SUPPORT DE COURS

Méthodes de conception

Date	Révision
26/05/2018	Publication initiale BTS SIO
03/04/2019	Ajout image projet et corrections mineures

TABLE DES MATIÈRES

1 GÉNÉRALITÉS

1.1 INTRODUCTION

L'informatique regorge de méthodes pour la conception de programmes. Le terme génie logiciel est même parfois utilisé pour décrire l'ensemble des méthodes à appliquer pour qu'un projet passe d'un besoin à une application opérationnelle.

En effet, les systèmes informatiques sont constitués à 20 % de matériel et 80 % de logiciel : la plupart des problèmes rencontrés ont une origine logicielle. Pour améliorer la qualité, il faut donc utiliser des méthodes, comme celles présentes dans le génie civil ou d'autres domaines.

Toutefois, les années passant, l'expérience a montré que les méthodes "empruntées" à d'autres domaines ne sont pas les meilleures et de nouvelles méthodes ont vu le jour.

Ce cours aborde donc les méthodes initiales et leurs défauts pour ensuite proposer les méthodes actuellement utilisées en entreprises.

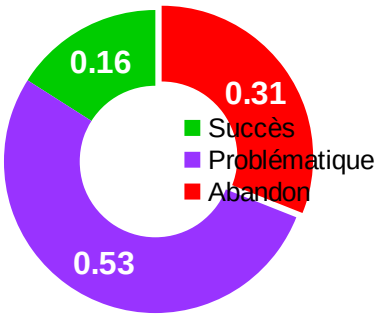
1.2 OBJECTIFS

À la fin de ce cours, vous serez capable :

- de comprendre les avantages et inconvénients des différentes méthodes
- d'expliquer le fonctionnement global des méthodes agiles
- de nommer quelques exemples de méthodes agiles
- d'intégrer une équipe agile
- de mettre en place une méthode de type agile

1.3 LES DIFFÉRENTS CYCLES DE DÉVELOPPEMENT

Une étude de 1995 montrait que sur 8380 projets, seuls 16 % étaient un succès.



Un tiers des projets ont été abandonnées et le taux de succès décroît avec la taille des projets (et des entreprises).

L'utilisation de méthodes est donc nécessaire pour l'amélioration de la qualité, cependant il est nécessaire d'avoir des indicateurs pour mesurer les écarts entre les besoins et le résultat.

Parmi les critères mesurables, certains sont faciles à mesurer, d'autres, beaucoup moins :

- Fiabilité, robustesse de l'application
- Conformité aux besoins
- Ergonomie, simplicité d'usage (UX)
- Durée et/ou coût de développement
- Adaptabilité, flexibilité du code
- Indépendance avec le matériel (environnement)
- coûts de maintenance...

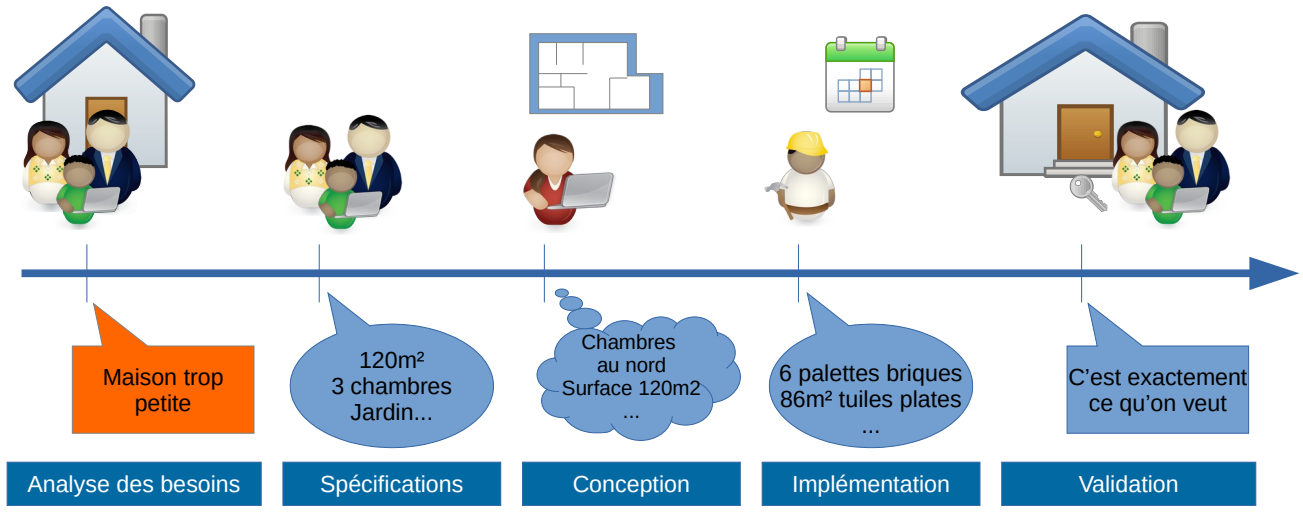
Dans l'industrie, "la qualité du processus de fabrication est garante de la qualité du produit". Il en est de même pour une application logicielle. Il existe d'ailleurs une norme qualité appelée ISO 9000

1.4 DÉFINITIONS

Voici les termes à connaître pour comprendre les différentes phases d'élaboration de logiciel.

