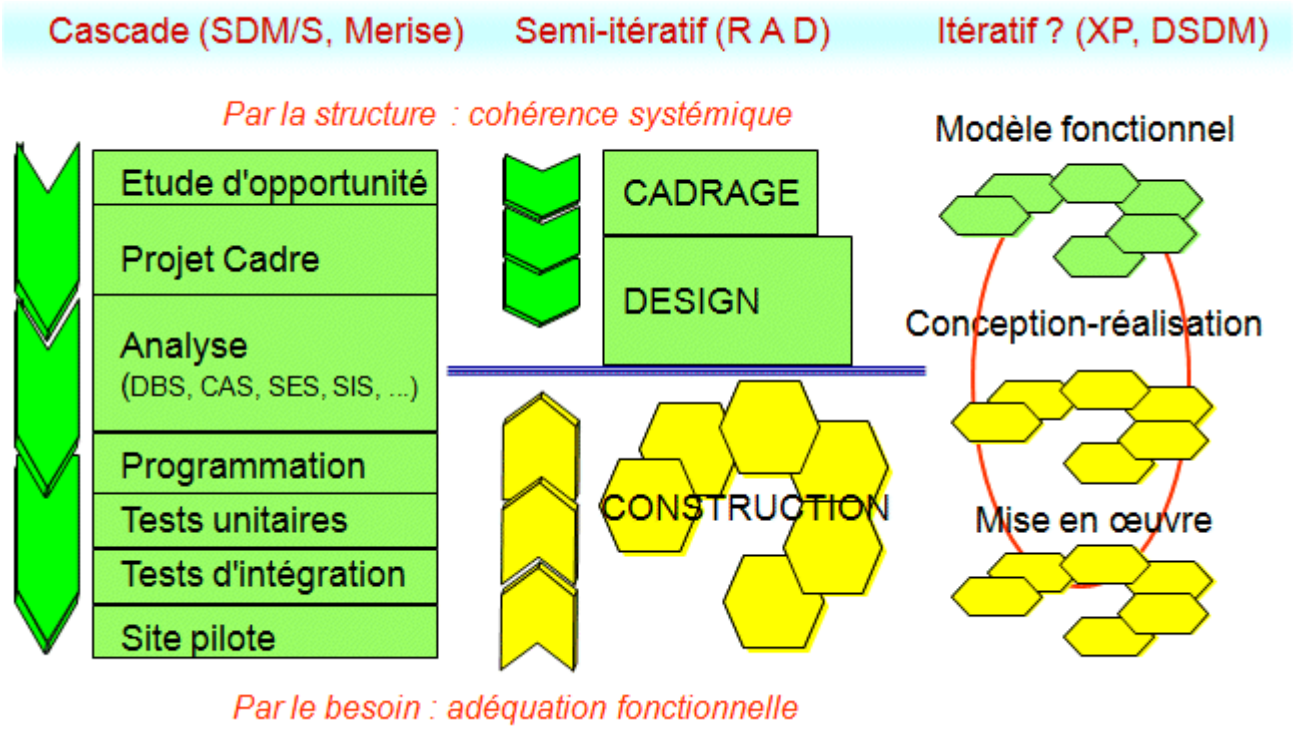
COURS PROJETS 2 :

PHASAGE DE PROJET

Table des matières

2ème étape-clef : l'organisation des actions projet

A partir des objectifs obtenus suite aux études d'opportunité et de faisabilité, l'idée va être de définir les actions associées au projet



Quel que soit la méthode choisie, le but sera de formaliser l'ensemble des actions attendues dans le cadre du projet, à chaque étape du processus

Les modèles décrits ne remettent pas fondamentalement en cause l'organisation projet vue précédemment : ils se concentrent sur les cycles des étapes de conception et réalisation et livraison du projet, et donc le périmètre PCDQ associé.
Ils permettent chacun à leurs manières la résurgence des actions minimales attendues lors de ces étapes.

Le modèle en cascade et le cycle en V

Le modèle en cascade est hérité de l'industrie du BTP : c'est le modèle « historique » de l'organisation projet industrielle.

Il repose sur les hypothèses suivantes :

- on ne peut pas construire la toiture avant les fondations ;
- les conséquences d'une modification en amont du cycle ont un impact majeur sur les coûts en aval (on peut imaginer la fabrication d'un moule dans l'industrie du plastique).

Les phases traditionnelles de développement sont effectuées simplement les unes après les autres, avec un retour sur les précédentes, voire au tout début du cycle. Le processus de développement utilisant un cycle en cascade exécute des phases qui ont pour caractéristiques :

- de produire des livrables définis au préalable ;
- de se terminer à une date précise ;
- de ne se terminer que lorsque les livrables sont jugés satisfaisants lors d'une étape de validation vérification.

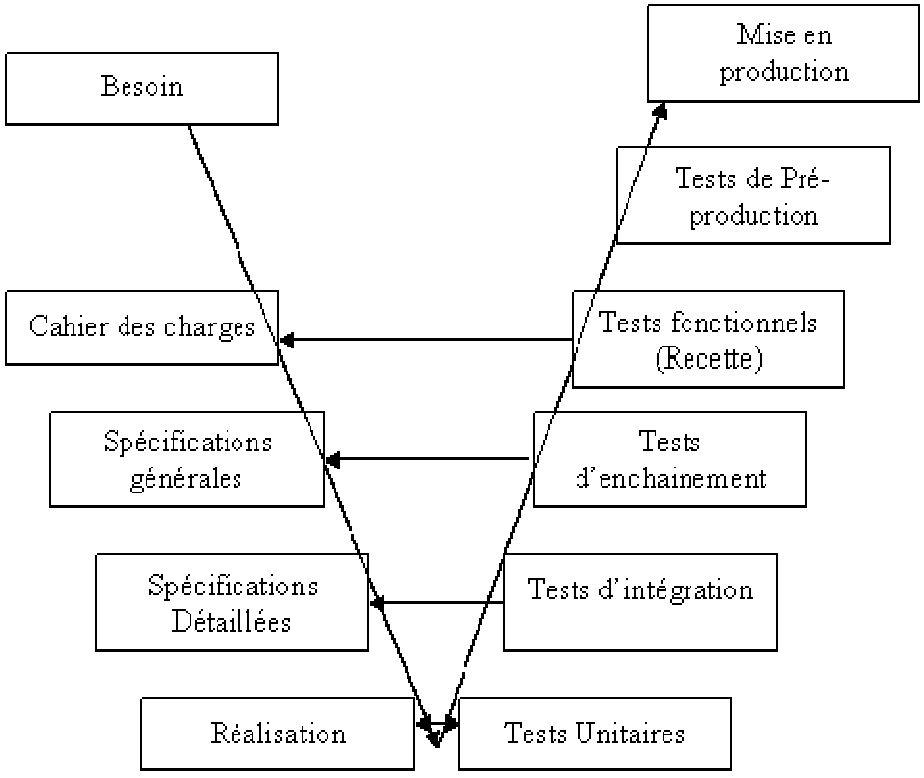
Le cycle en V

Le modèle du cycle en V a été imaginé pour permettre une analyse en cascade (du général vers le détail) en limitant au maximum les retours aux étapes précédentes, extrêmement coûteux.

Le principe est de mettre en place des phases de validation sur la partie montante du cycle, qui doivent renvoyer de l'information sur les phases en vis-à-vis lorsque des défauts sont détectés.

Le cycle en V met en évidence les étapes de spécifications ainsi que les étapes de validation associées.

Le cycle en V est devenu un standard dans le développement logiciel et la gestion de projet associée depuis les années 80.



Un des avantages des projets « cycle en V » est qu'on dispose d'un effet de lissage, qui permet de compenser un jour ce qu'on a moins bien fait la veille.
En outre, les difficultés rencontrées un jour par un membre de l'équipe ne sont pas mises en avant aux yeux de tous : le membre en question peut trouver une solution seul le lendemain ou demander de l'aide auprès de collègues.

Lexique

Expression des besoins, ou besoin : ce que veut le client, avec son langage

Cahier des charges : Description de la réponse qu'apportera le fournisseur

Spécifications générales :

→ Fonctionnelles : elles décrivent les fonctionnalités voulues par le client par acte de gestion, les performances et contraintes à respecter, les évènements déclencheurs et les éléments en sortie, le type de traitement (temps réel ou différé), les acteurs

→ Techniques : elles décrivent l'infrastructure technique et la configuration matérielle et logicielle associée, l'organisation générale des données et traitement, la décomposition du logiciel par unité de traitement (découpage technique de l'outil)

Spécifications détaillées :

→ Fonctionnelles : elles décrivent les phases des actes de gestion, et pour chacune les règles de gestion à respecter, les évènements déclencheurs et les éléments en sortie, le type de traitement (temps réel ou différé), les acteurs

→ Techniques : elles décrivent les fonctionnalités implémentées, les contrôles associés, et le mappage technique nécessaire pour réaliser la phase décrite

Développement : réalisation d'un lot de travaux cohérent

Tests unitaires (TU) : validation de l'implémentation des fonctionnalités du lot de travaux

Tests d'intégration (TI) : ils valident l'intégrité de la phase impactée par les modifications apportées par le lot de travaux (ou développements) nouvellement livrés

Tests d'enchainement (TE) : ils vérifient que la phase qui a été modifiée ne provoque aucun effet de bord sur le reste de l'acte de gestion

Recette : ensemble des tests réalisés par les utilisateurs ou leurs représentants dans le but de valider le produit avant sa mise à disposition, et faire ressortir les anomalies s'il y en a

Elle se décompose en 1 ou plusieurs plans de tests décrivant comment les tests vont se dérouler, en s'appuyant sur des jeux d'essais organisant dans le temps un certain nombre de cas de tests, qui sont les cas permettant de tests l'ensemble de la couverture fonctionnelle de l'outil

Tests de pré-production : vérifie la tenue du produit dans un environnement similaire à la production, en terme d'usage et de montée en charge

Production : utilisation du produit

Organisation détaillée

Livrable	Responsable	Validé par	Quand
Expression des besoins (EB)	Responsable utilisateurs	Tous les services utilisateurs	Avant-Projet – Opportunité
Cahier des charges (CdC)	Chef de projet MOE (CP MOE)	Chef de projet utilisateurs (CP MOA)	Avant-Projet – Faisabilité
Spécifs fonctionnelles détaillées	CP MOA	Responsable utilisateurs	Conception
Specifs techniques générales	Analyste	CP MOE	Conception
Spécifs fonctionnelles générales	CP MOA	Responsable utilisateurs	Conception / Réalisation
Specifs techniques détaillées	Développeur / Opérateur	CP MOE	Conception / Réalisation
Développement / réalisation	Responsable technique	CP MOE	Réalisation
Tests unitaires	Responsable technique	Responsable technique	Réalisation
Tests intégration	Analyste	CP MOE	Réalisation
Tests de validation / enchainement	CP MOA	CP MOA	Réalisation
Recette	Responsable utilisateurs	Tous les services utilisateurs	Réalisation
Tests pré-production	Responsable technique	CP MOE	Mise en place
Préparation mise en prod	CP MOE	CP MOA	Mise en place
Mise en production	CP MOE	CP MOA	Mise en place
Retrait du service	CP MOE	CP MOA	Démantèlement
Tests de livraison	Responsable utilisateurs	Tous les services utilisateurs	Exploitation

Risques inhérents à la méthode en cascade

Une fois l'ensemble des besoins formalisés et les spécifications établies, il arrive que des difficultés organisationnelles, techniques, et/ou humaines, apparaissent en phase réalisation remettant en cause les analyses précédentes : il est en effet difficile, voire impossible, de totalement détacher la phase de conception d'un projet de sa phase de réalisation.
C'est souvent au cours de l'implémentation qu'on se rend compte que les spécifications initiales étaient incomplètes, fausses, ou irréalisables.

