Méthodes agiles

Michel Plasse (mplasse@free.fr)

Objectif

- Comprendre l'intérêt et l'esprit des méthodes agiles
- En maîtriser les principales composantes

Bibliographie

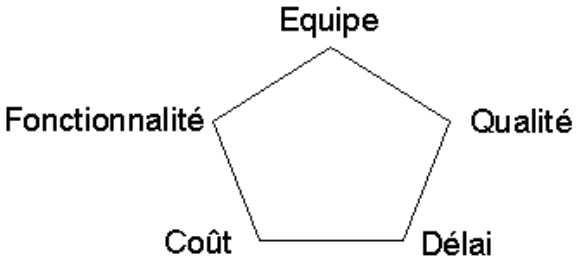
- www.scrumtrainingseries.com (en anglais, sous forme de bande dessinée)
- *Agile! The Good, the Hype and the Ugly* (190 pages), par Bertrand Meyer
- http://se.ethz.ch/~meyer/publications/methodology/agile_software.pdf, un court article de Bertrand Meyer sur les aspects bénéfiques et aussi sur les néfastes des méthodes agiles
- https://www.agileconnection.com/article/role-agile-coach sur le rôle du « coach agile », en particulier la nuance par rapport au « scrum master »
- http://www.anyideas.net/2014/09/12-images-comprendre-chef-de-projet/ sur la conduite de projet, visuel et excellent

1 Le besoin de méthodes

- Tout projet informatique
  - Vise un **objectif**. Dans le cas d'une application, l'objectif se décline en **fonctionnalités**.
  - Doit être terminé dans un **délai** donné, à un **coût** donné.
  - Le résultat doit avoir une certaine **qualité**, notamment :
    - *Ergonomie* : l'application est pratique, facile à utiliser.
    - *Evolutivité* : il est relativement facile de la faire évoluer.
- => Question :

**« Comment réussir un projet informatique,
avec une équipe donnée,
en respectant les coûts, les délais et la qualité ? »**

