Title: Méthode de Développement Logiciel
Author:
Language: FR
Pages: 106
Year published: 2010
Publisher: More: http://booksllc.net/?id=41172
Reprinted: 2010, General Books, Memphis, Tennessee, USA

Chapters: Extreme Programming, Scrum, Méthode Agile, Unified Process, Cycle de Développement, Cycle En V, Model Checking, Hermes, Manifeste Agile, Recette, Programmation Structurée, Test Driven Development, Non-Régression, Planning Poker, Structure de Kripke, Modèle D'amélioration de La Maintenance Du Logiciel, Méthode Moscow, Modèle En Spirale, Programmation Sans Ego, Two Track Unified Process, Capability Maturity Model, Test de Recette, Méthode Axial, Adaptive Software Development. Source: Wikipedia. Not illustrated.

Wikipedia

- http://fr.wikipedia.org/wiki/Categorie:Méthode_de_développement_logiciel
- http://fr.wikipedia.org/wiki/Ingénierie_dirigée_par_les_modèles
- http://fr.wikipedia.org/wiki/Adaptive_software_development
- http://fr.wikipedia.org/wiki/Capability_Maturity_Model
- http://fr.wikipedia.org/wiki/Cycle_de_développement
- http://fr.wikipedia.org/wiki/Cycle_en_V
- http://fr.wikipedia.org/wiki/Extreme_Programming
- http://fr.wikipedia.org/wiki/HERMES_(méthode)
- http://fr.wikipedia.org/wiki/Manifeste_agile
- http://fr.wikipedia.org/wiki/Méthode_agile
- http://fr.wikipedia.org/wiki/Méthode_AXIAL
- http://fr.wikipedia.org/wiki/Méthode_MoSCoW
- http://fr.wikipedia.org/wiki/Modèle_d'amélioration_de_la_maintenance_du_logiciel
- http://fr.wikipedia.org/wiki/Modèle_en_spirale
- http://fr.wikipedia.org/wiki/Model_checking
- http://fr.wikipedia.org/wiki/Non-régression
- http://fr.wikipedia.org/wiki/Planning_poker
- http://fr.wikipedia.org/wiki/Programmation_sans_ego
- http://fr.wikipedia.org/wiki/Programmation_structurée
- http://fr.wikipedia.org/wiki/Recette_(informatique)
- http://fr.wikipedia.org/wiki/Scrum
- http://fr.wikipedia.org/wiki/Scrum_(méthode)
- http://fr.wikipedia.org/wiki/Test_de_recette
- http://fr.wikipedia.org/wiki/Test_Driven_Development
- http://fr.wikipedia.org/wiki/Two_Track_Unified_Process
- http://fr.wikipedia.org/wiki/Unified_Process

Manifeste agile

http://fr.wikipedia.org/wiki/Manifeste_agile

- Le ***Manifeste Agile*** est un texte rédigé par 17 experts reconnus pour leurs apports respectifs au développement d'applications informatiques sous la forme de plusieurs méthodes dont les plus connues sont Extreme Programming et Scrum. Ces experts estimaient que le traditionnel cycle de développement en cascade ne correspondait plus aux nouveaux besoins applicatifs. Le Manifeste Agile est considéré comme l'acte généralisateur des méthodes agiles sous la dénomination initiale de *Agile Manifesto* [1]. Les valeurs et principes du Manifeste Agile sont défendus par l'Agile Alliance.

Manifeste agile

- Le Manifeste Agile débute par la déclaration suivante (traduction) :
- " *Nous avons trouvé une voie améliorant le développement logiciel en réalisant ce travail et en aidant les autres à le faire. De ce fait nous avons déduit des valeurs communes.* "
- Le Manifeste Agile est constitué de 4 valeurs et de 12 principes fondateurs.

Les 4 valeurs :

Les quatre valeurs fondamentales Agiles sont de valoriser :

- **l'interaction avec les personnes** plus que les processus et les outils.
- **un produit opérationnel** plus qu'une documentation pléthorique.
- **la collaboration avec le client** plus que la négociation de contrat.
- **la réactivité face au changement** plus que le suivi d'un plan.