De plus le besoin évolue dans le temps : l'organisation des projets « longue durée » cherchant la satisfaction du besoin dans sa totalité n'est pas adaptée aux environnements changeant à fort besoin de réactivité, et ne permet ni la fourniture rapide minimale d'un service ni de s'adapter aux dérives potentielles du projet.

C'est suite à ce constat qu'ont été progressivement introduites les méthodes d'organisation par lotissements, par cycles en spirale, puis par processus semi-itératifs et itératifs (méthodes agiles) : l'idée sera de procéder à des « mini cycles en V » (ou mini projets) sur la partie Conception – Réalisation – Livraison

Le cycle en spirale

Le développement reprend les différentes étapes du cycle en V.

Par l'implémentation de versions successives intégrant de nouvelles fonctionnalités et des améliorations, le cycle recommence en proposant un produit de plus en plus complet et dur.

Le cycle en spirale met cependant plus l'accent sur la gestion des risques que le cycle en V : en effet, le début de chaque itération comprend une phase d'analyse des risques.
Ceci est rendu nécessaire par le fait que, lors d'un développement cyclique, il y a plus de risques de devoir défaire à la Ne itération ce qu'on a fait à la N-Xe.

Les cycles semi-itératifs et itératifs, les méthodes agiles

Le cycle semi-itératif a pour origine les travaux de James Martin publiés à partir de 1989 dans diverses revues Nord-Américaines. Il fut totalement formalisé en 1991.

Dans le cycle semi-itératif les deux premières phases classiques (top down, par la structure) consistent en l'expression des besoins et la conception de la solution.

C'est lors de la troisième et dernière grande phase, la construction du produit (bottom up, par le besoin) que la notion d'itérations courtes intervient.

C'est vers 2001 avec l'apparition de plusieurs méthodes dont ASD, FDD, Crystal, Scrum ou Extreme Programming et la vision uniformisée de leurs auteurs dans le cadre du Manifeste Agile (Agile Manifesto) et de l' Agile Alliance, que le cycle itératif fut généralisé.

La notion d'itérativité totale est cependant théorique, car toutes ces approches débutent par deux phases séquentielles, courtes mais bien réelles, d'exploration et de planning.

Gestion agile

Dans l'approche de gestion de projet traditionnelle, un projet est identifié, évalué, découpé en tâches et précisément planifié.

Une autre approche est de considérer l'ensemble de l'équipe comme une usine, c'est-à-dire comme un ensemble de postes de travail, de machines, dont les activités sont inter-reliées.
On y injecte les besoins du client comme matière première, qui sera progressivement transformé en produit final.

Par exemple, on tente de réduire ce qu'on appelle l'inventaire des "en cours".
Dans le cas du développement logiciel, cet inventaire est constitué des besoins du client exprimés, mais non encore livrés. Cela peut être des spécifications écrites, d'architecture, de conception, du code, etc., non encore livrés en un logiciel fonctionnel.
Il s'agit d'argent investi, qui dort dans l'entrepôt du logiciel. Les méthodes agiles tentent de réduire cet inventaire, en permettant au client de changer d'idée facilement.

es méthodes agiles sont les groupes de pratiques associées aux cycles semi-itératifs et itératifs, pouvant s'appliquer à divers types de projets, mais se limitant plutôt actuellement aux projets de développement logiciel en informatique.

Elles impliquent au maximum le demandeur (client) et permettent une grande réactivité à ses demandes et visent la satisfaction réelle du besoin du client et non les termes d'un contrat de développement : elles se veulent plus pragmatiques que les méthodes traditionnelles.

Les méthodes agiles ont été formalisée dans le Manifeste agile par les auteurs de ces méthodes, ayant toutes une structure commune (cycle de développement itératif, incrémental et adaptatif) et une base de pratiques, soit communes, soit complémentaires.

Parmi ces méthodes on trouve en premier lieu la méthode RAD de James Martin (1991), puis DSDM, la version anglaise du RAD (1995).
Les deux méthodes agiles les plus connues en France sont : la méthode Scrum (1996) et la méthode XP, pour Extreme Programming (1999).

Grands principes

Agile = itératif, incrémental et adaptif

Une méthode agile est donc avant tout itérative sur la base d'un affinement du besoin mis en œuvre dans des fonctionnalités en cours de réalisation et même déjà réalisées.
Cet affinement, indispensable à la mise en œuvre du concept adaptatif, se réalise en matière de génie logiciel sous deux aspects :

- ⇨ fonctionnellement, par adaptation systématique du produit aux changements du besoin détecté par l'utilisateur lors de la conception-réalisation du produit
- ⇨ techniquement, par remaniement régulier du code déjà produit (refactoring).

Lorsque le projet, quel que soit le nombre de participants, dépasse en durée une dizaine de journées en moyenne, la production de ses fonctionnalités s'effectuent en plusieurs incréments.

De plus, le manifeste agile résume sa philosophie en quatre oppositions entre les concepts traditionnels et les concepts proposés.

Individus et interactions contre processus et outils

Ce sont les individus qui font la valeur du travail accompli, ce sont donc eux que l'on doit privilégier : sans l'artisan, les meilleurs outils ne servent à rien.
Selon les auteurs, les processus qui définissent ce que doit faire chaque personne bride le potentiel caché derrière chacun : faire interagir les gens au maximum est bien plus fructueux et permet d'améliorer grandement l'efficacité et la qualité du travail fourni, en rassemblant des visions différentes d'un même problème.

Logiciel qui fonctionne contre documentation exhaustive

Les processus lourds génèrent une documentation qui se veut exhaustive avec tous ses inconvénients : ambigüité du langage, coût de la rédaction, coût du maintien en accord avec la réalité, utilité réelle, etc.
Même une conception technique initiale peut être complètement remise en cause en phase de codage (ou après) : comment peut-on alors déterminer l'avancement du projet ? Une régression ?

Dans les méthodes Agiles, un seul critère permet de mesurer l'avancement d'un projet : le logiciel qui fonctionne. La documentation n'est qu'un support concret qui aide à produire et utiliser le logiciel.

Collaboration du client contre négociation de contrat

Dans tout projet, le but premier est de gagner de l'argent, autant pour le client (rentabilisation) que pour le fournisseur (prestation).
Si la négociation protège plus ou moins des risques financiers, elle peut provoquer l'échec des projets (délais non respectés, budgets insuffisants) et engendrer d'interminables procès où tout le monde y perd au bout du compte (le client n'a pas son logiciel et le fournisseur ferme boutique).

Il faut sortir de la guerre client/fournisseur et penser en équipe qui veut atteindre un but commun : réussir le projet.

Réponse au changement contre suivi d'un plan prédéfini

Un plan prédéfini a tendance à nous rendre autistes aux événements qui surviennent pendant le projet.
Il est en plus à l'origine des conflits client/fournisseur classiques sur les délais de livraison ou respect des besoins (vs les exigences initiales).
Pour le client, pouvoir adapter les besoins en cours de projet est un atout concurrentiel : il est réactif aux fluctuations des marchés et s'assure en plus que le logiciel développé répond parfaitement à ses véritables besoins.

Les méthodes Agiles sont conçues pour s'adapter au changement, en assurant un plan macroscopique précis et adaptatif.

Les idées clés

- ⇨ Le client au cœur du projet
- ⇨ L'esprit d'équipe
- ⇨ La communication
- ⇨ La simplicité, l'efficacité et la qualité
- ⇨ La flexibilité aux changements
- ⇨ L'avancement basé sur le concret

Tronc des pratiques communes à l'ensemble des méthodes agiles

Les pratiques communes liées aux ressources humaines

- ⇨ Participation de l'utilisateur final aux groupes de travail.
- ⇨ Groupes de travail disposant du pouvoir de décision.
- ⇨ Autonomie et organisation centralisée de l'équipe.
- ⇨ Spécification et validation permanente des Exigences.

Les pratiques communes liées au pilotage du projet

- ⇨ Niveau méthodologique variable en fonction des enjeux du projet.
- ⇨ Pilotage par les enjeux et les risques.
- ⇨ Planification stratégique globale basée sur des itérations rapides.
- ⇨ Réalisation en jalons par prototypage actif itératif et incrémental.
- ⇨ Recherche continue d'amélioration des pratiques.

Les pratiques communes liées à la qualité de la production

- ⇨ Recherche d'excellence technique de la conception.
- ⇨ Vision graphique d'une modélisation nécessaire et suffisante.
- ⇨ Vision de la documentation nécessaire et suffisante.
- ⇨ Normes et techniques de qualité raisonnables.
- ⇨ Architecture à base de composants.
- ⇨ Gestion des changements automatisée.

Méthodologie (vision Celerial)

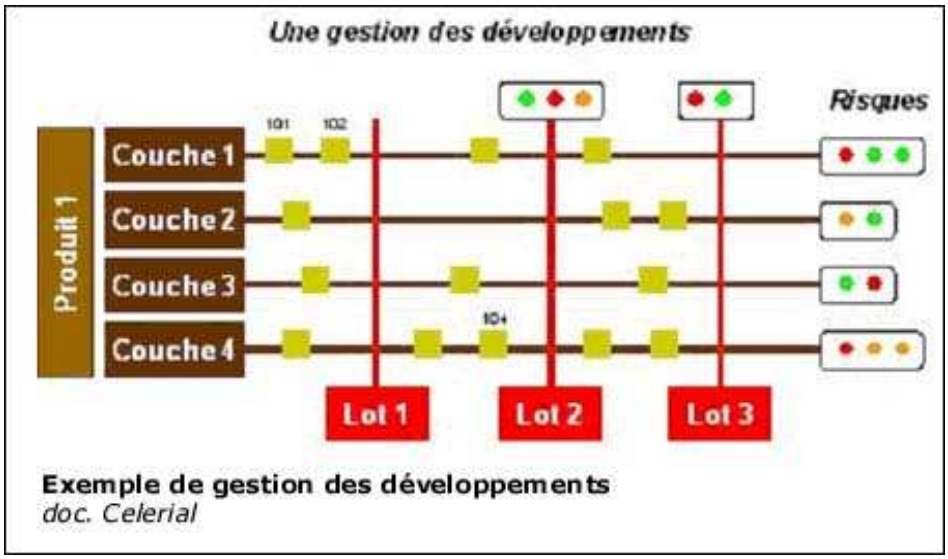
Voici quelques avancées méthodologiques significatives apportées par l'Agilité en matière d'état de l'art de méthodologie de gestion de projet :

- ⇨ le projet agile ne se focalise plus sur le produit à réaliser mais d'avantage sur le service à rendre.
  Le maître d'œuvre n'est donc plus engagé sur un livrable *clés en main* mais sur le respect des services qu'il met à disposition de son client pour réaliser le produit.
  Son engagement est traduit dans une convention de service et non dans un contrat d'intégration au forfait par exemple.