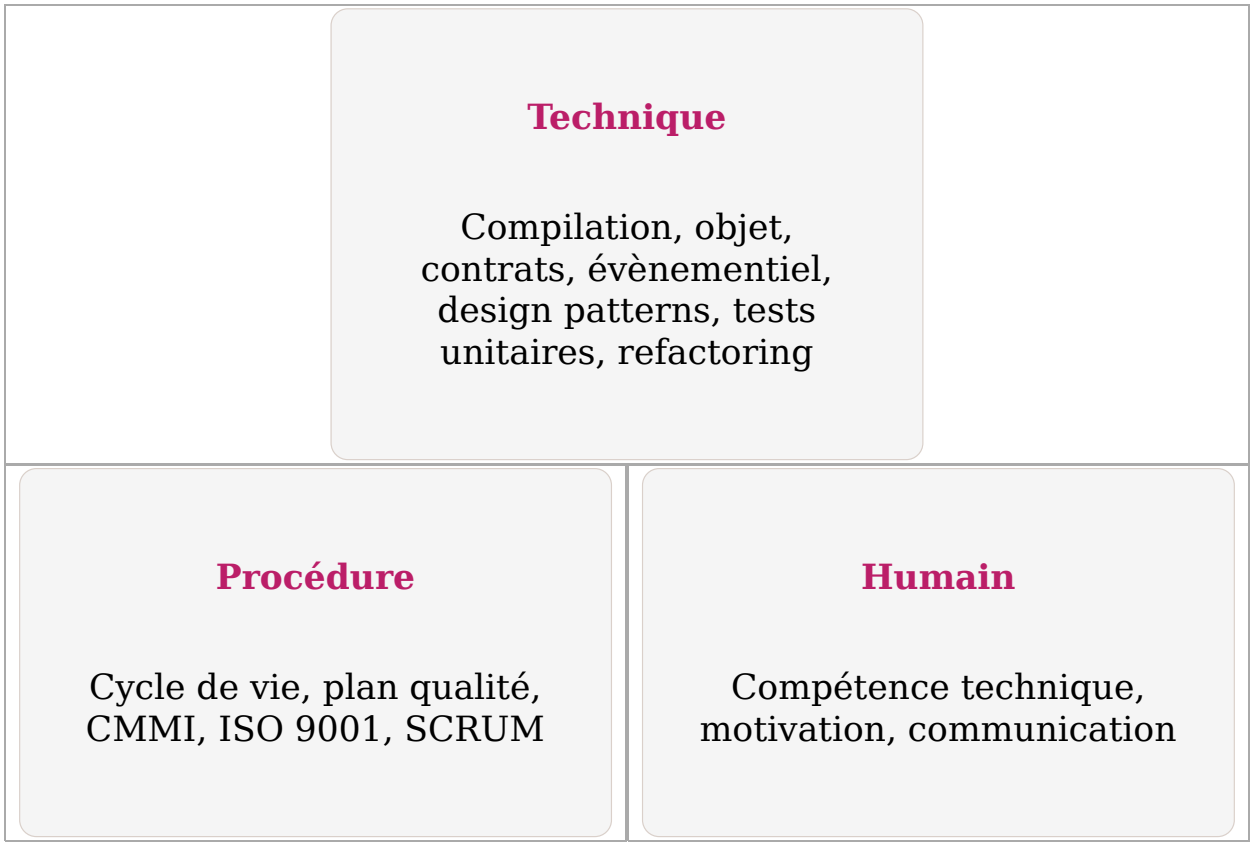
1.1 Des paramètres interdépendants



- Cette figure est souvent réduite à un triangle *coût-délai-fonctionnalité*.
- Pourtant, dans le cas des applications, la **qualité** est essentielle au succès :
  - Sans *ergonomie*, l'application sera boudée des utilisateurs (voire clients). A moins qu'ils ne soient captifs, l'échec est assuré.
  - Sans *évolutivité*, il sera difficile de l'adapter aux nouveaux besoins.
- L'**équipe** est également cruciale : sa *compétence*, sa cohésion et sa *motivation* font toute la différence.

1.2 Réponses à la question

Elles sont sur trois grands axes :



1.2.1 Les méthodes antérieures aux méthodes agiles

- Visent la *qualité du processus* (qui contribue à la qualité du résultat) :
  - **Organisation** efficace des acteurs : MOA (maitrise d'ouvrage), MOE (maitrise d'œuvre), chefs de projet, comité de pilotage, plan qualité.
  - **Cycle de vie** du projet bien maîtrisé. Le plus connu est le cycle en V.
- Influence de l'*industrie*, où la qualité des processus est le facteur le plus important.
- + Influence du *bâtiment*, avec MOA et MOE.
- + Constats de bon sens : il faut
  - comprendre le besoin *avant* d'y répondre,
  - savoir ce qu'on doit faire et imaginer comment on va le faire *avant* de le faire,
  - s'assurer que c'est correct dès que c'est possible,
  - planifier les tâches et répartir le travail.

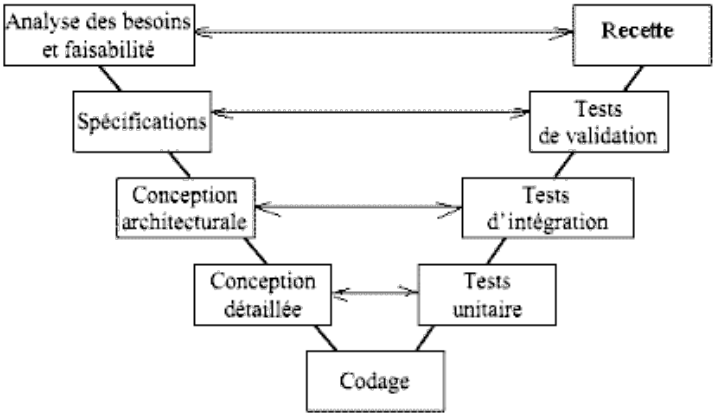
1.3 Organisation : les acteurs

Client ou MOA
Exprime un besoin
Valide que le produit fourni satisfait ce besoin

Prestataire	**Utilisateur**
Conçoit et réalise le produit	Utilise le produit livré

- En général, une personne n'appartient qu'à un groupe
  (sauf pour les projets d'outils logiciels où les développeurs sont leurs propres clients et utilisateurs).
- Le cahier des charges et le dossier de recette ont valeur de contrat.

1.4 Le cycle en V



- *Le cycle de vie le plus employé en entreprise*.
- Chaque étape produit un résultat (**livrable**), souvent sous forme de document.
- Intérêt : fait correspondre chaque phase avant la programmation à une phase de vérification.

