou absence de retours) lors des *Sprint Reviews*, etc. sont autant d'indicateurs définis comme spécifiés dans CMMI/Mesures et analyses. Et ces indicateurs permettent tout d'abord de suivre les risques identifiés et évalués (CMMI/RSKM), de vérifier l'efficacité des actions demandées (l'implication des acteurs), de chiffrer les écarts.

Smile Foundations : le référentiel

Les réponses que nous introduisons par rapport à quelques cas réels lèvent un peu le voile sur ce qu'est notre méthode.

Le propos de ce livre blanc n'est pas de promouvoir Smile Foundations comme la dernière coqueluche à attraper.

Notre méthode s'appelle Smile Foundations, puisqu'il faut bien lui donner un nom, mais il ne s'agit pas de proposer, et encore moins d'imposer, une méthodologie nouvelle. L'essence même de Smile Foundations est l'adaptabilité et notre méthode pourrait tenir en un précepte : prendre le meilleur des méthodes agiles lorsqu'on le peut et y intégrer des éléments issus de méthodes moins agiles lorsqu'il le faut.

Plus généralement, nous verrons en dernière partie la démarche que nous préconisons pour qu'une entreprise créé sa propre méthode adaptable.

Mais il serait dommage de ne pas présenter succinctement Smile Foundations alors que nous l'évoquons ! Cela peut aussi servir d'illustration du genre de référentiel méthodologique auquel une approche « multi standards » peut aboutir.

Objectifs, applicabilité et contraintes

Nous avons besoin pour nos activités au forfait d'une méthode qui nous procure :

- Des **sécurités**, contre les risques inhérents à ce type de projets.
- De **l'homogénéité** afin de permettre à nos clients de nous reconnaître, quels que soient les acteurs du projet. Et permette à ces acteurs (nos collaborateurs) de pouvoir transiter rapidement sur un projet nouveau.
- De **l'agilité**, afin de servir au mieux le besoin client. Mais aussi pour que nos équipes aillent rapidement à l'essentiel sans passer par les cases « redondances », « report de mises à jour », « informations inexploitées », etc.

Et notre méthode doit satisfaire ces objectifs tout en prenant en compte des contraintes telles que :

- Un processus d'avant-vente rapide, aboutissant à une estimation détaillée du périmètre.
- Des projets de 1 à 12 mois, regroupant chacun de 1 à 10 personnes : c'est-à-dire que nos ressources pour être occupées en continu, passent d'un projet à l'autre très vite ou sont affectées sur plusieurs projets en parallèle.
- Des technologies et des profils de ressources qui évoluent souvent en compétences.
- Des profils d'interlocuteurs clients très variés. Selon la fonction : marketing, communication, DSI, comptabilité et gestion, direction générale. Et selon l'entreprise : TPE/PME, grosses entreprises, industrie, administration publique.

- Un mode contractuel forfaitaire, avec garantie de 3 à 6 mois et des pénalités possibles en cas de dépassement de délais ou de non qualité.

Les rôles

Nous venons d'évoquer les principales parties prenantes de nos projets : nos ressources et bien sûr les interlocuteurs client. Il y a d'autres acteurs. Voici une présentation des principaux acteurs:

Le Sponsor. Pas de surprise, un projet n'existe que s'il est financé. Le Sponsor est celui qui maitrise le budget. C'est donc un acteur client et il détient le « final cut », c'est-à-dire la décision ultime en cas de modifications des paramètres du projet : périmètre, délais, qualités, technologies.

Cet acteur est impliqué au moyen de réunions d'un Comité de Pilotage (Copil), qu'il préside.

Le **Product Owner** ou Chef de projet client. Il connaît le besoin fonctionnel et les contraintes techniques. Il a en tout cas la capacité d'obtenir rapidement les informations fonctionnelles et techniques nécessaires.

Cet acteur est très souvent impliqué dans le projet, du lancement à la clôture. Notamment aux moments des tests, lesquels sont répartis tout au long du cycle de vie du projet.