Exemple de séquencement des mises à disposition
doc. Celerial

- ⇨ l'utilisateur ou son représentant est présent au cœur même des plateaux de développement pour apporter la connaissance métier de l'entreprise au bon moment et intégrer dans sa réflexion des contraintes qu'il ne mesurait jusqu'alors que par les termes *çà coûtera plus* et *ce sera pour plus tard*.

- ⇨ un projet agile ne se gère plus par les délais mais par les tassements et les glissements. En d'autres termes, il est plus important de conserver la possibilité de déplacer un travail ou d'en modifier son contenu que de la réaliser à une date précise au détriment du reste. Dans ce contexte, les diagrammes de Gant ou de Pert sont remplacés par une planification fonctionnelle.



Exemple de gestion des développements
doc. Celerial

⇨ des cartographies des produits et des services sont réalisées et placées directement sur des pans *muraux*. Ce type de représentation permet en *un coup d'œil* de comprendre la dimension du projet et de suivre son avancement, ce qui est impossible à réaliser avec les outils de planification existants. De plus, ainsi réalisées, ces cartographies deviennent d'excellents outils de communication et de pilotage de projet.

Représentation murale du projet
doc. Celerial

⇨ l'étape de test est inversée dans le déroulement du projet et remplace l'étape de spécification des méthodes traditionnelles.
Cette façon de procéder garantit l'automatisation des tests de non régression indispensable au mécanisme d'intégration continue (cf. ci-après) et évite ainsi la mise à jour récurrente des spécifications, lourdeur bien connue des méthodes traditionnelles.

⇨ l'organisation des équipes change complètement afin de garantir une responsabilité globale du projet.
Il n'y a plus de cloisonnement des responsabilités, plus de séparation tayloriste des rôles.

⇨ la gestion du temps est différente d'un projet traditionnel. En particulier, la méthode s'appuie sur le travail en binôme.
Ce point est fondamental et est assez "révolutionnaire" : il s'appuie sur le constat simple que 2 personnes de même niveau ont une capacité de production supérieure à 200% du fait du partage des idées et de la sollicitation intellectuelle qu'elles s'auto-génèrent entre elles.

⇨ le projet est rythmé par des cycles de livraison de l'ordre de quelques semaines, voire quelques jours.
Ce mécanisme de livraison fréquente allant jusqu'à l'intégration continue permet de fournir à la Maîtrise d'Ouvrage des informations claires sur les niveaux de maturité des produits et constitue un facteur clé en matière de prise en compte des changements.

Bénéfices

Sur le plan quantitatif, même s'il est difficile de définir un gain financier, on estime que les bénéfices sont de l'ordre de 10 à 15% pour l'ensemble du projet.
Cette estimation prend en compte notamment la disparition ou de l'allégement des tâches consommatrices de temps et ayant peu de valeur ajoutée :

- la diminution du nombre d'avenants,
- la baisse des coûts de tests de non-régression (par l'automatisation des tests),
- la baisse des coûts de maintenance et d'apprentissage (par le maintien de la qualité du code).

Sur le plan qualitatif, on gagne surtout en satisfaction du client interne : les difficultés qu'il éprouvait à exprimer son besoin complet à l'entrée du projet sont fortement diminuées par le mécanisme d'intégration continue.
A cela, s'ajoute la satisfaction d'avoir des résultats tangibles et directement exploitables très rapidement.
Coté Maîtrise d'œuvre, la responsabilité collective est un facteur clé de réussite.

Il n'existe plus ce cloisonnement des responsabilités qui induisait une certaine forme de désolidarisation entre membres de l'équipe opérationnelle.
Ce facteur humain est extrêmement important : on limite l'usage de parapluies pour se protéger ou justifier de tel ou tel choix, c'est l'efficacité qui prime avant tout.

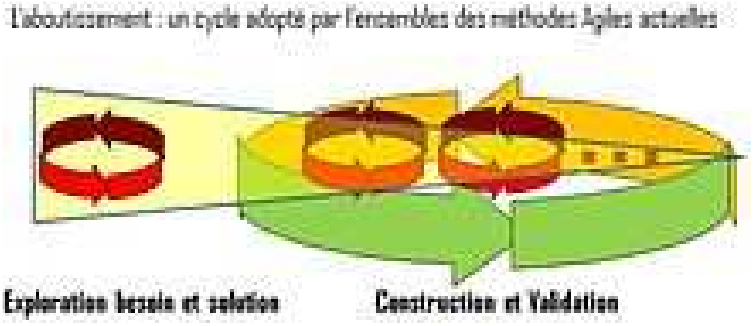
A noter

A l'usage, cette approche se combine avec la méthode RAD – XP (eXtreme Programming).

Exemple de cycle semi-itératif : la méthode RAD

La méthode RAD (pour Rapid Application Development) est la première méthode de développement de logiciels où le cycle de développement est en rupture fondamentale par rapport à celui des méthodes antérieures dites en cascade.

Principes de planification

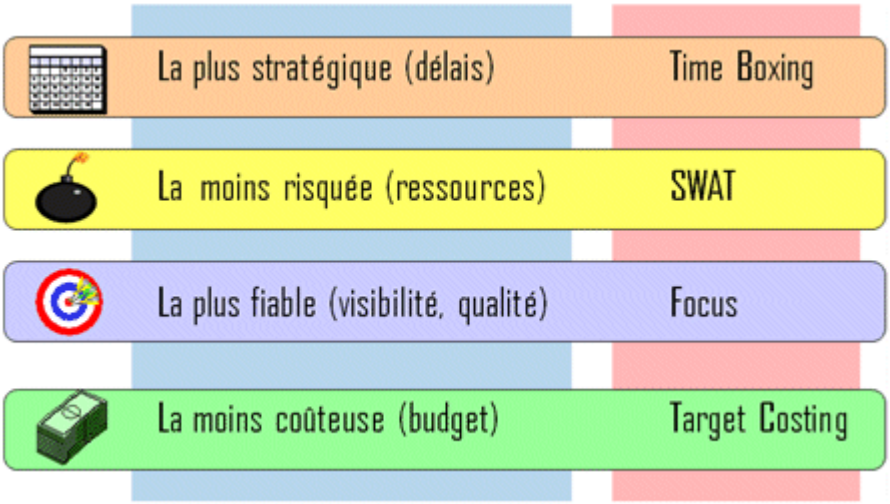


Après deux courtes phases de formalisation structurée de l'expression des besoins (CADRAGE) et de définition globale de l'architecture technique (DESIGN), cette méthode s'appuie sur une phase principale (CONSTRUCTION) de réalisation, validation immédiate et tests d'une application en mode itératif-incrémental-adaptatif.
L'objectif de la méthode, qui implique activement l'utilisateur final dans un principe de "validation permanente", est d'obtenir un applicatif en adéquation avec les besoins réels.

La planification adaptative de la méthode RAD répondait, à l'origine, aux besoins de projets simples : elle se limitait généralement à jouer sur les fameux paramètres sur lesquels s'appuie toute gestion de projet (qui restaient fixes dans les méthodes cascades), à savoir : performance, coût, délai, qualité, le but étant de fixer au moins l'un de ces paramètres en fonction du besoin immédiat de l'utilisateur (valeur ajoutée).

Cette planification était qualifiée d'opérationnelle et était modifiable par l'utilisateur en cours de projet.

Un niveau supérieur de planification (dite stratégique) fut ajouté par la suite par Jean-Pierre Vickoff (processus RAD2 publié par le Gartner Group). La figure suivante en décrit les principes.



Structure de la méthode

Le RAD préconise la formation d'une équipe de développement particulière, de profils techniques complémentaires : le SWAT team.

Cette équipe est autonome, et se compose essentiellement de concepteurs-développeurs possédant des compétences particulières : les modélisateurs doivent maitriser les AGL et être en mesure de modéliser le système en direct lors des réunions, par exemple.

L'équipe travaille avec les utilisateurs et, généralement avec un animateur, dans une salle spéciale, isolée, spécialement équipée dans le style war room, où les murs sont utilisés pour afficher un "radiateur d'information" (une forme de cockpit de management de projet).

Outre l'animateur (essentiel pour les réunions, neutre, qui maitrise les techniques d'entretien de groupe et les principes du RAD), l'équipe doit désigner un rapporteur-secrétaire, qui présente en direct lors des réunions et points d'avancement la synthèse des décisions et les fait valider.

Le rôle de chef de projet, n'est ni prohibé, ni obligatoire. Par contre, les décisions concernant l'organisation du projet sont consensuelles.

Sur le plan des principes de mise en opération, la méthode RAD implique :

- Un cycle de développement court fondé sur un phasage simple (cadrage, design, construction) et l'absolu respect d'une dimension temporelle (90 jours optimum, 120 jours maximum) [Martin 1991]

- Une architecture de communication engageant des groupes de travail de structure et de composition variables selon les besoins de chaque phase [Mucchielli 1987].

- Des méthodes, techniques et outils permettant de définir et d'appliquer des choix portant sur quatre natures d'objectifs potentiellement contradictoires : budget, délais, qualité (technique et fonctionnelle) et visibilité [Vickoff 1999].

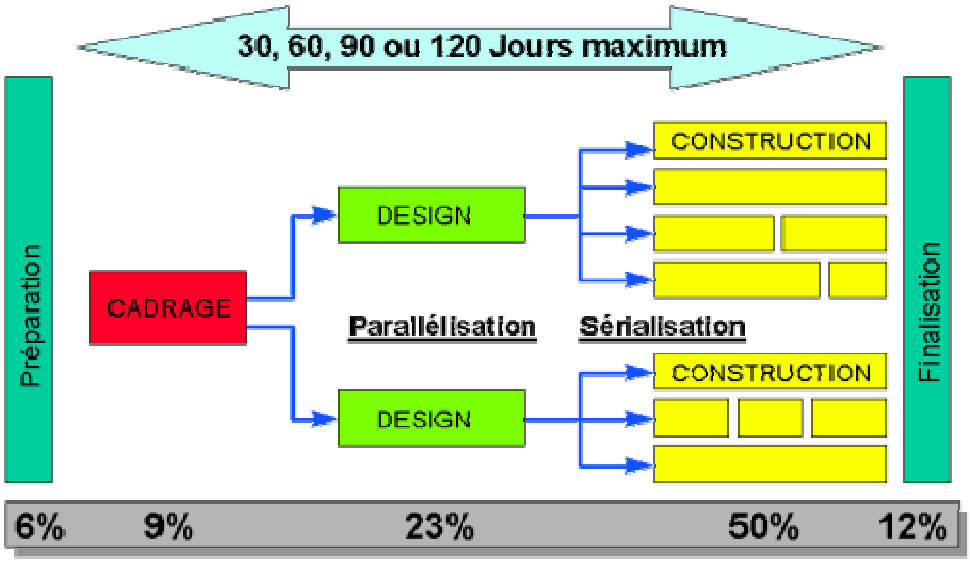
- Une architecture de conception s'appuyant sur les techniques de l'objet et particulièrement sur celles qui permettent une conception «en vue de modifications» [McCarty 1997].

- Une architecture de réalisation qui impose des normes minimales, des revues de projet et des jalons « zéro-défaut » pour garantir la qualité technique, et qui recommande le prototypage actif et les *focus* de visibilité pour garantir la qualité fonctionnelle [McConnell 1996].