Terme	Définition
Analyse des besoins	L'analyse des besoins consiste à recueillir les demandes du client pour avoir un cahier des charges. **On détermine les fonctionnalités attendues**.
Spécifications	Les **spécifications déterminent le périmètre** du logiciel : l'environnement matériel, les contraintes, les objectifs. On décrit les fonctionnalités et les interfaces souhaitées.
Conception	La conception exprime comment le logiciel fonctionne : découpage en entités, relations internes et externes des entités. **La conception correspond aux choix des structures de données, au découpage en parties élémentaires, aux algorithmes et à la réalisation de maquettes ou prototypes**.
Implémentation	**L'implémentation correspond au codage d'une application** : c'est l'écriture du programme dans le langage retenu lors de la conception
Validation	**La validation est la création de tests unitaires et de tests d'intégration.** Lorsque l'ensemble des tests sont passés avec succès, il est possible de livrer tout ou une partie du code. • Tests unitaires : tests effectués sur les modules séparés du prog. • Test d'intégration : tests des interactions entre les modules.
Maintenance	La maintenance est une phase dans laquelle l'application est mise en production. Durant cette phase, les bogues rencontrés sont corrigés, mais il n'y a pas d'ajout de fonctionnalité. On l'appelle aussi phase de support.

Voici un moyen (allégorie) pour se souvenir de chaque étape :

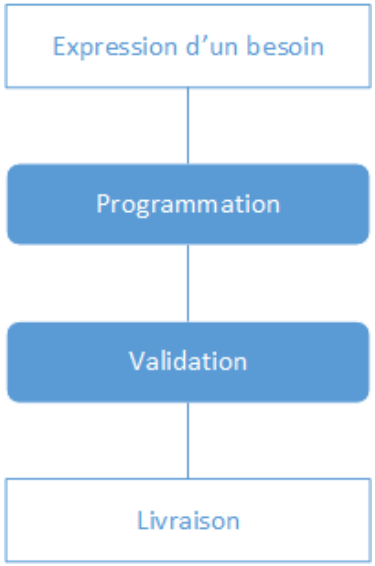


2 LES MÉTHODES LINÉAIRES

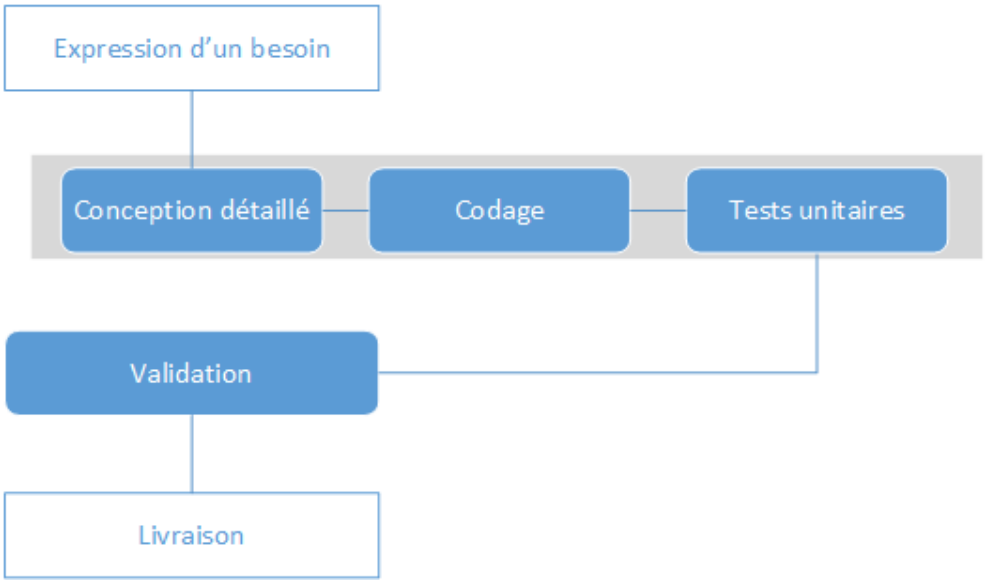
2.1 CODE AND FIX

Dans les années 1950-1960, les programmes sont relativement simples : les machines ne sont pas très évoluées (pas de multitâches, pas de parallélisation) et les interfaces aux utilisateurs sont proposées uniquement à des spécialistes.