Les 12 principes :

- Notre première priorité est de satisfaire le client en livrant tôt et régulièrement des logiciels utiles.
- Le changement est accepté, même tardivement dans le développement. Les processus agiles exploitent le changement comme avantage compétitif pour le client.
- Livrer fréquemment une application fonctionnelle, toutes les deux semaines à deux mois, avec une tendance pour la période la plus courte.
- Les experts métier et les développeurs doivent collaborer quotidiennement au projet.
- Bâtissez le projet autour de personnes motivées. Donnez leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail.
- La méthode la plus efficace pour transmettre l'information est une conversation en face à face.
- Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet.
- Les processus agiles promeuvent un rythme de développement durable. Commanditaires, développeurs et utilisateurs devraient pouvoir maintenir le rythme indéfiniment.
- Une attention continue à l'excellence technique et à la qualité de la conception améliore l'agilité.
- La simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle.
- Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto-organisent.
- À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens.

Résumé de la mise en pratique

http://fr.wikipedia.org/wiki/Méthode_agile

- Le développement Agile, appelé aussi développement adaptatif, se caractérise donc par un style de conduite de projet itératif incrémental, centré sur l'autonomie des ressources humaines impliquées dans la spécification, la production et la validation d'une application intégrée et testée en continu (Méthode Agile).
- C'est à partir de ces réalités pratiques, et non pas sur la base d'une théorie globale ou structurante, que l'Agilité progresse vers les sphères les plus hautes de l'organisation.

Méthode agile

- Les **méthodes Agiles** sont des groupes de pratiques pouvant s'appliquer à divers types de projets, mais se limitant plutôt actuellement aux projets de développement en informatique (conception de logiciel).
- Les méthodes Agiles se veulent plus pragmatiques que les méthodes traditionnelles. Elles impliquent au maximum le demandeur (client) et permettent une grande réactivité à ses demandes. Elles visent la satisfaction réelle du besoin du client et non les termes d'un contrat de développement.
- La notion de méthode agile a été officialisée en 2001 par un document, le Manifeste Agile (*Agile Manifesto*), signé par 17 personnalités impliquées dans l'évolution du génie logiciel, en particulier, en tant qu'auteur de leur propre méthode

Les méthodes Agiles

- Les méthodes Agiles et les pratiques qu'elles recouvrent seraient antérieures au Manifeste Agile, qui ne serait donc pas l'acte de naissance des méthodes Agiles ou du mouvement Agile, mais la formalisation consensuelle par les auteurs de ces méthodes, toutes nées dans la deuxième partie de la décennie 90, du fait qu'elles avaient des valeurs communes, une structure (cycle de développement) commune (itérative, incrémentale et adaptative) et une base de pratiques, soit communes, soit complémentaires.
- Parmi ces méthodes on trouve en premier lieu la méthode RAD (Développement rapide d'applications) de James Martin (1991), puis DSDM, la version anglaise du RAD (1995).
- Plusieurs autres méthodes comme ASD ou FDD reconnaissent leur parenté directe avec RAD (que certains de ses promoteurs présentent comme la première méthode Agile publiée).
- Les deux méthodes Agiles les plus connues en France sont : la méthode Scrum (1996) et la méthode XP, pour Extreme programming (1999).

Agile (adaptatif) = Itératif, Incrémental

- Agile (adaptatif) = Itératif, Incrémental. 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 (notion de validation permanente de l'utilisateur avec RAD et notion de conception émergente avec XP).
  - Techniquement par remaniement régulier du code déja produit (Refactoring).

Méthode Agile - incrémentale

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

Méthodes Agiles reconnues par date de publication officielle

- Rapid Application Development (RAD, 1991)
- Dynamic systems development method (DSDM, 1995, consortium anglais commercialisant le RAD)
- Scrum (1996)
- Feature Driven Development(FDD) (1999)
- Extreme programming (XP, 1999)
- Adaptive software development (ASD, 2000)
- Crystal clear (2004)

Autres Méthodes se reconnaissant de l'agilité

- MACAO
- Processus Urbanisant les Méthodes Agiles (PUMA)
- KANBAN
- Note RUP (Rational Unified Process) n'est pas une méthode Agile et, est de plus un produit propriété d'IBM. Il existe une déclinaison Agile, mais non libre de droits, de RUP sous l'acronyme de AUP (Agile Unified Process).