1.4.1 Mauvaise nouvelle : des effets contreproductifs

- Processus *incompatible avec le changement* : le besoin est figé au début.
  Or le besoin évolue de plus en plus :
  - Concurrence. Ex : un concurrent met en ligne tel de ses services => il faut 'aligner dessus.
  - Evolution technologique : Web, applications mobiles, cloud computing.
  - Evolution métier : contraintes européennes, normes internationales, certifications, changements de lois et règlements.
- Un processus respectant le cycle ne mène pas forcément à une bonne application dans le temps imparti :
  - *Dérive bureaucratique* : produire beaucoup de documents peu utiles au résultat (l'application), et désynchronisés du code.
  - Spécialisation : codeur, testeur => *démotivation et compétences pas cultivées*
- Les *échecs* sont toujours aussi nombreux (chiffres édifiants du Gartner Group).

1.4.2 Bonne nouvelle : de nouveaux outils qui rendent agiles

- Certains outils rendent le développement beaucoup plus *productif* et *adaptatif* :

    - **Tests** de régression automatisés (JUnit et ses équivalents dans d'autres langages).
    - **Intégration** continue et gestion de version (subversion, git).
    - IDE avec **refactoring** (renommage de variables, de routines, de classes, ajouts de paramètres, déplacement de membres vers les classes ancêtres, etc.), compilation à la volée, suggestion de saisie, comparaison de versions etc.
    - Langages objet, grâce auxquels on peut réutiliser du code (surtout pour les couches technique).
- => Beaucoup plus de facilité à changer qu'avant = début d'**agilité**.

2 Les méthodes agiles

2.1 Histoire

- Années 1990 : des méthodes nouvelles réagissent contre les effets contreproductifs et tirent parti des outils.
  Leurs auteurs sont des pionniers du RAD (Rapid Application Development) et des concepteurs d'outils logiciels (JUnit, Wiki).
- 2001 : ils publient en commun le **manifeste agile**, qui met en avant 4 valeurs et 12 pratiques.
- Objectif de ces méthodes :

> Produire par *incréments*
> une application
> aux *besoins en évolution* ,
> qui *satisfait* et *implique*
> le client,
> en équipe *auto-organisée*
> et *pluridisciplinaire*

- Aujourd'hui, deux méthodes complémentaires dominent, et sont fréquemment utilisées conjointement :
    - SCRUM (mêlée en anglais) : centrée sur la gestion de projet
    - XP (eXtreme Programming) : centrée sur les pratiques de développement

2.1.1 Une métaphore (source : Inigo Soto, scrum master chez Cap Gemini)

Pour fabriquer un couteau suisse, on procèderait par étapes :

1. Fabriquer d'abord le manche avec une lame
2. Ajouter un tire-bouchon et une autre lame.
3. En fait, la concurrence investit le marché des bricoleurs, où le tire-bouchon est inutile => Le remplacer par un tournevis.
4. Toujours pour s'adapter au marché, ajouter un tournevis cruciforme.

2.2 Quatre valeurs

Personnes et interactions (*équipe*) plutôt que processus et outils	***Application*** **qui fonctionne** plutôt que documentation exhaustive
Collaboration **du client**	***Adaptation*** **au**

plutôt que négociation de contrat	**changement** plutôt que suivi d'un plan préétabli

1. L'auto-organisation des équipes et leur motivation sont importantes.
   Les développeurs (au sens américain du terme) ont plusieurs compétences et des idées.
2. Montrer au client une application qui fonctionne est plus utile et satisfaisant que lui présenter des documents.
3. Les besoins ne peuvent pas être connus entièrement au tout début. Il est important d'impliquer le client tout le long du projet pour les affiner et les faire évoluer.
4. Répondre rapidement aux demandes de changement, et développer de façon continue (incrémentale).
5. **Attention** : Les processus, outils, documentation, contrats et planification ne sont pas rejetés, ils sont simplement en arrière-plan.