Le développement d'une application est totalement linéaire :



Dans les années 1970-1980, la complexité des programmes s'accroît et la phase "Programmation" est scindée en trois nouvelles phases.

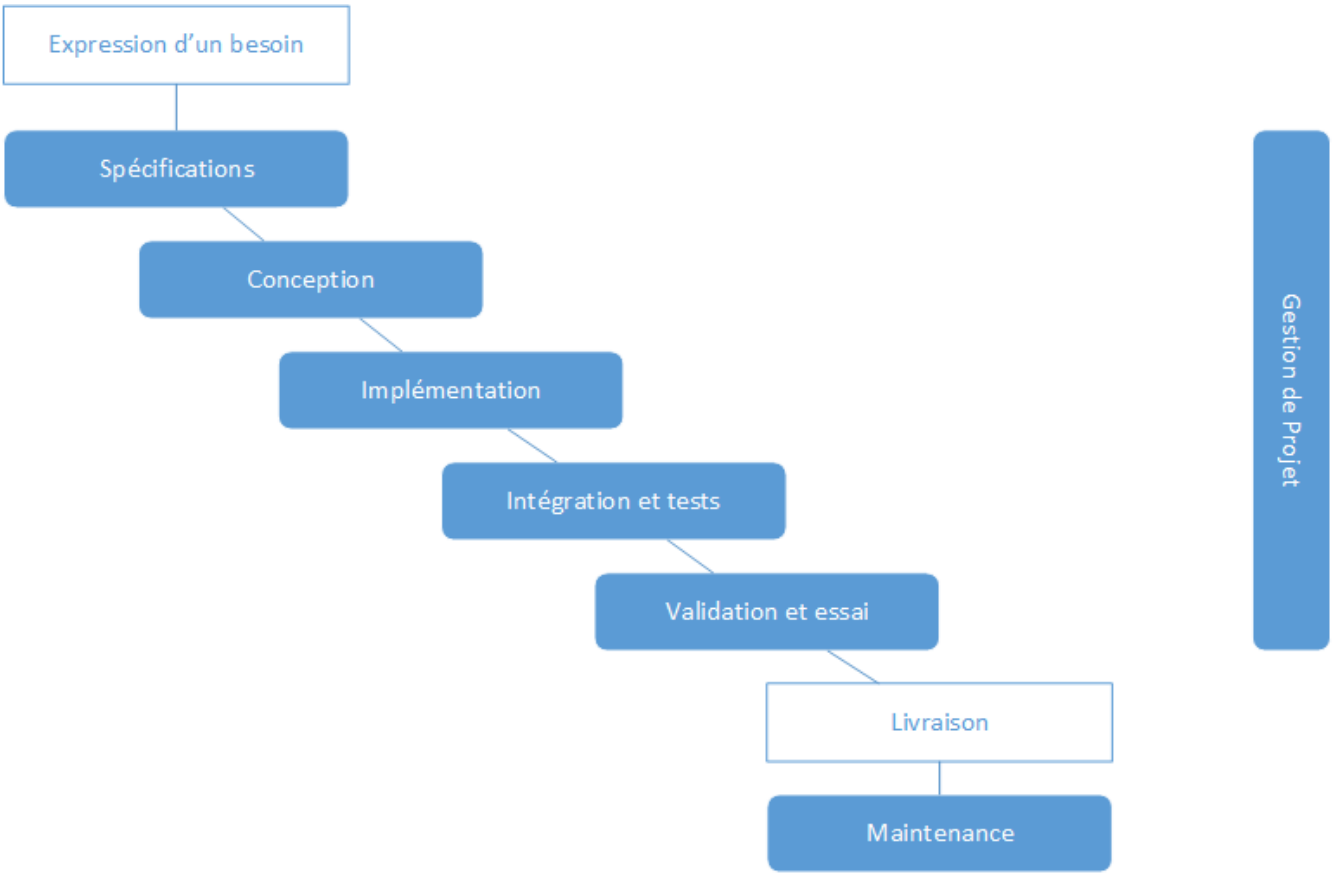


2.2 CYCLE EN CASCADE

C'est la phase la plus connue, qui provient du génie civil : un projet peut avoir des durées importantes et les erreurs initiales peuvent être catastrophiques !

Si les fondations d'un immeuble ne sont pas bonnes, l'électricien ou le plombier ne pourront rien y faire et les corrections seront complexes et coûteuses.