Description des phases

La méthode RAD structure le cycle de vie du projet en 5 phases (dont 3 systématiques) :

1. La préparation (ou initialisation) prépare l'organisation, puis détermine le périmètre et le plan de communication.
2. Le cadrage définit un espace d'objectifs, de solutions et de moyens.
3. Le design modélise la solution et valide sa cohérence systémique.
4. La construction réalise en prototypage actif (validation permanente).
5. La finalisation correspond à un contrôle final de qualité en site pilote.



Initialisation

Préparation de l'organisation et communication.

Cette phase permet de :

- ⇨ définir le périmètre général du projet
- ⇨ structurer le travail par thèmes
- ⇨ sélectionner les acteurs pertinents
- ⇨ amorcer une dynamique de projet.

Elle représente environ 6 % du projet en charge.

Cadrage

Analyse et expression des exigences.

La spécification des exigences est du ressort des utilisateurs.
Ils expriment leurs besoins lors d'entretiens de groupe.

Il est généralement prévu de 2 à 5 jours de sessions par commission (thème).
Cette phase représente environ 9 % du projet.

Design

Conception et modélisation.

Les utilisateurs sont également impliqués dans cette étape.
Ils participent à l'affinage et à la validation des modèles organisationnels : flux, traitements, données.
Ils valident également le premier niveau de prototype présentant l'ergonomie et la cinématique générale de l'application.

Il est prévu entre 4 et 8 jours de sessions par commission.
Cette phase représente environ 23 % du projet.

À partir de la phase de Design, la parallélisation du travail est possible.

Construction

Réalisation, prototypage.

Durant cette phase, l'équipe RAD (SWAT) doit construire l'application module par module.

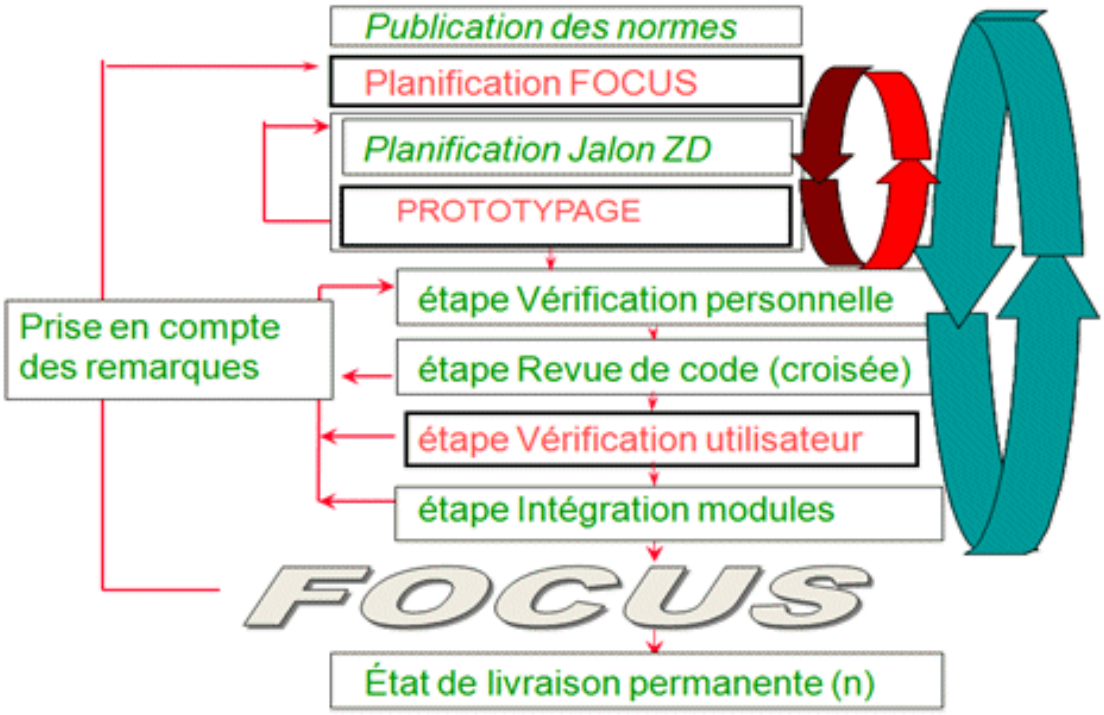
L'utilisateur participe toujours activement aux spécifications détaillées et à la validation des prototypes.
Plusieurs sessions itératives sont nécessaires.
Cette phase représente environ 50 % du projet. À partir de la phase de construction, à la parallélisation du travail peut s'ajouter la sérialisation.

Finalisation

Recette et déploiement.

Des recettes partielles ayant été obtenues à l'étape précédente, il s'agit dans cette phase d'officialiser une livraison globale et de transférer le système en exploitation et maintenance. Cette phase représente environ 12 % du projet.

Le fonctionnement du principe itératif, incrémental et adaptatif



Le Jalon ZD (Zéro Défaut) est une intégration partielle ou totale de l'itération, validée techniquement et fonctionnellement.
Le FOCUS (ou SHOW) est une présentation de l'itération de livraison venant de s'achever.
Cette démonstration est effectuée par le ou les utilisateurs impliqués dans le prototypage à destination de l'ensemble des autres intervenants du projets.

Les fondamentaux de la méthode

Le RAD a pour objectifs principaux la qualité et la réduction des temps de développement en charge (gain d'argent) et en délais (gain de temps).
Le RAD impose naturellement une motivation des futurs «clients» de l'application.
Le RAD permet aux utilisateurs et aux informaticiens d'aborder conjointement l'évolution d'une organisation à travers ses projets.
Le résultat est un outil conçu par les utilisateurs, réalisé sous leurs directives, en conformité avec leurs attentes.

Cette approche n'encourt pas le rejet de l'outil par l'utilisateur et, à lui seul, ce point justifie l'utilité du RAD.

Le RAD ne remplace pas la modélisation, mais choisit une forme de modélisation adaptée au type de projet

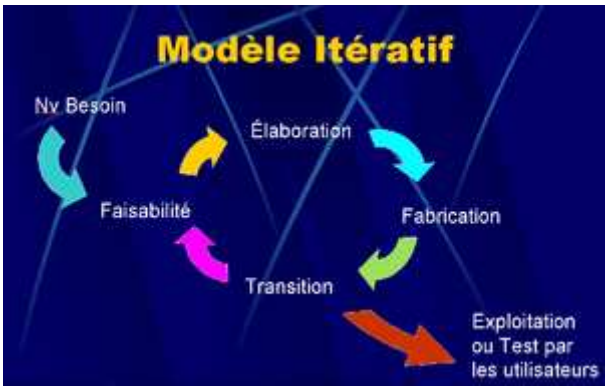
Avec le RAD l'aspect documentaire diffère des techniques classiques : le niveau de décomposition et de détail s'adapte et s'affine au cours des sessions successives.
Un spécialiste en documentation utilisateur encadre ou effectue sa production.
La documentation technique est suffisante, minimum et gérée directement dans l'AGL de conception ou dans le code des modules.

On optimise en planifiant des lots de 60, 90, 120 jours et en étant rigoureux sur le respect des jalons.

Pour plus d'info : voir http://www.rad.fr/

Le cycle itératif

Dans le cycle itératif, les spécifications détaillées et la conception de la solution sont intégrées aux cycles courts.



Sachant que chaque itération ne dépasse jamais huit semaines, l'idée est de livrer au plus tôt quelque chose qui puisse être testé par le client, le but n'étant pas de livrer un outil ou une fonctionnalité complète.
Pour chaque artéfact (produit issu d'une activité) à informatiser, on cherchera à appliquer la loi de Pareto : se concentrer sur les 20% de fonctionnalités qui couvrent 80% du périmètre fonctionnel restant à développer.

Ceci est vrai quel que soit l'artéfact : documentation, outil, …

Généralement un cycle itératif est une boucle avec les étapes suivantes :

- Faisabilité : acceptation d'un nouveau besoin
- Elaboration : choix du mode de réalisation
- Fabrication : construction de l'outil
- Transition : livraison au client

Exemple: l'eXtreme Programming (XP)

L'eXtreme Programming a été inventée par Kent Beck, Ward Cunningham et Ron Jeffries pendant leur travail sur un projet « C3 » de calcul des rémunérations chez Chrysler en 1996.
La méthode est cependant née officiellement en octobre 1999 avec le livre eXtreme Programming Explained de Kent Beck.

Dans ce livre, cette méthode est définie comme :

- ⇨ une tentative de réconcilier l´humain avec la productivité ;
- ⇨ un mécanisme pour faciliter le changement social ;
- ⇨ une voie d'amélioration ;
- ⇨ un style de développement ;
- ⇨ une discipline de développement d´applications informatiques.

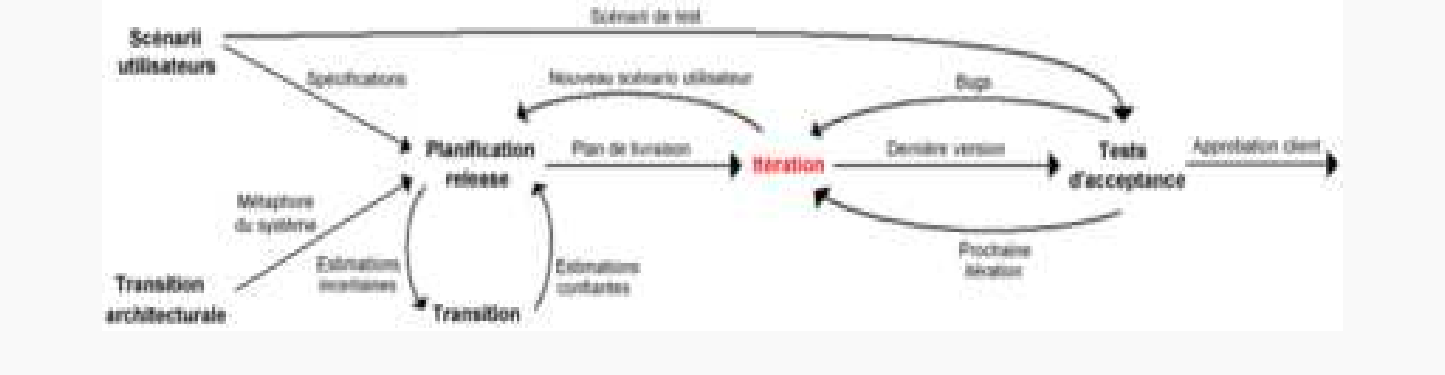
Son but principal est de réduire les coûts du changement : XP s´attache à rendre le projet plus flexible et ouvert au changement en introduisant des valeurs de base, des principes et des pratiques.

Les principes de cette méthode ne sont pas nouveaux : ils existent dans l'industrie du logiciel depuis des dizaines d'années et dans les méthodes de management depuis encore plus longtemps.

L'originalité de la méthode est de les pousser à l'extrême :

- ⇨ puisque la revue de code est une bonne pratique, elle sera faite en permanence (par un binôme) ;
- ⇨ puisque les tests sont utiles, ils seront faits systématiquement avant chaque implantation ;
- ⇨ puisque la conception est importante, elle sera faite tout au long du projet ;
- ⇨ puisque la simplicité permet d'avancer plus vite, nous choisirons toujours la solution la plus simple ;
- ⇨ puisque la compréhension est importante, nous définirons et ferons évoluer ensemble des métaphores ;
- ⇨ puisque l'intégration des modifications est cruciale, nous l'effectuerons plusieurs fois par jour ;
- ⇨ puisque les besoins évoluent vite, nous ferons des cycles de développement très rapides pour nous adapter au changement.