2.3 Douze principes

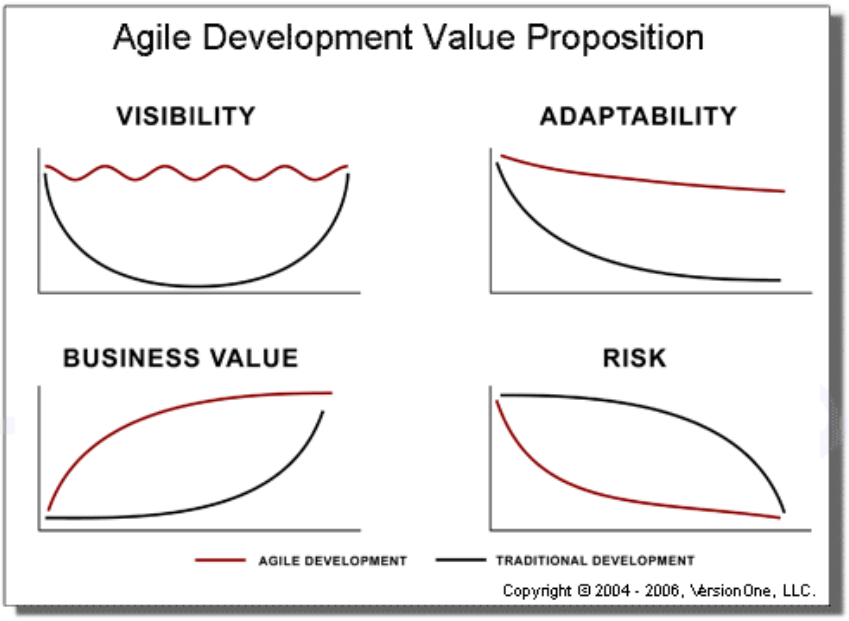
(source : http://www.agilemanifesto.org/principles.html)

- L'application est l'essentiel, et elle évolue continûment :
  1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  2. Welcome changing requirements, even late in development.
     Agile processes harness change for the customer's competitive advantage.
  3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
  4. Working software is the primary measure of progress.
- Des conditions sont importantes pour que l'équipe soit motivée et productive :
  1. Business people and developers must work together daily throughout the project.
  2. Build projects around motivated individuals.
     Give them the environment and support they need, and trust them to get the job done.
  3. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
  4. Agile processes promote sustainable development.
     The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Veiller en permanence à la qualité du résultat et du travail :
  1. Continuous attention to technical excellence and good design enhances agility.
  2. Simplicity – the art of maximizing the amount of work not done – is essential.
  3. The best architectures, requirements, and designs emerge from self-organizing teams.
  4. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Le dernier point concerne l'**amélioration continue** (cf la roue de Deming en qualité des processus).

2.4 Quatre bénéfices des méthodes agiles

Selon VersionOne (www.versionone.com/), les méthodes agiles tiennent 4 promesses par rapport aux méthodes précédentes (cycle en V notamment) :



- La **valeur** produite (les fonctionnalités selon leur priorité) croit régulièrement en agile, avec la forme d'un log, tandis qu'en V c'est plutôt une courbe plate avec une brusque asymptote à la fin.
  A mi-chemin du projet (sa demi-vie), on atteint 80% du besoin en agile, et 25% en V.
- La **visibilité**, qui oscille autour d'une valeur constante en agile, à haut niveau, alors que le V chute très vite près de 0 pour revenir au maximum à la fin du projet (on appelle cela l'effet tunnel).
- Le **risque** (« un problème en puissance ») diminue dans les deux cas, car quand les problèmes en puissance surviennent, ils ne sont plus « en puissance », mais réalisés. Seulement, en agile ils diminuent très vite (toujours une courbe de forme 1/x), car les principaux sont traités dès le début, au contraire du V.
- La **facilité à changer** le produit développé, qui de grande au début, diminue un peu et continûment en agile, alors qu'elle chute beaucoup plus en V;

2.5 SCRUM

- « Mêlée » de rugby en anglais : il s'agit que l'équipe fasse avancer la balle (l'application). Insiste sur l'aspect « équipe » qui est très fort.
- Auteurs : Ken Schwaber et Jeff Sutherland.
- Rôles :

Commanditaire (product owner)
= MOA
Donne les fonctionnalités et les priorités

Equipe de développement
Principalement auto-organisée
Analyse, conçoit, code et teste l'application

Facilitateur (scrum master)
Veille à la qualité de l'environnement, évite les perturbations
N'est pas un chef de projet (plutôt un coach)

2.5.1 Itérations - incréments

- Appelés « **sprint** » en SCRUM.
- Durée : 2 à 4 semaines, fixée au début du projet.
  - La durée ne varie pas au cours du projet, les dates de livraison sont ainsi connues à

l'avance.
    - C'est la fonctionnalité à délivrer qui varie, et se négocie au début de chaque sprint.
- Le commanditaire (product owner) fournit (en dialoguant avec l'équipe)
    - Une liste de besoins pour le produit = carnet de produit (**product backlog**).
    - Un sous-ensemble à réaliser dans le sprint = carnet d'itération (**sprint backlog**)
- Pour chaque élément du carnet :
    - La priorité et la valeur pour le produit sont définis par le directeur de produit.
    - Le temps pour sa réalisation est estimé par l'équipe.
    - Il s'exprime simplement, en général sur un A5.