C'est dans ce modèle qu'apparaît le rôle de chef de projet.



Le cahier des charges est enrichi avec deux éléments :

- L'expression des besoins (étude de l'existant, problématique, solutions potentielles, faisabilité)
- Le plan qualité (planification, normes, risques...)

Chaque phase du cycle de vie en cascade peut communiquer avec les phases les plus proches : *la phase "implémentation" peut donc demander des modifications de "conception" ou recevoir des demandes de corrections de la phase "Intégration et tests".*

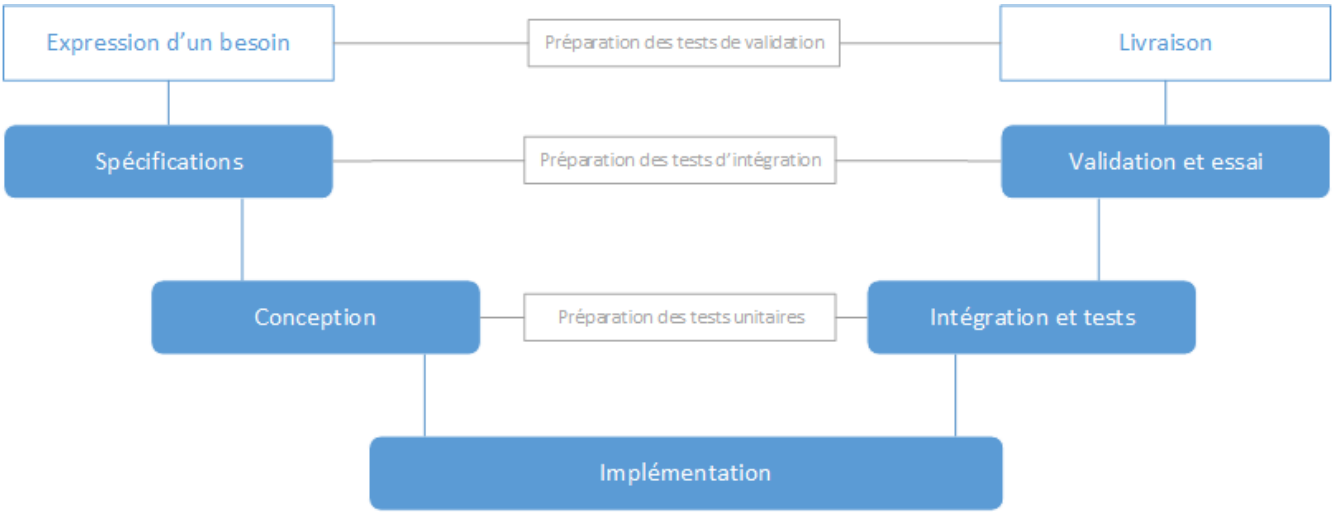
Ce découpage se justifie par le nombre d'intervenant dans le projet : clients, chef de projet, développeurs, concepteurs, etc.

2.3 CYCLE EN V

Cette méthode est développée en 1970 par Winston W. ROYCE mais n'est adaptée à l'informatique qu'en 1990 par deux ingénieurs : Kévin John FORSBERG et Harold MOOZ.

La méthode du cycle de vie en V tente de résoudre un problème simple : dans le cycle en cascade, les gens qui font les tests ne sont pas les mêmes que ceux qui ont conçu le logiciel. De même pour ceux qui valident le logiciel et ceux qui font le cahier des charges.

Il est alors décidé d'employer une méthode permettant travailler par rôle, en créant des couches successives.



L'importance des tests rend la méthode plus fiable que la précédente. De plus, elle favorise l'anticipation et la proactivité.

Malgré tout, cette méthode présente tout de même trois inconvénients majeurs :

- Pour obtenir un résultat correct, elle nécessite la rédaction d'une documentation abondante et très structurée
- Son organisation est lourde, car elle fait intervenir de nombreuses entités (MOA, MOE, MOO, équipes techniques, chef de projet...)
- Elle est très rigide et ne supporte pas les changements : cela remet en cause la phase de conception initiale, il faut donc réécrire la documentation !

3 LES MÉTHODES CYCLIQUES

Bien que les méthodes précédentes soient nommées "cycle en cascade" ou "cycle en V", il n'y a réellement pas de notions de cycle : ce sont des méthodes linéaires !