L'**Ingénieur d'Affaire (IA)**. Il est chez nous le premier contact du client, Sponsor et Product Owner. Il propose des solutions fonctionnelles et techniques et un budget adapté, sur la base de métriques par technologies et unités d'œuvres.

Sollicité par les appels d'offres nombreux qui exigent des réponses rapides, l'IA travaille en flux tendu sur de nombreux sujets. Il est donc impliqué en début de projet pour passer le relai à un Chef de projet, puis intervient au cours des réunions de Comité de Pilotage.

Le **Chef de projet**. Il est la pierre angulaire du projet. Il possède en profondeur la compréhension fonctionnelle et budgétaire du projet.

Le Chef de projet intervient en continu sur les projets dont il a la charge. Il peut en suivre plusieurs en parallèle, mais pas plus d'un seul dans une phase donnée du cycle de vie.

Le **Responsable Opérationnel**. Il seconde le Chef de projet sur un projet précis. Il possède la connaissance fonctionnelle et technique du projet. Notamment, il organise les tests et en est responsable.

Le **Consultant Ergonome** et le **Directeur Artistique (DA)**. Ils interviennent en début de projet pour identifier avec le client les comportements du produit. Ils proposent au client les comportements répondant aux meilleures pratiques du web (notamment en matière d'accessibilité) et les créations graphiques.

L'**Ingénieur d'Etude et Développement**. Répartis en binômes sur des modules, ils construisent le produit : montage, tests unitaires, développements, intégrations.

Les **Directions Supports**. Technique et Qualité, elles apportent de la sécurité aux équipes en évaluant les facteurs de risques et en préconisant les outils adéquats. Elles procurent de la sécurité au client également en opérant plusieurs revues d'un même projet pour en identifier et faire corriger les dysfonctionnements (par rapport à la présente démarche).

Cycle de vie du projet

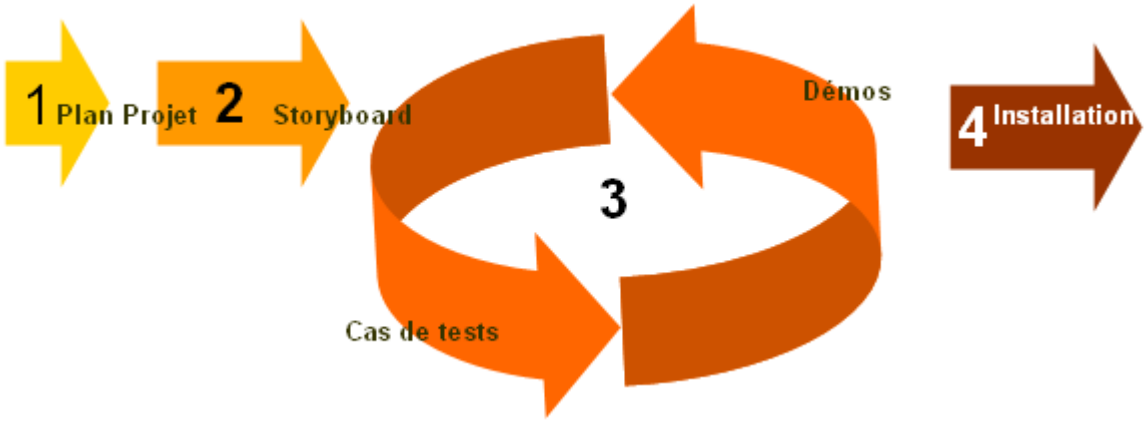
Un projet Smile est découpé en quatre phases principales. Les phases 1 et 2 – qui comme pour UP, sont baptisées Initialisation et Elaboration - préparent le forfait.

L'Initialisation fixe la *baseline* du projet dans un **Plan Projet ou PQP** (Plan Qualité Projet) : jalons, périmètre, acteurs du projet. Le périmètre est décrit en relation avec le cahier des charges dans une ***Product Backlog*** simple et priorisée.