Cycle de développement



L'eXtreme Programming repose sur des cycles rapides de développement (des itérations de quelques semaines) dont les étapes sont les suivantes :

- ⇨ une phase d'exploration détermine les scénarios clients (users stories) qui seront fournis pendant cette itération ;
- ⇨ l'équipe transforme les scénarios en tâches à réaliser et en tests fonctionnels ;
- ⇨ chaque développeur s'attribue des tâches et les réalise avec un binôme ;
- ⇨ lorsque tous les tests fonctionnels passent, le produit est livré.

Le cycle se répète tant que le client peut fournir des scénarios à livrer.

Généralement le cycle de la première livraison se caractérise par sa durée et le volume important de fonctionnalités embarquées.

Après la première mise en production, les itérations peuvent devenir plus courtes (une semaine par exemple).

La programmation comme discipline collective

Tout en mettant l'accent sur les bonnes pratiques de programmation, XP préconise un déroulement par itérations courtes et gérées collectivement, avec une implication constante du client.
Il en découle une redéfinition de la relation entre client et fournisseur avec de surprenants résultats en termes de qualité de code, de délais et de satisfaction de la demande du client.

Valeurs

L'eXtreme Programming repose sur cinq valeurs fondamentales :

- ⇨ La communication

C'est le moyen fondamental pour éviter les problèmes.

Les pratiques que préconise l'XP imposent une communication intense.
Les tests, la programmation en binôme et le jeu du planning obligent les développeurs, les décideurs et les clients à communiquer.
Si un manque apparait malgré tout, un coach se charge de l'identifier et de remettre ces personnes en contact.

⇨ La simplicité

La façon la plus simple d'arriver au résultat est la meilleure : anticiper les extensions futures est une perte de temps.

Une application simple sera plus facile à faire évoluer.

⇨ Le feedback

Le retour d'information est primordial pour le programmeur et le client.
Les tests unitaires indiquent si le code fonctionne.
Les tests fonctionnels donnent l'avancement du projet.
Les livraisons fréquentes permettent de tester les fonctionnalités rapidement.

⇨ Le courage

Certains changements demandent beaucoup de courage. Il faut parfois changer l'architecture d'un projet, jeter du code pour en produire un meilleur ou essayer une nouvelle technique. Le courage permet de sortir d'une situation inadaptée. C'est difficile, mais la simplicité, le feedback et la communication rendent ces tâches accessibles.

⇨ Le respect

Pratiques

Ces cinq valeurs se déclinent en treize pratiques qui se renforcent mutuellement :

⇨ Client sur site

Un représentant du client doit être présent pendant toute la durée du projet.
Il doit avoir les connaissances de l'utilisateur final et avoir une vision globale du résultat à obtenir.
Il réalise son travail habituel tout en étant disponible pour répondre aux questions de l'équipe.

⇨ Jeu du Planning ou Planning poker

L'estimation se fait en unités d'œuvre intitulées points de récits ou "journées idéales" (Ideal Day) (ou heures idéale en cas de faible granularité des tâches).

La suite de Fibonacci est utilisée pour les évaluations. Le paquet de cartes utilisé pour le planning poker doit donc comporter les valeurs suivantes : 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. Certains simplifient les grandes valeurs en les transformant en 20, 40, 100.

Il est à noter que plus le scénario est gros, moins l'évaluation est précise.

Déroulement

1. Les participants s'installent autour d'une table, placés de façon à ce que tout le monde puisse se voir.
2. Le responsable de produit explique à l'équipe un scénario utilisateur (user story) : une phrase simple dans le langage de tous les jours permettant de décrire avec suffisamment de précision le contenu d'une fonctionnalité à développer, contenant généralement trois éléments descriptifs de la fonctionnalité : Qui ? Quoi ? Pourquoi ?.
3. Les participants posent des questions au responsable de produit, discutent du périmètre du scénario, évoquent les conditions de satisfaction qui permettront de le considérer comme "terminé".
   Les points de récits permettent d'obtenir une véritable mesure de complexité relative : l'équivalent en jours-hommes est propre à chacun, selon ses compétences, son expérience et sa connaissance du domaine.
4. Chacun des participants évalue la complexité de ce scénario, choisit la carte qui correspond à son estimation et la dépose, face vers le bas, sur la table devant lui.
5. Au signal du facilitateur, les cartes sont retournées en même temps.
6. S'il n'y a pas unanimité, la discussion reprend.
7. On répète le processus d'estimation jusqu'à l'obtention de l'unanimité.

Une procédure optimisée consiste, après la première "donne", de demander aux deux acteurs ayant produit les évaluations extrêmes d'expliquer leurs points de vue respectifs. Ces explications achevées et comprises de tous, une nouvelle estimation est produite et c'est alors la moyenne arithmétique de ces estimations qui est prise en compte.

(Voir http://fr.wikipedia.org/wiki/Planning_poker)

- ⇨ Intégration continue

Lorsqu'une tâche est terminée, les modifications sont immédiatement intégrées dans le produit complet.
On évite ainsi la surcharge de travail liée à l'intégration de tous les éléments avant la livraison.
Les tests facilitent grandement cette intégration : quand tous les tests passent, l'intégration est terminée.

- ⇨ Petites livraisons

Les livraisons doivent être les plus fréquentes possibles.
L'intégration continue et les tests réduisent considérablement le coût de livraison.

- ⇨ Rythme soutenable

L'équipe ne fait pas d'heures supplémentaires.
Si le cas se présente, il faut revoir le planning. Un développeur fatigué travaille mal.

- ⇨ Tests de recette (ou tests fonctionnels)

À partir des scénarios définis par le client, l'équipe crée des procédures de test qui permettent de vérifier l'avancement du développement.
Lorsque tous les tests fonctionnels passent, l'itération est terminée.
Ces tests sont souvent automatisés mais ce n'est pas toujours possible : en effet, seuls les tests de non régression peuvent être potentiellement automatisés du fait de leur récurrence.
La recette fonctionnelle d'une application est de plus en plus souvent confiée à des experts du test indépendants des développeurs.

- ⇨ Tests unitaires

Avant d'implémenter une fonctionnalité, le développeur écrit un test qui vérifiera que son programme se comporte comme prévu. Ce test sera conservé jusqu'à la fin du projet, tant que la fonctionnalité est requise. À chaque modification du code, on lance tous les tests écrits par tous les développeurs, et on sait immédiatement si quelque chose ne fonctionne plus.

- ⇨ Conception simple

L'objectif d'une itération est d'implémenter les scénarios sélectionnés par le client et uniquement cela. Envisager les prochaines évolutions ferait perdre du temps sans avoir la garantie d'un gain ultérieur.
Les tests permettront de changer l'architecture plus tard si nécessaire.
Plus l'application est simple, plus il sera facile de la faire évoluer lors des prochaines itérations.

- ⇨ Utilisation de métaphores

On utilise des métaphores et des analogies pour décrire le système et son fonctionnement.
Le fonctionnel et le technique se comprennent beaucoup mieux lorsqu'ils sont d'accord sur les termes qu'ils emploient.

- ⇨ Refactoring (ou remaniement du code)

Amélioration régulière de la qualité du code sans en modifier le comportement.
On retravaille le code pour repartir sur de meilleures bases tout en gardant les mêmes fonctionnalités.
Les phases de refactoring n'apportent rien au client mais permettent aux développeurs d'avancer dans de meilleures conditions et donc plus vite.

- ⇨ Appropriation collective du code

L'équipe est collectivement responsable de l'application.
Chaque développeur peut faire des modifications dans toutes les portions du code, même celles qu'il n'a pas écrites.
Les tests diront si quelque chose ne fonctionne plus.

- ⇨ Convention de nommage

Puisque tous les développeurs interviennent sur tout le code, il est indispensable d'établir et de respecter des normes de nommage pour les variables, méthodes, objets, classes, fichiers, etc.

- ⇨ Programmation en binôme

La programmation se fait par deux : le premier appelé driver (ou pilote) tient le clavier.
C'est lui qui va travailler sur la portion de code à écrire.
Le second appelé partner (ou co-pilote) est là pour l'aider en suggérant de nouvelles possibilités ou en décelant d'éventuels problèmes.
Les développeurs changent fréquemment de partenaire ce qui permet d'améliorer la connaissance collective de l'application et d'améliorer la communication au sein de l'équipe.

Autres principes

- ⇨ Ne pas ajouter de fonctionnalités plus tôt que prévu ;
- ⇨ N'optimiser qu'à la toute fin.

Cette méthode s'appuie sur :

- ⇨ une forte réactivité au changement des besoins du client ;
- ⇨ un travail d'équipe ;
- ⇨ la qualité du code ;
- ⇨ la qualité des tests effectués au plus tôt.

Environnements défavorables

En lieu et place d'inconvénient de la méthode, on parlera plus aisément d'environnements défavorables dans lesquels la méthode XP n'est pas applicable.
Dans ce cas, seule une partie des pratiques peut être réalisée.

Les principaux contextes défavorables sont :

- ⇨ un blocage culturel

Quand le client ou les développeurs ont l'habitude de fonctionner autrement.
L'ingénieur à qui l'on attribue des mini-tâches quotidiennes et qui doit les réaliser avec un binôme peut avoir le sentiment que sa fonction est déqualifiée.
Son principal rôle n'est plus d'être force de proposition, mais bel et bien de produire et d'accroitre sa productivité (Taylorisme, Fordisme).
Une telle vision peut en effet provoquer un blocage culturel qui peut conduire à un turn-over important.

- ⇨ une équipe de vingt développeurs ou plus car la communication devient difficile

Il est fréquent de voir au fil du temps des binômes travailler toujours ensemble, toujours sur les mêmes sujets, et former ainsi des sous-équipes spécialisées, ce que ne veut pas XP.
Par ailleurs, les évolutions possibles dans l'entreprise sont très limitées dans la mesure où tous les membres de l'équipe jouent tous les mêmes rôles.

- ⇨ un coût de changement exponentiel car « XP nécessite du code propre et simple »
- ⇨ un feedback long ou difficile à obtenir : le retour sur investissement est visible sur le moyen/long terme
- ⇨ un aménagement qui empêche la communication ou la programmation en binôme (il est nécessaire d'avoir une infrastructure open-space)
- ⇨ respect d'une discipline collective et travail en binôme au quotidien

Cette méthode ne peut pas être appliquée avec n'importe qui, car elle dégage très peu de temps à l'autonomie et au travail individuel, dans la mesure où systématiquement le travail se fait à deux sur un même poste de travail.
Les développeurs auront de la difficulté à établir un projet professionnel dans leur entreprise.

- ⇨ la résistance au changement.

Autre exemple : SCRUM

La méthode Scrum a été conçue lors de projets de développement de logiciels. Cependant elle peut aussi être utilisée par des équipes de maintenance.
Dans le cas de très grands projets, les équipes se multiplient et on parle alors de Scrum de Scrums.
La méthode Scrum peut théoriquement s'appliquer à n'importe quel contexte ou à un groupe de personnes qui travaillent ensemble pour atteindre un but commun comme planifier un mariage, gérer des projets de recherche scientifique, des écoles et même des églises comme le précise le site de son principal promoteur Jeff Sutherland.