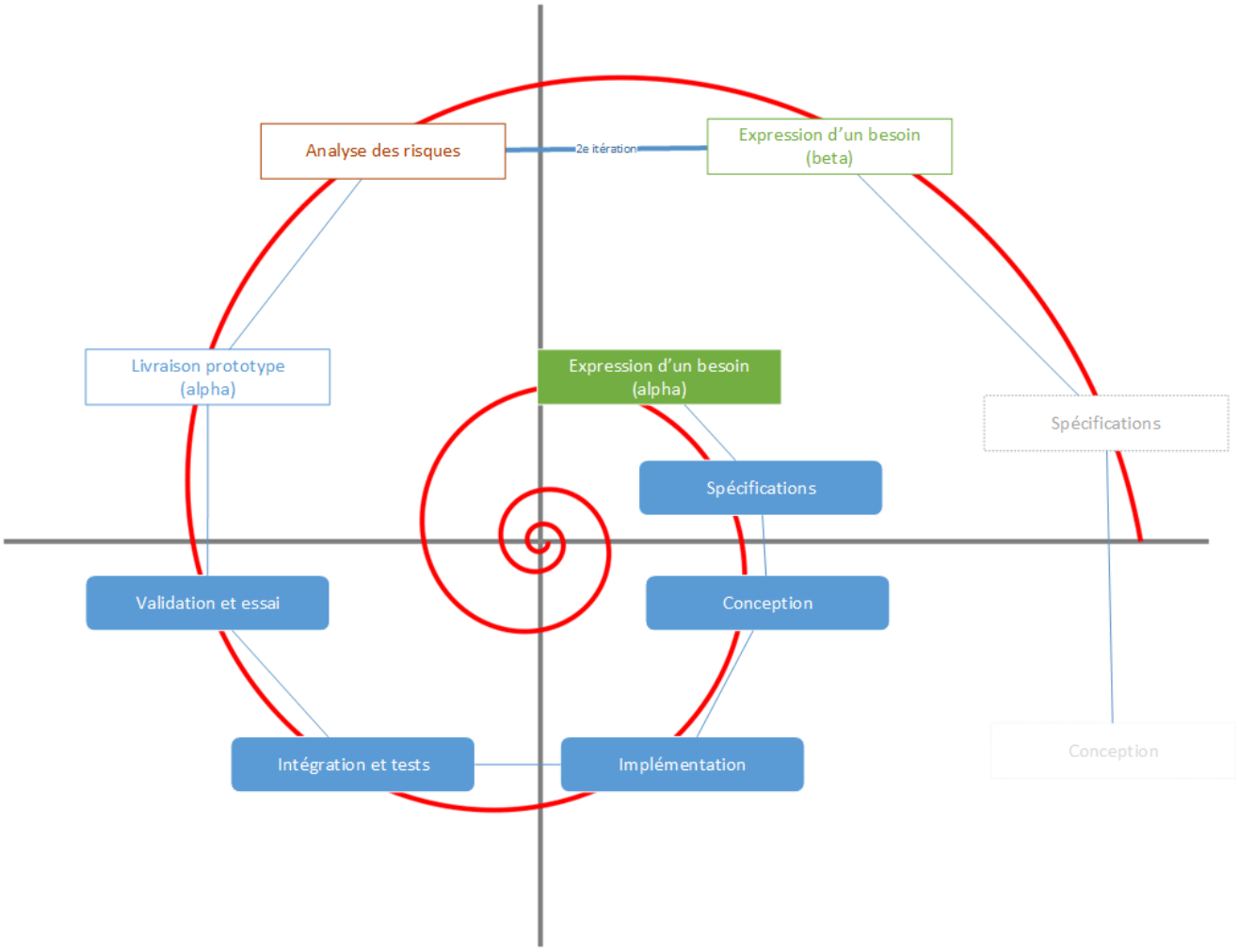
La méthode en spirale et les méthodes Agiles sont des méthodes dans lesquelles la notion de cycle prend toute son importance, car elles sont itératives. Voici le fonctionnement global.

3.1 CYCLE EN SPIRALE

La méthode en spirale reprend les étapes du cycle en V, mais son concepteur, Barry BOEHM prévoit l'implémentation de **prototypes successifs**. La gestion des risques y prend une place prépondérante.

Il ne faut pas mélanger prototype et version :

- Le prototype utilise les notions d'**alpha, beta, release candidate et finale**
- La version est un incrément des fonctionnalités du logiciel : V1, V2…



3.2 Cycle "Agile" (Scrum, Kanban, XP, etc.)

Le développement Agile n'est pas un cycle à proprement parler. Il s'agit plutôt d'une philosophie et d'un état d'esprit. La particularité étant que le développement n'est plus vraiment limité dans le temps : tant que le client demande des modifications, le projet n'est pas clos.