L'Elaboration met en œuvre la description du produit : technique au moyen d'un **document d'architecture** ou d'un **prototype**. Fonctionnelle au travers d'un **Storyboard** et d'un document de **Spécifications Fonctionnelles** qui en fait complète le Storyboard en décrivant les comportements des pages IHM définies.

L'Elaboration permet aussi de préparer et de démarrer les tests : un document de **Stratégie de tests** défini quels tests seront réalisés, par qui, quand et avec quels critères de réussites. Un **Cahier de**

tests (éventuellement matérialisé dans un outil tel que TestLink par exemple) permet de détailler les comportements concrets brossés par les Spécifications Fonctionnelles.



Cycle de vie Smile Foundation

La phase 3 de Construction est très itérative. Elle déroule des périodes successives de 2 semaines. Chacune démarre par un approfondissement des **cas de tests** correspondant aux *Backlog Items* priorisés en Initialisation / Elaboration. Puis s'achèvent sur un **Point de Visibilité** formel, c'est-à-dire une présentation d'une version intermédiaire testée et stable au Product Owner. Les items de la backlog démontrés sont marqués « acquis ».

Les demandes supplémentaires ou de corrections sont ajoutées à la *Product Backlog*. Les items sont repriorisés.

Ce que Scrum appelle la *Sprint Backlog* est matérialisée par l'équipe projet en fonction de son besoin : post-it sur tableau blanc pour une équipe localisée ou document partagé type Google doc pour une équipe distribuée.

Les méthodes agiles ne fricotent pas trop avec le reporting formel, mais la contrainte forfait ou encore le besoin particulier de certains interlocuteurs client (voir les profils évoqués précédemment) nous l'impose. Un **Compte Rendu Hebdomadaire** fait un état exhaustif et prédictif du projet : où en est-il par rapport à la *baseline* d'Initialisation ? Quelles scénarios et probabilités de bonne fin a-t-on ?

Enfin, la phase 4 dite de Transition recouvre les tests client (recette client, VABF), les **formations utilisateurs**, la **documentation technique**, le packaging et le déploiement.

Synoptique des livrables par processus et par phases

Tous les livrables indiqués dans ce tableau sont instanciés par l'équipe projet à partir de modèles téléchargeables depuis notre wiki méthodologique interne.

MANTIS est l'outil que nous avons choisi pour enregistrer et suivre les demandes d'évolutions ou de corrections.

Processus				
Planifier, Gérer les changements et Suivre	CR Hebdo Initial PQP Lancement	CR Hebdos Backlog Copils	CR Hebdos Backlog Copils	CR Hebdos Backlog Copils
Concevoir et Spécifier		Storyboard Spécifications Architecture Cas de Tests	Cas de Tests	Cas de Tests
Gérer les Environnements	Outillage Processus	Env de développement et d'intégrations	Env de recette et de pré prod	Env de prod
Développer et Tester		Proto Stratégie tests	Rapport Tests Points de Visibilité Versions MANTIS	Rapports Tests Suivi de recette Versions MANTIS
Déployer			Doc d'installation D'exploitation	Formations
Phases :	Initialisation	Elaboration	Construction	Transition

Activité principale	Activité soutenue	Activité à faire	Activité absente ou ténue

DEMARCHE POUR CREER SA PROPRE METHODE

Smile Foundations est notre démarche. Elle est conçue pour répondre à nos besoins.

Comment avons-nous créé cette méthode ? Comment l'avons-nous déployée ?

C'est à ces questions que répond cette dernière partie. Nous y décrivons un programme, plutôt classique dans sa démarche, d'étude de situation, de définition méthodologique et de déploiement.