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 (motivation).
  - 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 raisonnables de qualité du code (métrique).
  - Architecture à base de composants.
  - Gestion des changements automatisée.

Pratiques différenciatrices des méthodes Agiles

- Seules quelques techniques complémentaires entre elles, ou mieux adaptées à des typologies et à des tailles de projets spécifiques, différencient les méthodes Agiles (y compris ASD ou Crystal Clear).
- Les pratiques différenciatrices les plus marquantes sont :
- La méthode DSDM se particularise par la spécialisation des acteurs du projet dans une **notion de « rôles »**. Ainsi, l'on trouvera dans les réunions DSDM des sponsors exécutifs, des ambassadeurs, des utilisateurs visionnaires, des utilisateurs conseillers, sans oublier l'animateur-facilitateur et des rapporteurs.
- La méthode Scrum affirme sa différence dans des pratiques de courtes réunions quotidiennes (*Stand-Up meeting*). Ces temps de travail commun ont pour objectifs d'améliorer la motivation des participants, de synchroniser les tâches, de débloquer les situations difficiles et d'accroître le partage de la connaissance.
- Pour FDD, la particularité nommée *Mission focused* réside dans une forte orientation vers un but immédiat mesurable guidé par la notion de **valeur métier**. C'est en fait l'ambition globale d'une itération qui se trouve ainsi renforcée. Cet aspect se retrouve aussi dans la méthode RAD sous la forme des objectifs de Focus ou dans Scrum dans la notion de Sprint. FDD préconise aussi le Features Driven Development. Cette technique se caractérise par des notions de Feature et de Features set (fonctionnalités et groupes de fonctionnalités). La priorité est donnée aux fonctionnalités porteuses de valeur. Le RAD propose des techniques proches : livraison en fonctionnalité réduite ou livraison par thèmes.
- XP (extreme programming) est très axé sur la partie Construction de l'application. Une de ses originalités réside dans l'approche de planification qui se matérialise sous la forme d'un jeu intitulé **Planning game** et qui implique simultanément les utilisateurs et les développeurs. On notera aussi des techniques particulières liées à la production du code comme la programmation en binôme (Pair programming), l'appropriation collective du code, la Refactorisation (*refactoring*) et l' Intégration continue. La méthode RAD préconise dans ce sens des revues de code personnelles, puis collectives et l'intégration avant chaque Focus (ou Show). Par contre, le RAD limite la programmation en binôme aux parties les plus stratégiques.