En ingénierie logicielle, il faut adjoindre à la méthode Scrum un ensemble de pratiques circonstanciées en matière d'obtention et de contrôle de la qualité du logiciel ou des méthodes type Extreme Programming

Caractéristiques

Le terme Scrum signifie mêlée.
Ce processus s'articule en effet autour d'une équipe soudée, qui cherche à atteindre un but, comme c'est le cas en rugby pour avancer avec le ballon pendant une mêlée.

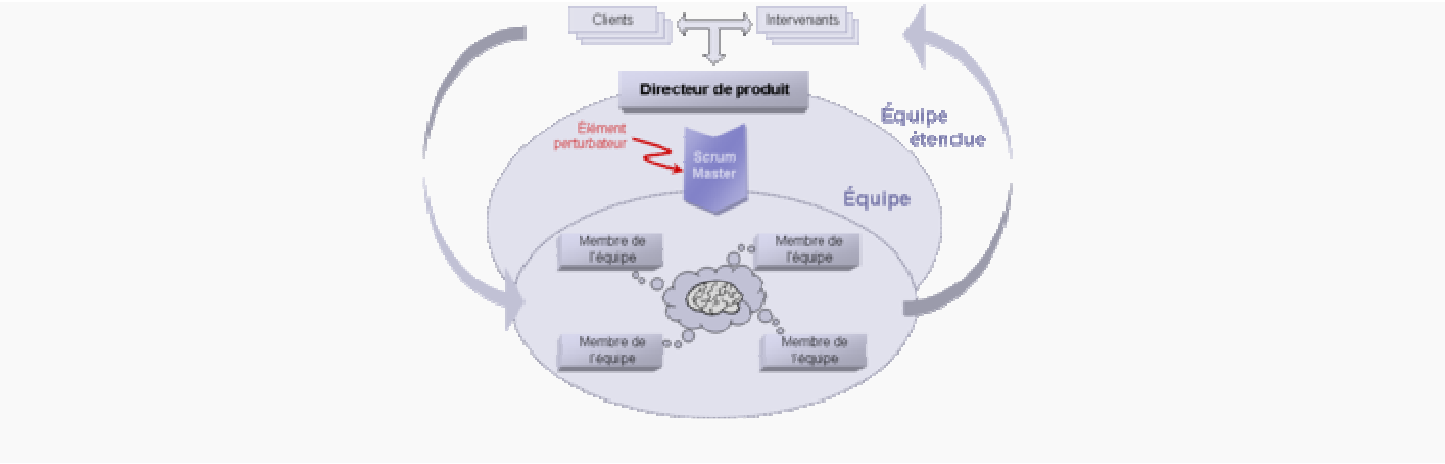
Le principe de base de Scrum est de focaliser l'équipe sur une partie limitée et maîtrisable des fonctionnalités à réaliser.

L'incrémentation des fonctionnalités (incrément) se réalisent successivement lors de périodes de durée fixe de une à quatre semaines, appelées sprints.

Chaque sprint possède, préalablement à son exécution, un but à atteindre, défini par le directeur de produit, à partir duquel sont choisies les fonctionnalités à implémenter dans cet incrément.
Un sprint aboutit toujours à la livraison d'un produit partiel fonctionnel.
Pendant ce temps, le ScrumMaster a la charge de minimiser les perturbations extérieures et de résoudre les problèmes non techniques de l'équipe.

Un principe fort en Scrum est la participation active du client pour définir les priorités dans les fonctionnalités du logiciel et pour choisir celles qui seront réalisées dans chaque sprint.
Il peut à tout moment compléter ou modifier la liste des fonctionnalités à produire, mais jamais celles qui sont en cours de réalisation pendant un sprint.
Cette interdiction va formellement à l'encontre du principe d'itération (revenir sur les caractéristiques de la fonctionnalité en cours de développement que le concept de "présence de l'utilisateur sur le site" permettrait de favoriser) et d'adaptabilité immédiate, telle la notion de conception émergente basée sur le feedback issu du travail en cours qui représente une des bases de l'Extrême Programming.

Rôles



Scrum définit trois rôles principaux : le directeur de produit, le facilitateur / animateur et l'équipe. Des intervenants peuvent s'intégrer également au projet de façon plus ponctuelle.

- ⇨ Directeur de produit

Le directeur de produit (Product Owner) est le représentant des clients et utilisateurs : c'est lui qui définit l'ordre dans lequel les fonctionnalités seront développées et qui prend les décisions importantes concernant l'orientation du projet.
Le terme directeur n'est d'ailleurs pas à prendre au sens hiérarchique du terme, mais dans le sens de l'orientation.

Dans l'idéal, le directeur de produit travaille dans la même pièce que l'équipe. Il est important qu'il reste très disponible pour répondre aux questions de l'équipe et pour lui donner son avis sur divers aspects du logiciel (interface par exemple).

- ⇨ Facilitateur / Animateur

Le facilitateur / animateur (ScrumMaster) joue un rôle capital : c'est lui qui est chargé de protéger l'équipe de tous les éléments perturbateurs extérieurs à l'équipe et de résoudre ses problèmes non techniques (administratifs par exemple).
Il doit aussi veiller à ce que les valeurs de Scrum soient appliquées, mais il n'est pas un chef de projet ni un intermédiaire de communication avec les clients.

On parle parfois d'équipe étendue, qui intègre en plus le ScrumMaster et le directeur de produit : ce concept renforce l'idée que client et fournisseur travaillent d'un commun effort vers le succès du projet.

- ⇨ L'équipe

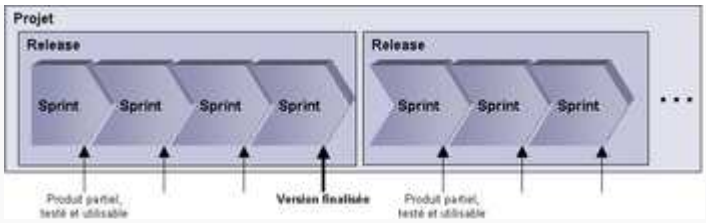
L'équipe ne comporte pas de rôles prédéfinis, elle est autogérée.
Il n'y a pas non plus de notion de hiérarchie interne : toutes les décisions sont prises ensemble et personne ne donne d'ordre à l'équipe sur sa façon de procéder.
L'équipe s'adresse directement au directeur de produit.
Il est conseillé qu'elle lui montre le plus souvent possible le logiciel développé pour qu'il puisse ajuster les détails d'ergonomie et d'interface par exemple.

- ⇨ Les intervenants

Les intervenants (Stakeholders) sont les personnes qui souhaitent avoir une vue sur le projet sans réellement s'investir dedans.

Il peut s'agir par exemple d'experts techniques ou d'agents de direction qui souhaitent avoir une vue très éloignée de l'avancement du projet.

Planification



Scrum propose une planification opérationnelle à trois niveaux : release/projet, sprint et quotidien.

⇨ Releases

Pour améliorer la lisibilité du projet, on regroupe généralement des itérations en releases.
Bien que ce concept ne fasse pas explicitement partie de Scrum, il est utilisé pour mieux identifier les versions.
En effet, comme chaque sprint doit aboutir à la livraison d'un produit partiel, une release permet de marquer la livraison d'une version aboutie, susceptible d'être mise en exploitation.

Il est intéressant de planifier à l'échelle d'une release, en répartissant les items du backlog de produit sur les sprints, en respectant leur priorité. Bien entendu, ce qui est planifié au-delà du sprint courant peut changer à tout moment, rien n'est figé à l'avance.

⇨ Sprints

Scrum est un processus itératif : les itérations sont appelées des sprints et durent en théorie 30 jours calendaires.
En pratique, les itérations durent généralement entre 2 et 4 semaines : chaque sprint possède un but et on lui associe une liste d'items de backlog de produit (fonctionnalités) à réaliser.
Ces items sont décomposés par l'équipe en tâches élémentaires de quelques heures, les items de backlog de sprint.

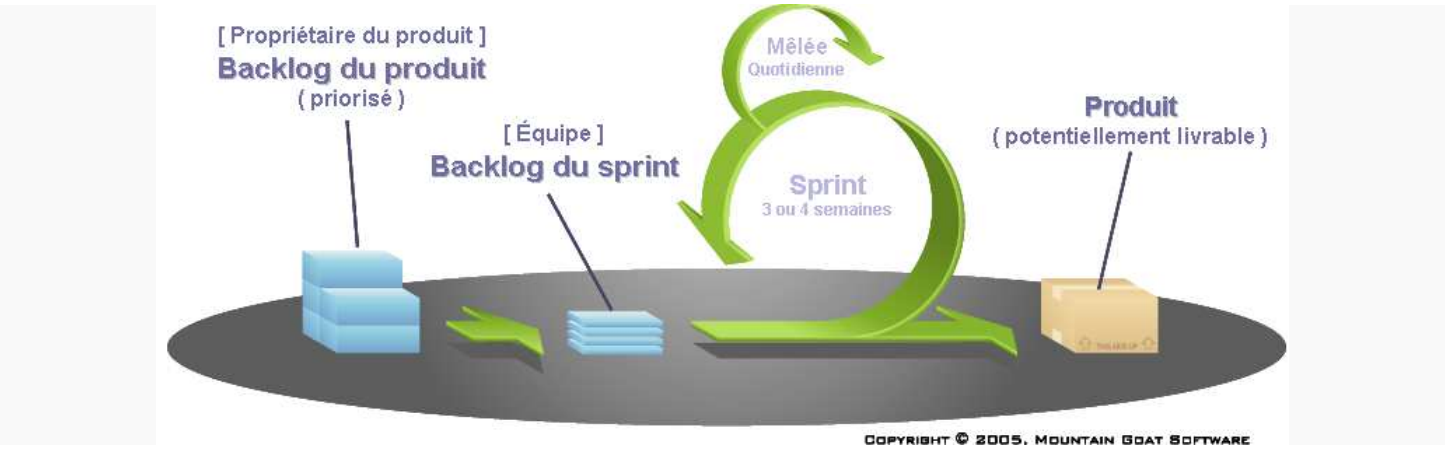
Pendant un sprint, les items de backlog de sprint à réaliser ne peuvent pas être changés : les changements éventuels sont pris en compte dans le backlog de produit et seront éventuellement réalisés dans les sprints suivants.

Il y a une exception à cela : il se peut que l'équipe se rende compte en cours du sprint qu'elle n'aura pas le temps de terminer un item du backlog de sprint ou au contraire qu'elle aura fini en avance.
Dans ce cas, et seulement d'un commun accord entre l'équipe et le directeur du produit, on peut enlever ou ajouter un item à ce qui a été prévu.

⇨ Quotidien

Au quotidien, une réunion, le ScrumMeeting (appelé également réunion Post-it), permet à l'équipe et au ScrumMaster de faire un point d'avancement sur les tâches et sur les difficultés rencontrées.

Gestion des besoins



⇨ Backlog de produit

Scrum utilise une approche fonctionnelle pour récolter les besoins des utilisateurs.
L'objectif est d'établir une liste de fonctionnalités à réaliser, que l'on appelle backlog de produit (le terme backlog peut être traduit par cahier, liste ou carnet de commandes, mais l'esprit du terme anglais évoque aussi une réserve, un retard accumulé ; aussi ce terme a été gardé tel quel).