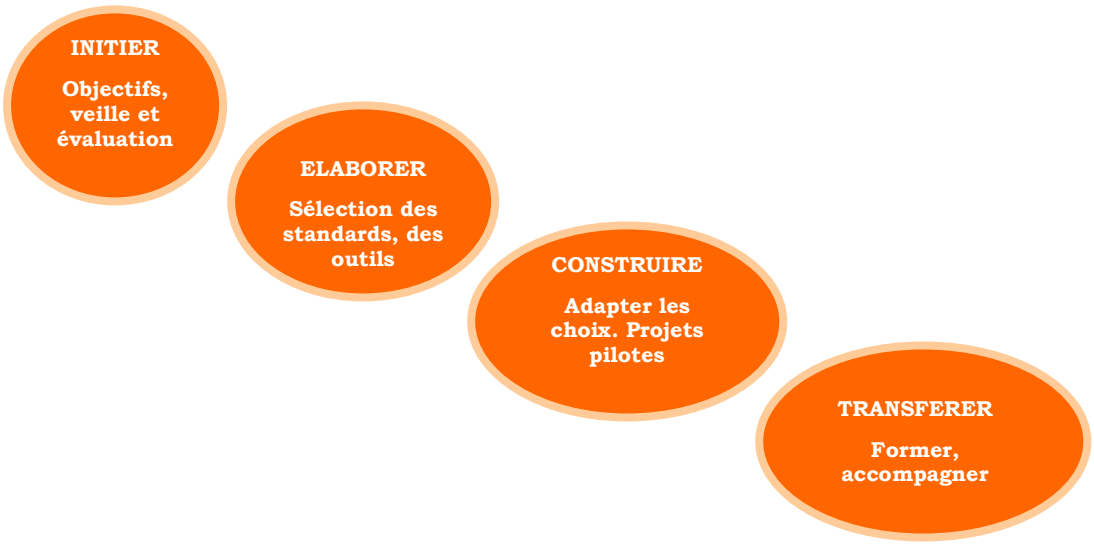
Ce type de programme peut être déroulé dans différents types d'organisations, aussi ne vise-t-il pas à l'originalité (l'originalité n'étant d'ailleurs pas vraiment un but en soi en matière méthodologique).

Programme méthodologique

Ce programme est tout simplement le projet de conduite du changement d'une organisation voulant se doter d'une méthode personnalisée et reposant sur les hypothèses suivantes :

- Utilisation maximale de standards agiles
- Utilisation autant que de besoin de standards plus formels

Le programme se déroule selon les phases suivantes :



Ce déroulement est lui-même plutôt standard. Rappelons encore une fois que l'objectif du programme est de transmettre une capacité à produire sa propre méthode. Mais en se basant sur des outils publics et standards.

Initier.

L'unité en charge de créer et de maintenir des applications éprouve le besoin d'optimiser son fonctionnement. Sentiments de processus lourds, de redondances, de dysfonctionnements…Pourquoi ne pas passer à l'Agile ? Les principes sont sains et de bon sens, nous l'avons vu dans la première partie de ce document.

Une compréhension des objectifs de l'unité s'impose. Il ne saurait être question de faire de l'Agile pour la seule beauté du geste. Veut-on intégrer de nouvelles technologies dans les projets ? Veut-on adresser de nouveaux types de projets ? Veut-on résoudre des blocages ? Souhaite-t-on remotiver les équipes, leur apprendre l'autonomie et le 'travailler ensemble' ? Etc.

Connaissant ces objectifs, une évaluation des contraintes de l'unité et de ses pratiques courantes par rapport aux standards présentés plus haut permettra :

1. De mesurer la distance ou l'effort à résoudre pour atteindre l'objectif.

2. De ramener, si nécessaire, l'objectif dans la zone du faisable.

Comme pour un projet classique, on pourra envisager des étapes ou lots.

Parallèlement, une veille / étude méthodologique et technologique permet d'entendre les possibilités nouvelles qui peuvent faire tomber un obstacle du rang de contrainte à celui de problème. C'est-à-dire à quelque chose de soluble.