2.5.2 Organisation

- Le travail est découpé en tâches n'excédant pas deux jours, et estimées en heures.
- Un tableau (virtuel dans le cas d'une équipe éclatée sur plusieurs sites) dispose les tâches sur 3 colonnes :
    - A faire
    - En cours (prise en charge par untel)
    - Fait

  Un tableau blanc permet à chacun de voir où on en est, avec le temps mis sur chaque tâche.

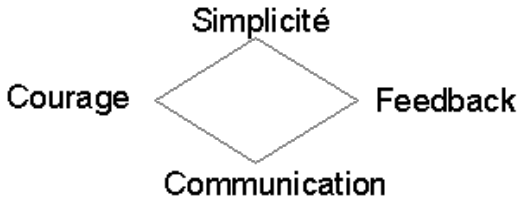
- En permanence, on voit ce qu'il reste à faire (en fonctions et en temps).
- **Vélocité** = nombre de fonctions réalisées par l'équipe dans un sprint. On la mesure et l'estime à chaque sprint.

2.5.3 Réunions

- Réunion de planification. 4h maximum. Tous sont présents.
- Réunion quotidienne de l'équipe + scrum master :
    - Ai fait
    - A faire aujourd'hui
    - Difficultés rencontrées
- Revue du sprint. 4h Maximum. Tous présents.
    - Rétrospective du sprint : feedback, voir comment améliorer le fonctionnement.

2.6 eXtreme Programming

- Pousse à l'extrême les pratiques de développement qui ont prouvé leur valeur. D'où son nom.
- Quatre valeurs fondamentales :



    - Simplicité : la conception et le code doivent être toujours le plus simple possible. Pour cela, on fait grandement appel aux outils de remaniement (refactoring) des IDE modernes. De même, si l'ajout d'une nouvelle fonctionnalité complique le code existant, on simplifie l'ensemble en revoyant la conception (ex : introduction d'une classe qui généralise une classe existante et une nouvelle ; nouveau découpage du travail entre les classes).
    - Le courage intervient lorsqu'il faut annoncer au client que l'on rencontre telle difficulté, et que donc il va falloir revoir le planning ou les priorités. Le courage consiste aussi à garantir aux développeurs un rythme de travail soutenable : un développeur fatigué travaille mal (ou voudra partir). Les heures supplémentaires ne deviennent donc pas une habitude.
    - La communication s'entend entre les développeurs, et aussi avec le client. Il est même

préférable que celui-ci soit intégré à l'équipe. Ceci est d'ailleurs un excellent facteur pour une meilleure ergonomie.
    - Le feedback permet de savoir très vite si un besoin change, ou si un problème survient. Les tests unitaires automatiques (JUnit ou NUnit) permettent cette réactivité sur les tests de non-régression.

2.6.1 Rôles

- Client = Directeur de produit de SCRUM.
- Développeur = équipe de SCRUM.
- Rôles annexes :
    - Manager
    - Coach
    - Tracker : s'occupe du planning, surveille la vélocité.
    - Testeur : gère infrastructure de test et d'intégration.

2.6.2 Pratiques

XP est surtout connue pour ses pratiques :

- Pratiques de management :
    1. Le client fait partie de l'équipe.
    2. Respect des rôles : le client donne les fonctionnalités et priorités, l'équipe estime et prend les décisions techniques.
    3. Faire des livraisons régulières.
    4. Rythme de travail régulier, soutenu et soutenable.
- Pratiques de programmation :
    1. La conception et le code doivent être simples, et le rester.
    2. Pour cela, ne pas hésiter à refondre (refactor en anglais) en permanence.
    3. Utiliser les standards (de nommage, de classes etc.).
    4. Utiliser un vocabulaire commun.
- Pratiques de développement :
    1. Ecrire les tests avant le code.
    2. Le code appartient à chacun =>
        - chacun a une vision de la totalité de l'application,
        - chacun revoit le code et peut le simplifier
        - le travail est plus diversifié
    3. Intégration permanente
    4. Travail en binôme (ex : A écrit le test et B le code testé).

2.7 Récits utilisateurs (user story)

- Exprime le besoin.
- Résume en quelques phrases ce que l'utilisateur fait dans son travail.
- S'exprime dans le langage de tous les jours ou du métier, sans jargon informatique.
- Quelques grilles pour écrire de bons récits ont été proposées.