Pratiques d'autres méthodes proches ou ayant un rapport avec les méthodes agiles

- 2TUP (2 track unified process, prononcez "toutiyoupi") préconise un cycle de vie en Y qui dissocie la résolution des questions fonctionnelles et techniques. Le cycle de vie de 2TUP s'apparente à un cycle de développement en cascade mais introduit une forme itérative interne à certaines tâches. Il n'est pas certain que ce cycle s'apparente réellement à une approche Agile. Par contre, 2TUP préconise des formes de recherche de qualité et de performance intéressantes telles que les services réutilisables et la conception générique (Framework et Patron de conception *Design pattern*) proches des mécanismes architecturaux de RUP.
- RUP se caractérise par une approche globale nommée « **Vue 4+1** ». Les 5 composants de cette vue sont : la vue des Cas d'utilisation, la vue Logique, la vue d'Implémentation, la vue du Processus, la vue du Déploiement. RUP offre aussi, à l'identique du RAD 2, un Processus guide formel et adaptable comme guide d'activité. Dans le cas de RUP, il est propriétaire et orienté outil.
- Le plus sérieux apport du RAD à la communication de projet et à la formalisation des exigences applicatives : le **Groupe d'Animation et de Rapport** (GAR).
- Le RAD dans sa version 2 recommande la **variabilité** de la taille et de la maturité des groupes de travail en fonction des phases du projet afin d'optimiser l'engagement des ressources et de préserver leur intérêt par un travail adapté à leurs préoccupations.
- Avec RAD 2, l'organisation performante des réunions est basée sur un **mode opératoire des entretiens** et sur des **techniques de validation permanente**.
- Toute méthode de conduite de projet devrait inclure un mode opératoire pour les arrêts d'urgence (**Go/NoGo**). Sur ce point la force du RAD se situe dans la présence d'un animateur-facilitateur. Cette ressource, de préférence externe, doit être neutre en regard des autres intervenants. Elle répond à une autorité supérieure à tous les participants du projet. Ainsi, lorsque le contexte stratégique, économique ou technique d'un projet évolue, ou si les conditions de réalisation se dégradent, l'animateur reporte le problème au dirigeant qui l'a mandaté. Ce dernier peut alors prendre, dans les meilleurs délais, et avec des informations objectives les décisions qui s'imposent.
- Le RAD comme les autres méthodes Agiles préconise la formation d'une équipe de développement particulière, le SWAT (*Skill With Advanced Tools* dérivé de Special Weapons And Tactics). Cette équipe est autonome, spécialement formée, concrètement motivée et outillée. Elle se compose essentiellement d'un profil unique de concepteurs-développeurs formés à des spécialités techniques complémentaires. Le SWAT travaille avec les utilisateurs dans une salle permanente spécialement équipée style **War Room** qui transforme les murs en Information Radiator (un équivalent de cokpit de management de projet).

Caractéristiques

- Le terme *Scrum* est emprunté au rugby à XV et 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. Ces incréments 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 base de l'Extrême Programming). En résumé, la notion d'incrément est une valeur facilitant le contrôle de pilotage du projet et la notion d'itération (Méthode itérative) est une valeur conduisant à l'adaptabilité ainsi qu'à la qualité fonctionnelle ou technique. Ces points représentent les principes fondamentaux d'une Méthode agile appliquée à la complexité de l'Ingénierie du logiciel

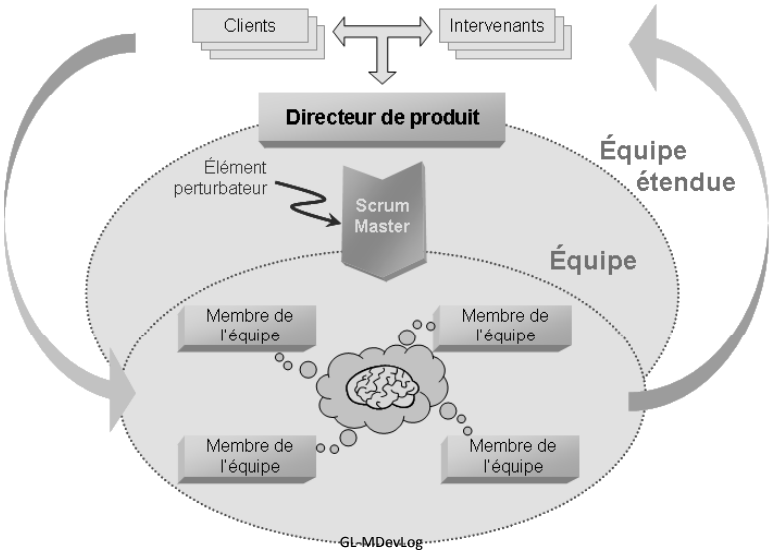
Quelques rappels sur les méthodes agiles

- 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. Les processus qui définissent ce que doit faire chaque personne brident 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, etc. Ces documents ne sont qu'une illusion d'avancement du projet. 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 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. 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.