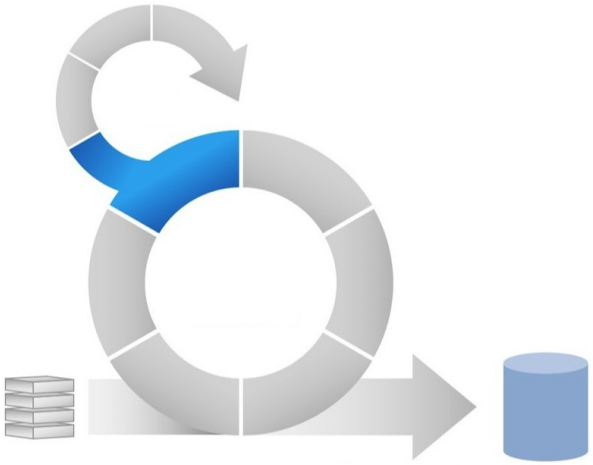
Cependant, Agile propose un fonctionnement cyclique à l'intérieur d'un cycle itératif (comme le cycle en spirale).

L'idée majeure des méthodes agiles est de rendre le développement agile, souple, rapide.

L'idée est donc de soumettre rapidement un travail partiel à la critique du client, qui peut ainsi rapidement valider et permettre soit de passer aux fonctions suivantes, soit de corriger les défauts constatés immédiatement.

Le graphique ci-dessous présente schématiquement les deux cycles dans une méthode Agile (comme SCRUM ou KANBAN) :

- Le grand cercle représente la présentation d'un prototype au client (toutes les 2 semaines par exemple)
- Le petit cercle, représente une itération d'une journée, faite par une personne avec l'équipe, pour déterminer rapidement les points de blocage et l'avancement du cycle global.



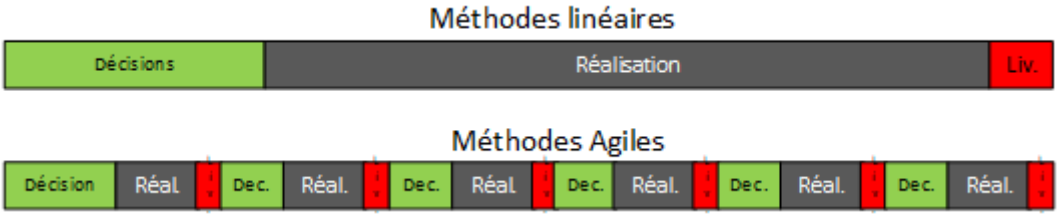
Cependant, avant de décrire plus loin ces méthodes, il faut savoir que la manière dont elles sont implémentées dans les entreprises n'est pas toujours bonne : souvent, elles finissent par bloquer la réussite de l'application, par sa mauvaise gestion de projet[1].

La méthode Agile la plus contraignante (XP, eXtreme Programming) est peut-être la seule à ne pas avoir cet inconvénient.

1 https://www.developpez.com/actu/204756/Les-developpeurs-devraient-abandonner-les-methodes-agiles-selon-Ron-Jeffries-l-un-des-signataires-du-Manifeste-Agile/