À chaque item de backlog sont associés deux attributs : une estimation en points arbitraires et une valeur client, qui est définie par le directeur de produit (retour sur investissement par exemple).

Ce dernier définit dans quel ordre devront être réalisés ces items.
Il peut changer cet ordre en cours de projet et même ajouter, modifier ou supprimer des items dans le backlog.

La somme des points des items du backlog de produit constitue le reste à faire total du projet.
Cela permet de produire un release burndown chart, qui montre les points restant à réaliser au fur et à mesure des sprints.

Remarque : il arrive souvent qu'on utilise dans Scrum les User Stories de la méthode Extreme Programming, qui proposent des pratiques et des techniques intéressantes (le Planning poker pour les estimer par exemple).

Scrum présuppose que le backlog de produit est déjà défini au début du projet. Une approche possible pour constituer ce backlog est de réaliser une phase de lancement. Cette phase de lancement s'articule autour de deux axes de réflexion : l'étude davant-projet et l'expression initiale des besoins.

L'étude d'avant-projet est très variable d'un projet à l'autre, tout dépend du contexte de l'entreprise, de la nature du projet (sous-traitance ou interne), etc. : chaque entreprise a sa propre politique sur cette activité.

L'expression initiale des besoins n'est pas un élément défini dans Scrum et n'est surtout pas une activité de contractualisation d'un cahier des charges.
L'esprit d'une telle activité dans les méthodes Agiles est de définir d'une part le cadre du projet (pour que l'équipe s'imprègne du contexte métier) et d'autre part une première analyse globale des besoins.
L'objectif est d'identifier un maximum de fonctionnalités que le logiciel devra implémenter, en se limitant à un niveau de précision assez grossier.
On peut par exemple utiliser une approche par raffinages successifs, en partant des secteurs métiers concernés par l'application, puis en identifiant les activités métier, qu'on décompose en tâches métier qui correspondent à des fonctionnalités que l'on doit/peut ou non implémenter dans le logiciel final.

L'objectif pour Scrum est de produire la première version du backlog de produit.

⇨ Backlog de sprint

Lorsqu'on démarre un sprint, on choisit quels items du backlog de produit seront réalisés dans ce sprint.
L'équipe décompose ensuite chaque item en liste de tâches élémentaires (techniques ou non), chaque tâche étant estimée en heures et ne devant pas durer plus de 2 jours : on constitue ainsi le backlog de sprint.

Pendant le déroulement du sprint, chaque équipier s'affecte des tâches du backlog de sprint et les réalise.
Il met à jour régulièrement dans le backlog du sprint le reste à faire de chaque tâche.
Les tâches ne sont pas réparties initialement entre tous les équipiers, elles sont prises au fur et à mesure que les précédentes sont terminées.

La somme des heures des items du backlog de sprint constitue le reste à faire total du sprint.
Cela permet de produire un sprint burndown chart qui montre les heures restantes à réaliser au fur et à mesure du sprint.

Estimations

Scrum ne définit pas spécialement d'unités pour les items des backlogs. Néanmoins, certaines techniques se sont imposées de fait.

⇨ Items de backlog de produit

Les items de backlog de produit sont souvent des User Stories empruntées à Extreme Programming.
Ces Users Stories sont estimées en points relatifs, sans unité.
L'équipe prend un item représentatif et lui affecte un nombre de points arbitraire.
Cela devient un référentiel pour estimer les autres items.
Par exemple, un item qui vaut 2 points représente deux fois plus de travail qu'un item qui en vaut 1.
Pour les valeurs, on utilise souvent les premières valeurs de la suite de Fibonacci (1,2,3,5,8,13), qui évitent les difficultés entre valeurs proches (8 et 9 par exemple).

L'intérêt de cette démarche est d'avoir une idée du travail requis pour réaliser chaque fonctionnalité sans pour autant lui donner une valeur en jours que le directeur de produit serait tenté de considérer comme définitivement acquise.
En revanche, on utilise la vélocité pour planifier le projet à l'échelle macroscopique de façon fiable et précise.

⇨ Calcul de vélocité

Une fois que tous les items de backlog de produit ont été estimés, on attribue un certain nombre d'items à réaliser aux sprints successifs.
Ainsi, une fois un sprint terminé, on sait combien de points ont été réalisés et on définit alors la vélocité de l'équipe, c'est-à-dire le nombre de points qu'elle peut réaliser en un sprint.

En partant de cette vélocité et du total de points à réaliser, on peut déterminer le nombre de sprints qui seront nécessaires pour terminer le projet (ou la release en cours).
L'intérêt, c'est qu'on a une vision de plus en plus fiable (retours d'expérience de sprint en sprint) de la date d'aboutissement du projet, tout en permettant d'aménager les items de backlog du produit en cours de route.

⇨ Items de backlog de sprint

Les items de backlog de sprint sont généralement exprimés en heures et ne doivent pas dépasser 2 journées de travail, sinon il convient de les décomposer en plusieurs items. Par abus de langage, on emploie le terme de tâches, les concepts étant très proches.

Déroulement d'un sprint

⇨ Réunion de planification

Tout le monde est présent à cette réunion, qui ne doit pas durer plus de 4 heures.
La réunion de planification (Sprint Planning) consiste à définir d'abord un but pour le sprint, puis à choisir les items de backlog de produit qui seront réalisés dans ce sprint.
Cette première partie du sprint planning représente l'engagement de l'équipe.
Compte tenu des conditions de succès énoncées par le directeur de produit et de ses connaissances techniques, l'équipe s'engage à réaliser un ensemble d'items du backlog de produit.

Dans un second temps, l'équipe décompose chaque item du backlog de produit en liste de tâches (items du backlog du sprint), puis estime chaque tâche en heures.
Il est important que le directeur de produit soit présent dans cette étape, il est possible qu'il y ait des tâches le concernant (comme la rédaction des règles métier que le logiciel devra respecter et la définition des tests fonctionnels).

⇨ Au quotidien

Chaque journée de travail commence par une réunion de 15 minutes maximum appelée mêlée quotidienne (Daily Scrum).
Seuls l'équipe, le directeur de produit et le ScrumMaster peuvent parler, tous les autres peuvent écouter mais pas intervenir (leur présence n'est pas obligatoire).
A tour de rôle, chaque membre répond à 3 questions :

➔ Qu'est-ce que j'ai fait hier ?

➔ Qu'est-ce que je compte faire aujourd'hui ?

➔ Quelles sont les difficultés que je rencontre ?

Le tour de parole doit être scrupuleusement respecté pour éviter que le Scrum dérive sur des discussions techniques et déborde des 15 minutes.
Si le besoin s'en fait sentir, des discussions sont alors menées librement après le Scrum.

Cette réunion a un but de synchronisation pour l'équipe et ne doit pas être vécue comme un reporting d'activité.
C'est le niveau quotidien du principe « inspect and adapt » de Scrum.

L'équipe se met ensuite au travail : elle travaille dans une même pièce, dont le ScrumMaster a la responsabilité de maintenir la qualité d'environnement.
Les activités se déroulent éventuellement en parallèle : analyse, conception, codage, intégration, tests, etc. Scrum ne définit volontairement pas de démarche technique pour le développement du logiciel : l'équipe s'autogère et décide en toute autonomie de la façon dont elle va travailler.

Remarque : Il est assez fréquent que les équipes utilisent la démarche de développement guidé par les tests (Test Driven Development en anglais). Cela consiste à coder en premier lieu les modules de test vérifiant les contraintes métier, puis à coder ensuite le logiciel à proprement parler, en exécutant les tests régulièrement. Cela permet de s'assurer entre autres de la non-régression du logiciel au fil des sprints.

Analyse

- ⇨ Analyse du sprint

À la fin du sprint, tout le monde se réunit pour effectuer la revue de sprint, qui dure au maximum 4 heures.
L'objectif de la revue de sprint est de valider le logiciel qui a été produit pendant le sprint.
L'équipe commence par énoncer les items du backlog de produit qu'elle a réalisés.
Elle effectue ensuite une démonstration du logiciel produit.
C'est sur la base de cette démonstration que le directeur de produit valide chaque fonctionnalité planifiée pour ce sprint.

Une fois le bilan du sprint réalisé, l'équipe et le directeur de produit proposent des aménagements sur le backlog du produit et sur la planification provisoire de la release.
Il est probable qu'à ce moment des items soient ajoutés, modifiés ou réestimés, en conséquence de ce qui a été découvert.

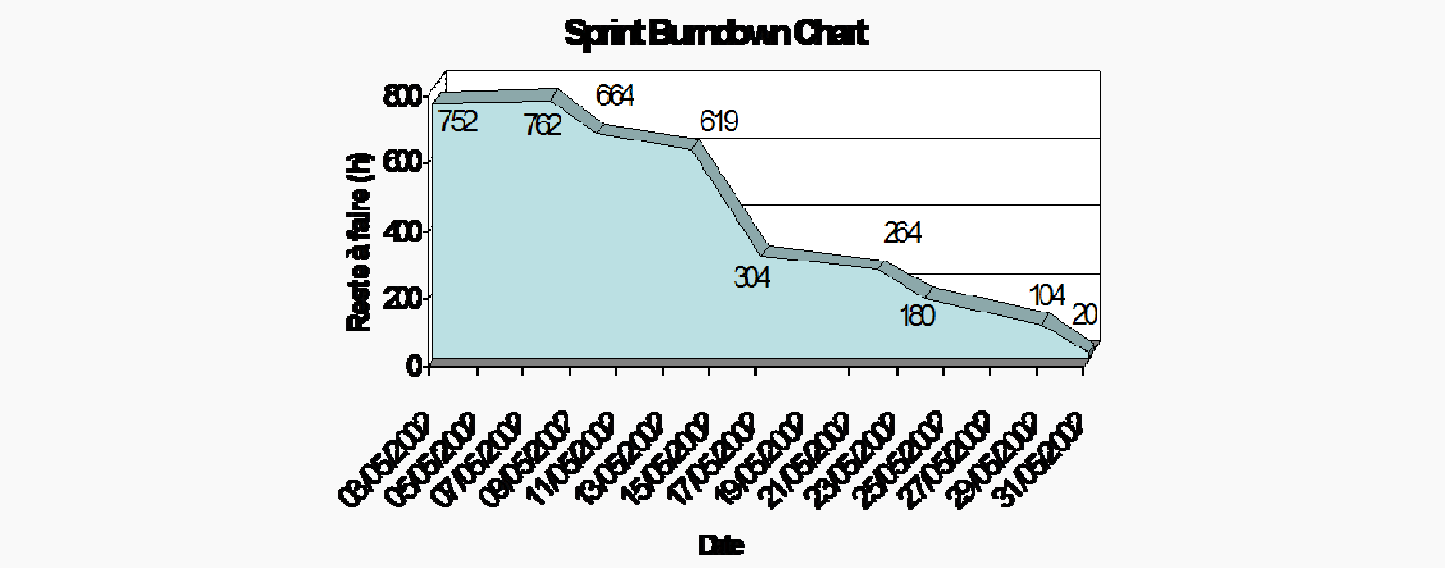
- ⇨ Rétrospective du sprint

La rétrospective du sprint est faite en interne à l'équipe (incluant le ScrumMaster) l'objectif est de comprendre ce qui n'a pas bien marché dans le sprint, les erreurs commises et de prendre des décisions pour s'améliorer.
Il est tout à fait possible d'apporter des aménagements à la méthode Scrum dans le but de s'améliorer.
Il faut être très vigilant à ne pas retomber dans des pratiques rigides des méthodologies plus classiques.