2.7.1 Rôle - fonctionnalité - bénéfice (Rachel Davies et Tim McKinnon, 2001)

- Ex :
    - **En tant que** client, (= **qui**)
    - **je peux faire un virement** sur un compte d'un autre client depuis le site Web de la banque, (= **quoi**)
    - **afin de** pouvoir en faire où je veux et quand je veux, plutôt que me déplacer en agence aux heures d'ouverture. (= **pour quoi**)
- Le qui et le quoi sont présents dans les cas d'utilisation d'UML.
- Le « pour quoi » indique quel bénéfice (valeur) apporte la fonctionnalité.
    - Le bénéfice peut être pour l'utilisateur comme pour le commanditaire.

2.7.2 Grille INVEST (Bill Wake, 2003)

Un bon récit utilisateur est :

Lettre	Signification	Description
I	Indépendant	Ne doit pas dépendre d'autres récits. A noter : « modifier son profil » ne dépend pas vraiment de « se connecter », on peut programmer et tester cette fonctionnalité avec un utilisateur en dur.
N	Negociable	Doit capturer l'essentiel. Les détails s'ajoutent au fil du temps par une co-création client/équipe. Tant que le récit n'est pas inclus dans l'itération, il peut être changé, réécrit, et même supprimé.
V	ayant de la Valeur	Doit apporter un bénéfice au client (souvent via un bénéfice pour l'utilisateur final).
E	Estimable	Le temps de réalisation doit pouvoir être estimé.
S	petit (Small)	De l'ordre de quelques jours-hommes au plus.
T	Testable	Par des tests unitaires, voire un test fonctionnel.

Source : http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Explique en outre comment l'auteur a inventé l'acronyme : instructif sur la créativité.

2.7.3 Grille Given-When-Then

Décrit en fait ce que doit contenir un test unitaire, ou une spécification de style conception par contrat : prérequis, action, garantie.

Etant donné un solde positif de mon compte, et aucun retrait cette semaine,
Lorsque je retire un montant inférieur à la limite de retrait,
Alors mon retrait doit se dérouler sans erreur ou avertissement.

Source : http://referentiel.institut-agile.fr/gwt.html.

2.7.4 Critères (= tests) d'acceptation (acceptance tests)

- Comment valider qu'un récit utilisateur est bien implémenté ? Par des tests d'acceptation (tests fonctionnels).
- Exemple :
  1. Le site liste mes comptes pouvant être débités, avec leur solde.
  2. Je peux saisir le montant et le compte à créditer.
  3. L'application vérifie l'existence du compte à créditer en indiquant l'identité de son possesseur.
  4. Il vérifie que le virement ne mettra pas mon compte débité à découvert.
  5. L'application demande une confirmation du virement.
  6. Toute l'opération se fait de façon sécurisée (via https).
  7. Le virement apparait ensuite dans la liste de mes virements.
- Ces critères s'expriment eux aussi en langage de tous les jours ou métier.

2.7.5 Récit vs critère

- Récits et tests s'expriment en *langage de tous les jours* ou dans le *langage métier du client* : ils sont donc compréhensibles par lui.
- Les récits sont *succincts*, si bien que leur liste se parcourt assez vite : utile pour avoir une idée globale du produit.
- Les tests d'acceptation sont beaucoup plus *détaillés* : ils listent les points clé à vérifier.
- Récits comme critères ne rentrent pas dans les détails techniques (ex : position d'un bouton dans une IHM).
  Ces détails se règlent au jour le jour : le client/MOA est facilement accessible en méthode agile.

2.7.6 Valider une itération

- A la fin d'une itération, une fonctionnalité prévue est considéré comme valide quand :
  - Tous ses **tests réussissent** :
    - Tests fonctionnels, validés par le client/MOA.
    - Tests unitaires.
  - Le **code est propre, simple et nettoyé de ce qui ne sert pas** (ce point demande une grande vigilance).
  - La **documentation** (notamment JavaDoc ou équivalent) est produite.

2.8 Estimation : poker planning

- C'est l'équipe qui estime la charge de travail pour développer chaque scénario utilisateur.
- Chaque membre inscrit sur une carte son estimation de la charge.
- La charge est estimée dans l'unité qui convient à l'équipe. Ex :
  - En points (1 point = temps pour réaliser complètement la plus petite fonctionnalité possible).
  - Avec 3 valeurs : petit, moyen, grand.
- Pour chaque scénario, les membres découvrent en même temps leur carte :
  - Si tous ont la même valeur, elle est adoptée.
  - Sinon, discussion sur les raisons, et rebelotte jusqu'à consensus.