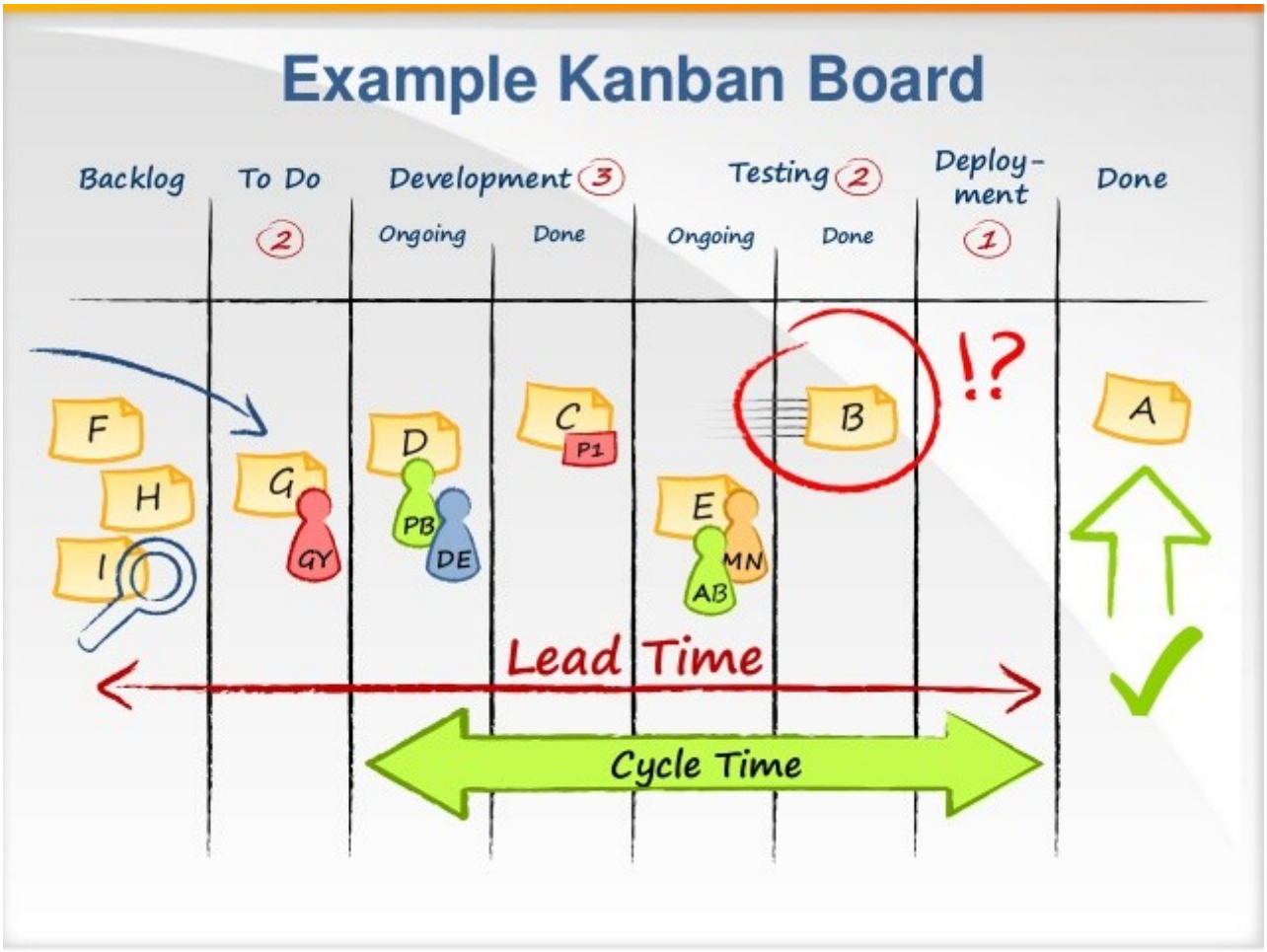
Les principes Agiles privilégient la stimulation positive de l'équipe, en essayant de limiter la pression hiérarchique.

Le principe fondamental de ces méthodes est d'intégrer le client dans le développement sur des phases plus réduites mais plus nombreuses.



En travaillant de cette manière, les écarts entre les attentes du client et le résultat livré sont réduits : le client dispose d'une application très proche de ses besoins (ce qui est primordial).

La plupart de ces méthodes s'appuient sur un tableau contenant les différentes étapes d'une tâche (colonnes) et les tâches (lignes).

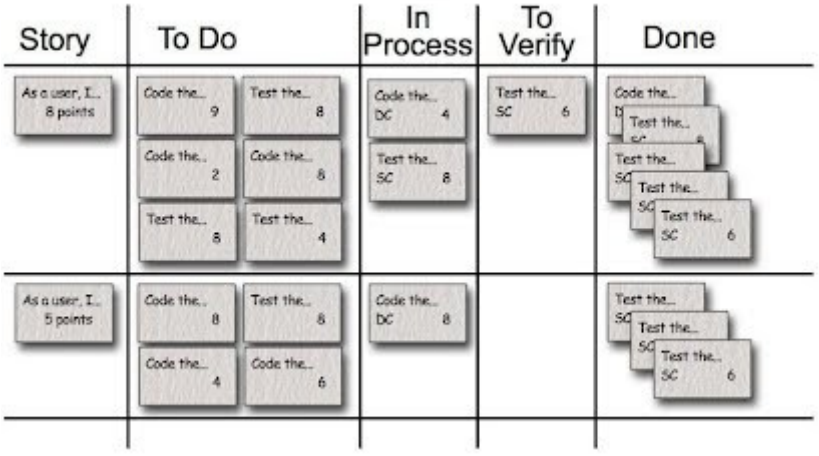


3.2.1 MÉTHODE SCRUM

La méthode **SCRUM** utilise un vocabulaire dédié et un fonctionnement déterminé par les exigences initiales.