Idées clé

- Le client au cœur du projet
- Esprit d'équipe
- La communication est la clé
- Simplicité, efficacité et qualité
- Flexibilité aux changements
- Avancement basé sur le concret

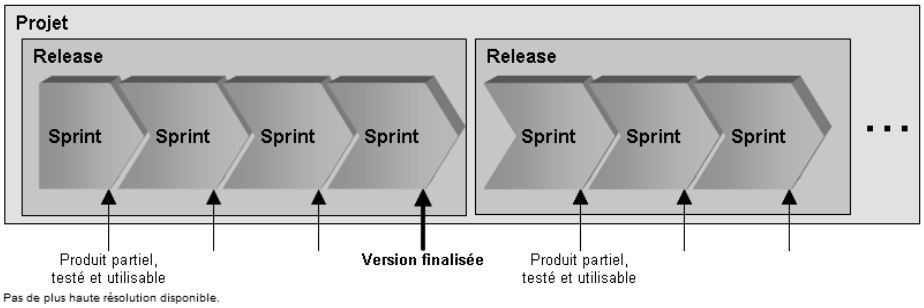
Rôles



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).
- **Équipe**
- L'**équipe** ne comporte pas de rôles prédéfinis, elle est **auto-gérée**. 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. Contrairement à ce que l'on pourrait croire, les équipes auto-gérées sont celles qui sont les plus efficaces et qui produisent le meilleur niveau de qualité de façon spontanée.
- 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.
- **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.
- **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



Planification

- Scrum propose une planification opérationnelle à trois niveaux : release/projet, sprint et quotidien.
- **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.
- **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.
- **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** (NDT : Le terme *backlog* peut être traduit par *cahier*, *liste* ou *carnet de commandes*, qui ne collent pas bien avec l'esprit du terme anglais qui 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** (voir Estimation) 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).
- **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 User 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, auquel cas 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 (1/2)

- **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'auto-gère 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.

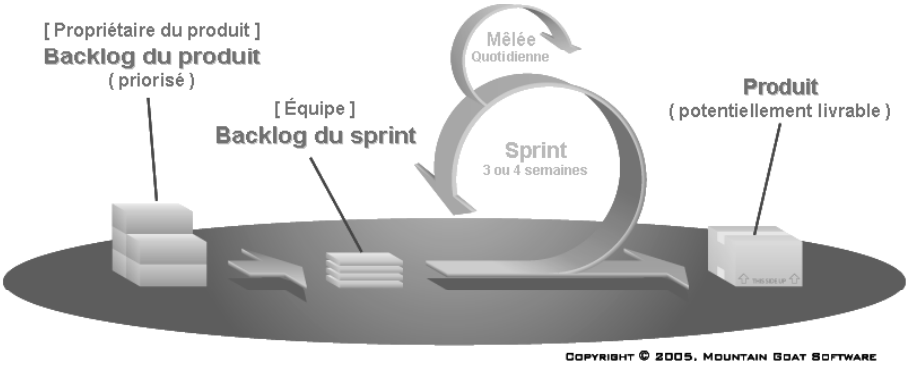
Déroulement d'un sprint (2/2)

- **Revue de 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.

Compléments

- **Lancement du projet**
- 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 d'opportunité et l'expression initiale des besoins.
- L'étude d'opportunité 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.

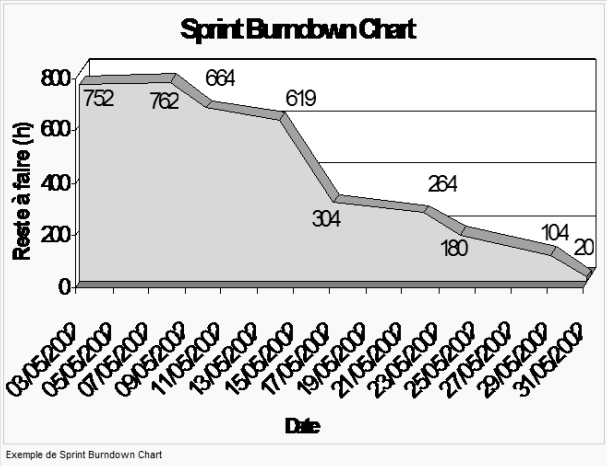
Vue globale



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.