⇨ Burndown Charts

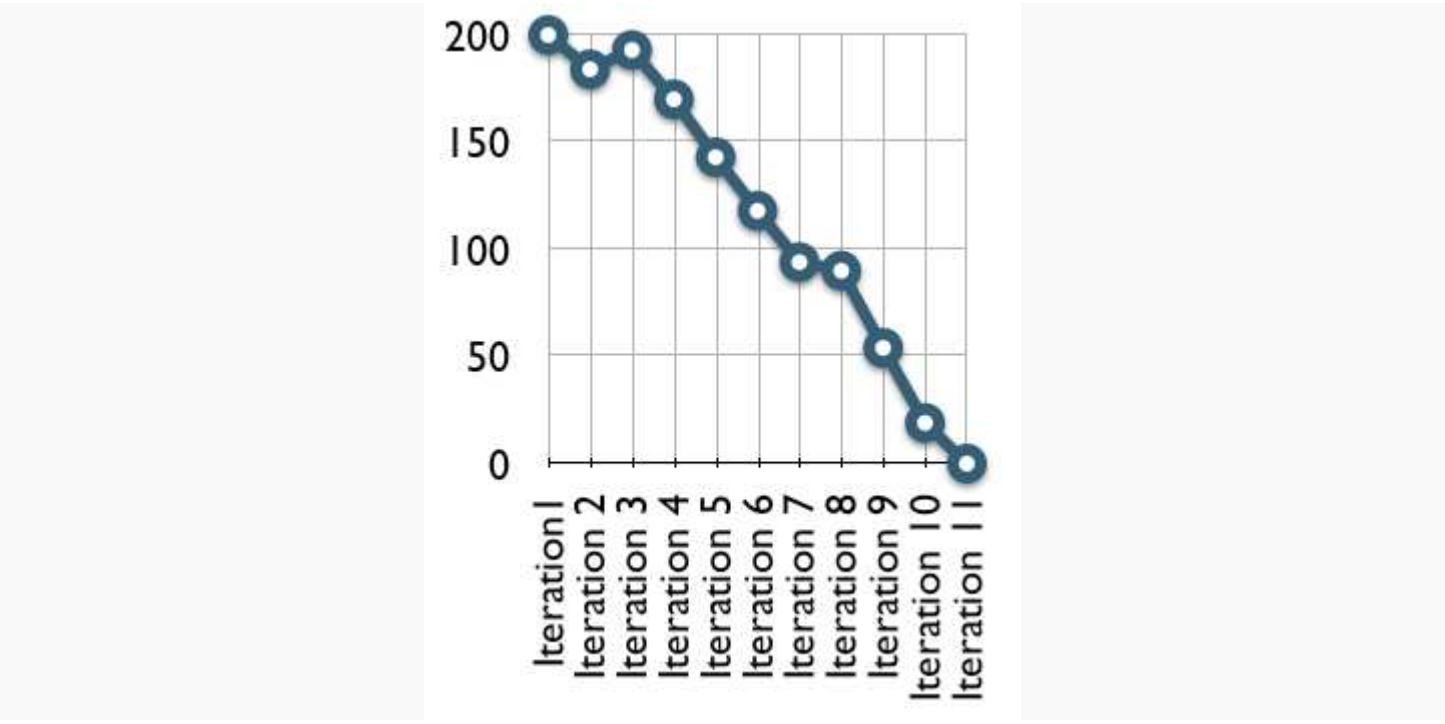
Les burndown charts (graphiques d'avancement) permettent de visualiser graphiquement l'avancement du travail.
Une interprétation simple permet d'avoir une idée sur les échéances futures.

- Sprint Burndown Chart



Ce graphique représente la quantité totale d'heures restantes à faire dans le sprint, au fil des jours. Il permet d'avoir une vue sur l'avancement du sprint.

- Release Burndown Chart



Ce graphique représente la quantité totale de points restant à faire dans la release, au fil des sprints. Il permet d'avoir une vue sur l'avancement de la release.

- Interprétation

Outre le fait de montrer l'avancement concret du travail, ces graphiques permettent d'anticiper de façon relativement fiable les échéances futures en cours du sprint ou de la release.

On peut observer de légères augmentations du reste à faire sur le burndown chart du sprint : cela correspond généralement à une réestimation à la hausse, suite à la prise en compte de contraintes techniques que l'on n'avait pas vues lors de l'estimation initiale.
Si c'est le cas, il est indispensable de comprendre la cause de ces augmentations.
Le même phénomène peut s'observer sur des légères diminutions, pour les mêmes raisons.

Ces graphiques donnent aussi la tendance du projet, et.donc la date a priori de fin du sprint, ce qui permet alors de prendre une décision sur la suppression (ou l'ajout) d'un item de backlog du produit à réaliser dans ce sprint.

C'est exactement la même chose pour les burndown charts de release.
Si la date de publication de la release est clairement au-delà de ce que l'on espérait, on peut aviser en enlevant des items de backlog du produit ou changeant leur ordre, de sorte que les fonctionnalités les moins importantes soient celles qui risquent de ne pas être développées à temps.

Cette approche, bien que basée sur des tendances approximatives, permet d'identifier très tôt les risques de défaillance et d'agir en conséquence, en conservant à l'esprit le caractère critique des fonctionnalités à développer.
Ces décisions importantes relèvent complètement du directeur de produit.

Une dernière chose importante : la fiabilité de la vélocité de l'équipe et des estimations qu'elle a faites augmente au fil des sprints : on élimine ainsi les risques majeurs au plus tôt dans le projet.

Facteur clef de succès : la qualité de l'environnement de travail

Un concept fort de Scrum est la qualité de l'environnement de travail de l'équipe.

Cela inclut :

- ⇨ Pas de changements imposés pendant un sprint
- ⇨ Toute l'équipe dans une même pièce
- ⇨ Un tableau blanc et/ou en liège
- ⇨ Un bon outil de suivi du projet
- ⇨ Prévenir des interventions extérieures (téléphone, irruption dans la pièce, etc.)
- ⇨ Tout ce qui peut rendre l'équipe plus sereine et efficace

Risques de la méthodologie

Dans Scrum, les risques, en particulier de dérives sont multiples.
Le risque majeur dans l'utilisation de Scrum est avant tout une application allant à l'encontre même des principes fondateurs de son fonctionnement.
On peut en citer deux, mais il en existe bien d'autres :

- ⇨ Le Scrum Master ne l'est plus.

L'un des plus graves risques de Scrum est la violation du principe même de son fonctionnement qui met l'équipe de développement au cœur même du processus.
Il n'est pas rare que le Scrum Master soit un chef de projet déguisé ou officiel et qu'il applique un contrôle trop strict sur les membres de son équipe.

Dans ce contexte, on demande aux développeurs de s'impliquer comme s'ils étaient responsables du projet mais en réalité ils ne le sont pas, ce qui peut apporter des frustrations côté développeur.
En outre, dans ce type de configuration, le Scrum Master, qui n'en est plus un, n'endosse plus les responsabilités du Scrum Master : être le protecteur de l'équipe, objectif, garde-fou, garant de la méthodologie appliquée, etc...
Les bénéfices de l'agilité peuvent ainsi disparaître : défiance de l'équipe vis-à-vis du Scrum Master, conflit à l'intérieur même de l'équipe, non-dit, désengagement qui engendre baisse de la qualité et absence de bonnes pratiques.

- ⇨ La communication constructive ne l'est plus.

La surexposition dans Scrum prend sa source dans les différentes réunions d'équipe telles que la quotidienne matinale, censée informer et apporter des solutions aux problèmes de la veille.
Sur certains projets, ces réunions se transforment en blâme des membres n'ayant pas réussi à mener à bien leurs tâches de la veille.
D'autres réunions quotidiennes peuvent être minées par le non-dit ou des propos vagues en ce qui concerne les problèmes rencontrés.
Dans les deux cas cela va à l'encontre même des principes de Scrum.

Documentation de projet

Scrum n'impose aucune documentation particulière pour les projets : des documents sont implicitement produits (backlogs, burndown charts), mais ils ont une vocation avant tout utilitaire.

Produire de la documentation pour de la documentation est inutile, et coût tant en production qu'en maintenance.
Aussi avant toute rédaction d'un document il faut se poser une question très simple : Est-ce que ce document va être vraiment utile, et tout de suite ?

Voici quelques exemples de documents utiles et dans quels cas :

- ⇨ diagrammes métiers (processus, objets, etc.), associé au backlog de produit : uniquement si la logique métier du client qui concerne l'application est vraiment complexe.
  Dans ce cas, l'équipe devrait produire ce document avec lui ;
- ⇨ diagramme de séquence, associé à un item du backlog du produit : uniquement si la fonctionnalité aura une utilisation complexe, tant au niveau métier qu'applicatif ;
- ⇨ diagrammes d'architecture du logiciel (classes, modules, composants, etc.), pour le projet : indispensable pour avoir toujours sous les yeux une vue de l'architecture et s'assurer ainsi qu'elle est de qualité ;
- ⇨ les manuels utilisateur, à chaque sprint : les manuels sont produits à chaque sprint et pas en fin de projet. Utiliser des vidéos de démonstrations commentées est une solution efficace ;
- ⇨ les FAQ pour le centre d'appel : des cas classiques où les utilisateurs ne vont pas comprendre un comportement métier.
  Cela permet de traiter un maximum de problèmes au niveau du centre d'appel, avant que cela n'arrive aux équipes de développement/maintenance.

Bref, un document ne doit être produit que si son utilité est réelle et immédiate.

Mise en garde

On entend de plus en plus de sociétés clamer qu'elles sont Agiles, comme argument commercial à la mode, parce qu'elles utilisent quelques pratiques des méthodes Agiles.
Mais être Agile, c'est bien plus que de mettre en pratique une méthode : l'Agilité requiert des dispositions particulières qui sont loin d'être faciles à mettre en place :

- ⇨ Une organisation adaptée :
  Il est très difficile de convaincre les organes décisionnels d'une entreprise de travailler de façon Agile : en effet, cela signifie de ne pas disposer dès le départ d'une date et d'un budget très précis, mais plutôt d'un ordre de grandeur.
- ⇨ Un état d'esprit : être Agile, c'est privilégier l'esprit d'équipe et pas seulement dans la réalisation technique.
  Le client doit comprendre que l'on attend de lui un investissement personnel bien supérieur à celui des méthodes plus classiques, en testant le logiciel souvent et en participant à toutes les réunions.
- ⇨ Un environnement favorable : éliminer les contraintes dans une entreprise n'est pas toujours évident.
  Comment limiter les appels téléphoniques trop fréquents, les irruptions dans la pièce de l'équipe, les opérations "urgentes" demandées aléatoirement par la direction, etc. ?

Bref, utiliser une méthode comme Scrum au niveau de l'équipe technique ne suffit pas.
L'Agilité est une façon de travailler différente de ce dont on a l'habitude, c'est l'attitude du changement, de la flexibilité, de l'adaptation et de l'amélioration continue.

Ce n'est pas aussi simple qu'une méthode…