Go ou no go pour la suite ? Dès cette étape l'unité peut décider que le jeu n'en vaut pas la peine ou qu'elle doit encore mûrir son besoin. Ou au contraire : une vision motivante a émergé.

Elaborer.

Quels outils ou exercices sont les plus adaptés à la situation ? Pour quels risques / bénéfices ?

Différents scénarios vont ressortir. On pourra en retenir plusieurs, pour les tester.

Des outils peuvent apparaître souhaitables. Des études d'opportunités permettront d'aboutir à des cahiers des charges pour les développer ou adapter des existants.

Un outillage méthodologique à la fois très standard et très simple comme celui préconisé dans Scrum (tableau blanc, liste partagées Google, tableur échangé…) peut tout de même donner lieu à description des spécificités souhaitées.

Go ou no go ? L'unité peut découvrir que le coût ou les bénéfices attendus ne sont pas ce qui était espéré. A contrario, des outils simples peuvent apparaître très appropriés alors qu'on les pensait inavouablement simplistes.

Construire.

Les choix et les spécifications des besoins établis précédemment sont mis en œuvre dans cette phase.

Les outils sélectionnés sont adaptés : on parle d'outils tant méthodologiques (CMMI, Open UP ou XP…) que techniques (Trac, MANTIS, Jira, Open Office documents, etc.)

L'outillage ainsi forgé est testé auprès d'utilisateurs critiques et sur des projets pilotes. Certains sont abandonnés, d'autres renforcés. Comme pour un projet classique, la bonne pratique des itérations est retenue. L'outillage est construit par incréments, ou versions partielles. Chacune de ces versions est vérifiée par un groupe utilisateurs. Puis testée sur des projets.

Des projets pilotes plus complets permettent d'éprouver une version aboutie de l'outillage, de bout en bout (sur tout le cycle de vie de projet).

Go ou no go? Les pilotes se différencient-ils en mieux ? Selon quelles mesures ? On aura pris soin en effet de définir les métriques indiquant la distance vis-à-vis de l'objectif, c'est-à-dire de la quantité à améliorer : par exemple densité de bugs dans les projets pilotes / non pilotes, durée de mise en production, satisfaction des utilisateurs de ces projets, durée de montée en compétence des collaborateurs, etc.

Transférer.

La méthode adaptée à l'unité a fait ses preuves. Il faut maintenant former les personnels, installer et déployer les outils, s'assurer que l'unité s'est approprié la méthode.

Une durée d'accompagnement est toujours nécessaire. L'accompagnement ne consiste pas à faire à la place de : on évitera les accompagnements à temps plein pour leur préférer des revues régulières.

Le but de l'accompagnement est de finaliser le transfert. Il passe par des remotivations de tous les échelons de l'unité.

Mesurer

Deux familles d'indicateurs permettent de suivre l'avancement et la valeur ajoutée de ce type de programme méthodologique.

Les indicateurs d'avancement mesurent la construction de la méthode (par exemple, les avancements des sous-projets de paramétrages des outils sélectionnés) et le déploiement de la méthode. Ainsi, chaque revue d'accompagnement évalue la prise en main des outils par les équipes.

Les indicateurs de valeur ajoutée mesurent avant déploiement le niveau de référence d'une valeur caractérisant l'objectif de l'unité organisationnelle : taux d'anomalies, coûts, dépassements, etc.

Nous mesurons ensuite, par unité, ces mêmes valeurs pendant et après le déploiement.

Conclusion

Nous avons présenté un ensemble de possibilités en passe de devenir de réels standards pour optimiser les processus de développements informatiques.

Notre propos, on l'aura compris, est de savoir sélectionner, adapter et finalement assembler celles de ces possibilités qui répondent efficacement aux contraintes d'une structure donnée. Sans se limiter dogmatiquement à un type de processus ou d'outils.

Nous espérons avoir convaincu que, même standards et donc validée par l'expérience de nombreuses entreprises, une approche méthodologique ne sera que rarement complète et ajustée à vos besoins propres.