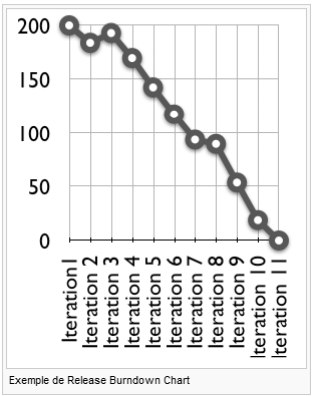
Sprint Burndown Chart



Exemple de 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



Exemple de 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

- Ces graphiques sont très intéressants à analyser et interpréter. Outre le fait de montrer l'avancement concret du travail, ils 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.
- On peut utiliser en cours du sprint la tendance de la courbe pour avoir une idée approximative de la fin du sprint. Cela consiste à prendre un segment de droite dont la pente est représentative des valeurs déjà recensées et de le prolonger jusqu'à son point d'intersection avec l'axe des abscisses. Cela nous donne alors la date *a priori* de la fin du sprint et nous 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 ce qui s'est probablement passé le 15 mai 2002 sur le graphique de sprint ci-dessus : le reste à faire diminue dans ce cas assez brutalement.
- 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 de cette façon les risques majeurs au plus tôt dans le projet.

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

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, c'est souvent utile, mais c'est aussi souvent inutile. En plus, il faut la maintenir à jour et c'est quelque chose qui est rarement fait sur le terrain. Pour savoir s'il faut rédiger un document, on peut se poser une question très simple : **Est-ce que ce document va m'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 la hotline : des cas classiques où les utilisateurs ne vont pas comprendre un comportement métier. Cela permet de traiter un maximum de problèmes au niveau de la hotline, 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**.

Outils pour Scrum

Un bon artisan n'est rien sans un bon outil. Il n'y a pas aujourd'hui d'outil *ultime* pour Scrum. Voici un petit panel des possibilités.

Papier et crayon / tableur[modifier]

Scrum peut être mis en pratique avec trois fois rien : deux listes suffisent. La liste des items du backlog de produit et la liste des items du backlog de sprint. La saisie et la mise à jour des données est simplement un peu moins agréable.

Outils libres[modifier]

Aldon Agile Manager Aldon Agile Manager est gratuit
IceScrum (open-source et démo en ligne)
theSCRUM open-source et démo en ligne, implémente un tableau blanc virtuel
EPF Eclipse Process Framework intègre un plug-in Scrum (open-source)
OpenERP OpenERP intègre un module Scrum (open-source)

Outils propriétaires[modifier]

Youkan (Gratuit pour l'utilisation en ligne)
AccuRev (Solutions de gestion de configuration logicielle intégrant les méthodes Agile et Scrum)
ScrumDesk (gratuit pour 5 utilisateurs maximum)
Intra'Know PROJET (cette solution collaborative intégre les grands principes de la méthode AGILE)
ScrumWorks (gratuit sous conditions)
VersionOne (gratuit sous conditions)
RallyDev
GreenHopper
Microsoft eScrum Version 1.0 pour Visual Studio Team Foundation Server
Scrumy (Démonstration en ligne gratuite)
ScrumEdge (Démonstration en ligne gratuite)
Serena Agile On Demand Entièrement en mode SAAS. Aucune installation nécessaire. Essai gratuit 60j.
Urban Turtle Plugin for TFS web Access (Essai gratuit 30 Jours).
Virtual SCRUM Board (Essais gratuit de 60 Jours)

Conclusion – Scrum

- Scrum est un processus de développement de logiciels qui s'intéresse plutôt à l'organisation du projet qu'aux aspects techniques. Son cadre de travail est sa force, si bien que cette méthode pourrait être appliquée à d'autres domaines. Son approche incrémentale et basée sur les besoins priorisés du client lui confèrent une flexibilité extrême. Elle incarne bien par là l'état d'esprit de la mêlée de rugby : avancer tous ensemble vers un but commun, la réussite du projet.
- Les pratiques de Scrum sont essentiellement orientées sur la maîtrise d'une livraison d'incréments (sprint) mais réfutent la possibilité de modifier les fonctionnalités en cours de réalisation (dans le backlog de sprint). Cette limite interdit la mise en œuvre d'une conception émergente comme celle d'XP ainsi que celle d' **itération**, base de l'adaptabilité (possibilité d'affinement par modification permanente). Scrum, ne disposant pas de métrique de gestion du changement à ce niveau, nécessite donc une importante spécification préalable à la mise en production (backlog produit) et, du fait de cette prédictibilité imposée, ne peut pas être considéré comme réellement itératif, donc adaptatif.
- Scrum est un processus intéressant comme premier pas vers les méthodes Agiles : il est facile à comprendre et à pratiquer. La seule difficulté relève plutôt de l'environnement organisationnel.

Conclusion - 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 faire admettre aux organes décideurs 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…

Conclusion – Glossaire

- Directeur de produit (*Product Owner*) personne responsable de produire et maintenir à jour le backlog de produit. C'est lui qui en détermine les priorités et qui prend les décisions concernant l'orientation du projet.
- ScrumMaster membre de l'équipe dont l'objectif principal est de la protéger des perturbations extérieures. Il est complètement transparent pour la communication entre l'équipe et les clients et n'a aucun pouvoir hiérarchique sur l'équipe. C'est en revanche un facilitateur pour les problèmes non techniques de l'équipe.
- Backlog de produit (*Product Backlog*) liste des fonctionnalités qui devront être réalisées par le logiciel.
- Backlog de sprint (*Sprint Backlog*) liste des tâches à accomplir pendant un sprint. Elles correspondent à la réalisation des items de backlog du produit affectés au sprint.
- Mêlée quotidienne (*Daily Scrum*) réunion quotidienne de 15 minutes qui a pour but de faire le point sur ce qui a été fait depuis la dernière mêlée, ce qui est prévu de faire jusqu'à la prochaine et quelles sont les embûches rencontrées durant le travail.
- Sprint nom d'une itération dans Scrum. Cette itération dure 30 jours calendaires en théorie, mais en pratique on utilise plutôt entre 2 et 4 semaines. Pendant une itération, l'équipe doit développer une liste d'items du backlog de produit qui a été définie au début de ce sprint
- Graphique d'avancement (*Burndown Chart*) graphique qui montre la tendance du reste à faire total de jour en jour (pour les sprints) ou de sprint en sprint (pour les releases).

Planning poker (1/2)

http://fr.wikipedia.org/wiki/Planning_poker

- Le **planning poker** est une façon ludique et efficace de produire des estimations sur la complexité des fonctionnalités à développer. Cette pratique est surtout utilisée en Scrum et dans les méthodes agiles en général pour évaluer les scénarios utilisateurs (*user stories*) du carnet de produit (*product backlog*). La méthode a été décrite pour la première fois par James Grenning en 2002 et popularisée par Mike Cohn dans le livre *Agile Estimating and Planning*.

- **Utilisation**
- L'avantage principal du planning poker est de permettre à tous de s'exprimer librement. L'estimation sera meilleure parce que plusieurs personnes l'auront validée : des participants avec des niveaux d'expérience et d'expertise différents. De plus, cette technique favorise les échanges entre le responsable de produits et l'équipe de développement.
- L'estimation se fait en points, ce qui permet d'obtenir une véritable mesure de complexité relative : les scénarios sont comparés entre eux. L'équivalent en jours-hommes est propre à chacun, selon ses compétences, son expérience et sa connaissance du domaine. L'avantage d'utiliser des points réside surtout dans le fait que l'échelle utilisée restera stable tout au long du projet. Peu importe la vitesse (*vélocité*) à laquelle l'équipe de développement accomplira ces tâches, nul besoin de reviser les estimations : c'est le rapport entre le temps réel et les points qui évoluera.

Planning poker (2/2)

- La suite de Fibonacci est utilisée pour les évaluations. Comme nous cherchons un ordre de complexité, le message est clair : plus le scénario est gros, moins l'évaluation est précise. 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.

- **Déroulement**
- Les participants s'installent autour d'une table, placés de façon à ce que tout le monde puisse se voir.
- Le responsable de produit explique à l'équipe un scénario utilisateur (*user story*).
- Les participants posent des questions au responsable de produit, discutent du périmètre du scénario, évoquent les conditions de satisfaction qui permettront de la considérer comme "terminée".
- 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.
- Au signal du facilitateur, les cartes sont retournées en même temps.
- S'il n'y a pas unanimité, la discussion reprend.
- On répète le processus d'estimation jusqu'à l'obtention de l'unanimité.