De plus, d'autres outils existent au-delà de ceux rappelés ici. Citons sans souci d'exhaustivité ou d'application : PMI, Cobit, ITIL, CMMI for acquisition, etc.

Non seulement aucune approche n'est complète, mais le choix est vaste. Ce qui est plutôt une bonne nouvelle.

Nous souhaitons donc achever ce livre blanc sur l'idée, mise en œuvre chez Smile, qu'au-delà de l'agilité une méthode doit être adaptable et adaptative pour être gagnante.

Cette notion d'adaptabilité n'est pas particulièrement nouvelle. Mais elle n'est pas souvent identifiée. Trop souvent une méthode figée répondant au goût du moment passe pour faire l'affaire. Se privant ainsi d'une démarche d'adaptation, seule capable de trouver le bon assemblage méthodologique répondant aux besoins réels.

REFERENCES

Ce texte est écrit en s'appuyant sur les sources suivantes :

- http://agilemanifesto.org/
- http://www.agilealliance.org/
- Scott W. Ambler http://www/agilemodeling.com/
- CMMI-DEV-v1.2.doc et http://www.sei.cmu.edu/cmmi/
- http://www.extremeprogramming.org/
- http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf
- http://www.eclipse.org/epf/downloads/openup/openup_downloads.php

Les livres blancs Smile

Les livres blancs Smile sont téléchargeables gratuitement sur www.smile.fr

- **Introduction à l'open source et au Logiciel Libre**

  Son histoire, sa philosophie, ses grandes figures, son marché, ses modèles économiques, ses modèles de support et modèles de développement. [52 pages]

- **Gestion de contenus : les solutions open source**

  Dans la gestion de contenus, les meilleures solutions sont open source. Du simple site à la solution entreprise, découvrez l'offre des CMS open source. [58 pages]

- **Portails : les solutions open source**

  Pour les portails aussi, l'open source est riche en solutions solides et complètes. Après les CMS, Smile vous propose une étude complète des meilleures solutions portails. [50 pages]

- **200 questions pour choisir un CMS**

  Toutes les questions qu'il faut se poser pour choisir l'outil de gestion de contenu répondra le mieux à vos besoins. [46 pages]

- **Conception d'applications web**

  Synthèse des bonnes pratiques pour l'utilisabilité et l'efficacité des applications métier construites en technologie web. [61 pages]

- **Les frameworks PHP**

  Une présentation complète des frameworks et composants qui permettent de réduire les temps de développement des applications, tout en améliorant leur qualité. [77 pages]

- **Les 100 bonnes pratiques du web**

  Cent et quelques « bonnes pratiques du web », usages et astuces, incontournables ou tout simplement utiles et qui vous aideront à construire un site de qualité. [26 pages]

- **ERP/PGI: les solutions open source**

  Des solutions open source en matière d'ERP sont tout à fait matures et gagnent des parts de marché dans les entreprises, apportant flexibilité et coûts réduits. [121 pages]

- **GED : les solutions open source**

  Les vraies solutions de GED sont des outils tout à fait spécifiques ; l'open source représente une alternative solide, une large couverture fonctionnelle et une forte dynamique. [77 pages]

- **Référencement : ce qu 'il faut savoir**

  Grâce à ce livre blanc, découvrez comment optimiser la "référençabilité" et le positionnement de votre site lors de sa conception. [45 pages]

- **Décisionnel : les solutions open source**

  Découvrez les meilleurs outils et suites de la business intelligence open source. [78 pages]

- **Collection Système et Infrastructure :**
  - Virtualisation open source [41 pages]
  - Architectures Web open source [177 pages]
  - Firewalls open source [58 pages]
  - VPN open source [31 pages]
  - Cloud Computing [42 pages]
  - Middleware [91 pages]

Smile
OPEN SOURCE SOLUTIONS

Contactez-nous, nous serons heureux de vous présenter nos réalisations de manière plus approfondie !
+33 1 41 40 11 00 / sdc@smile.fr