Les rôles sont les suivants :

- **Scrum Master** : vérifie que les valeurs et principes de SCRUM sont respectés. Ce n'est pas un chef dans le sens de responsabilité hiérarchique ; Il a plus un rôle de facilitateur (communication, production, difficultés). C'est donc souvent une personne expérimentée.
- **Product Owner** : cet expert métier joue le rôle du client. Il valide les fonctionnalités développées et établit les priorités.
- **Équipe** : chaque membre de l'équipe prend une tâche dans le **backlog** et développe les opérations qui y sont associées. En cas de problème, il peut communiquer à chaque **mêlée** quotidienne (SCRUM).
- **Product Backlog** : c'est le référentiel des besoins établit avec le client. Toutes les fonctions attendues y sont décrites. Pour chaque élément dans le backlog, sera associé une « **user story** ».
- **User Story** : ce sont les éléments qui constituent le backlog. L'histoire pourra être découpée en tâches sur les gros projets. Souvent, elle est identifiée par…
  - Un identifiant unique (ID)
  - Un nom court (de 2 à 10 mots)
  - Une priorité (qui peut changer au cours du projet)
  - Une estimation. L'unité peut être une durée ou un nombre de point et est déterminée par l'ensemble de l'équipe.
  - Une note (contient les détails utiles)



Dans la méthode SCRUM, à chaque fin de cycle, une mise à jour de l'application est fournie au client (sous la forme d'une démo), uniquement avec les stories complètes.

3.2.2 Méthode Kanban

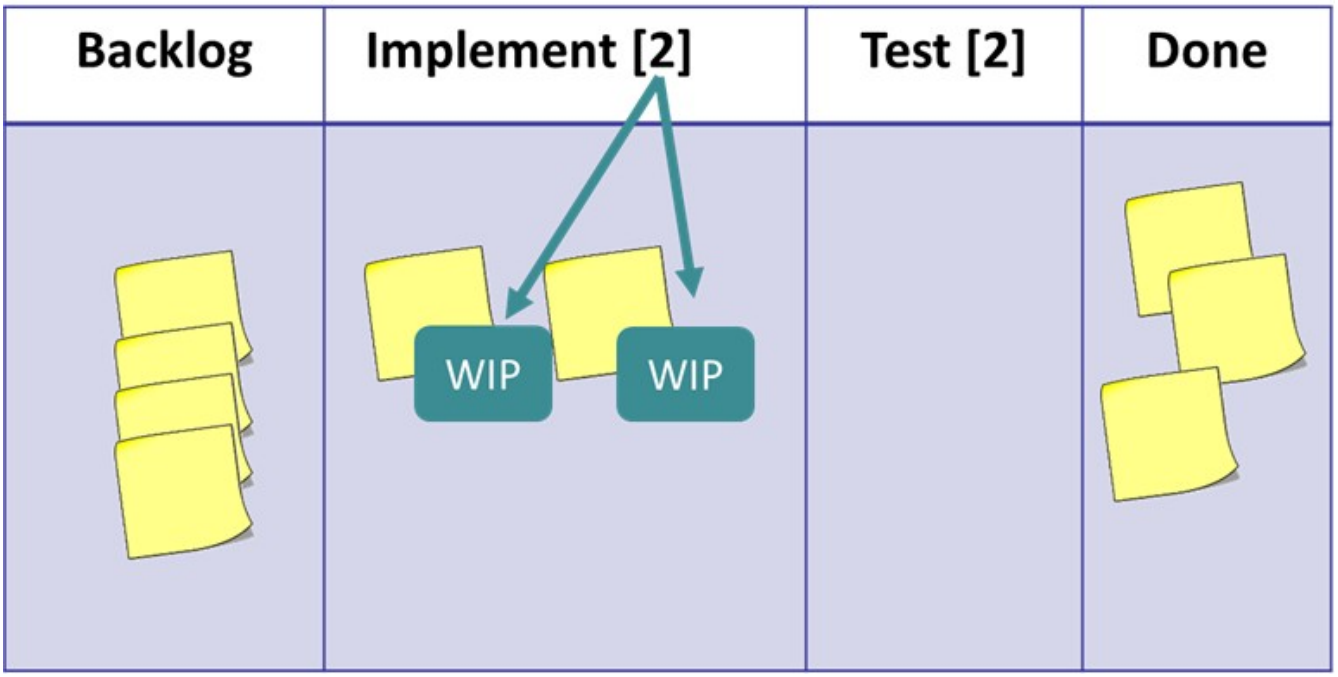
La méthode **Kanban** partage quelques éléments avec **Scrum** mais propose également des différences de fonctionnement.

3.2.2.1 Les rôles communs :

- **Agile Coach** : son rôle est similaire au Scrum Master… mais il n'est pas nécessaire dans la méthode Kanban.
- **Product Owner** : joue toujours le rôle de client.
- **Équipe** : ne change pas.
- **Product Backlog** : référentiel des besoins énoncés par le client.

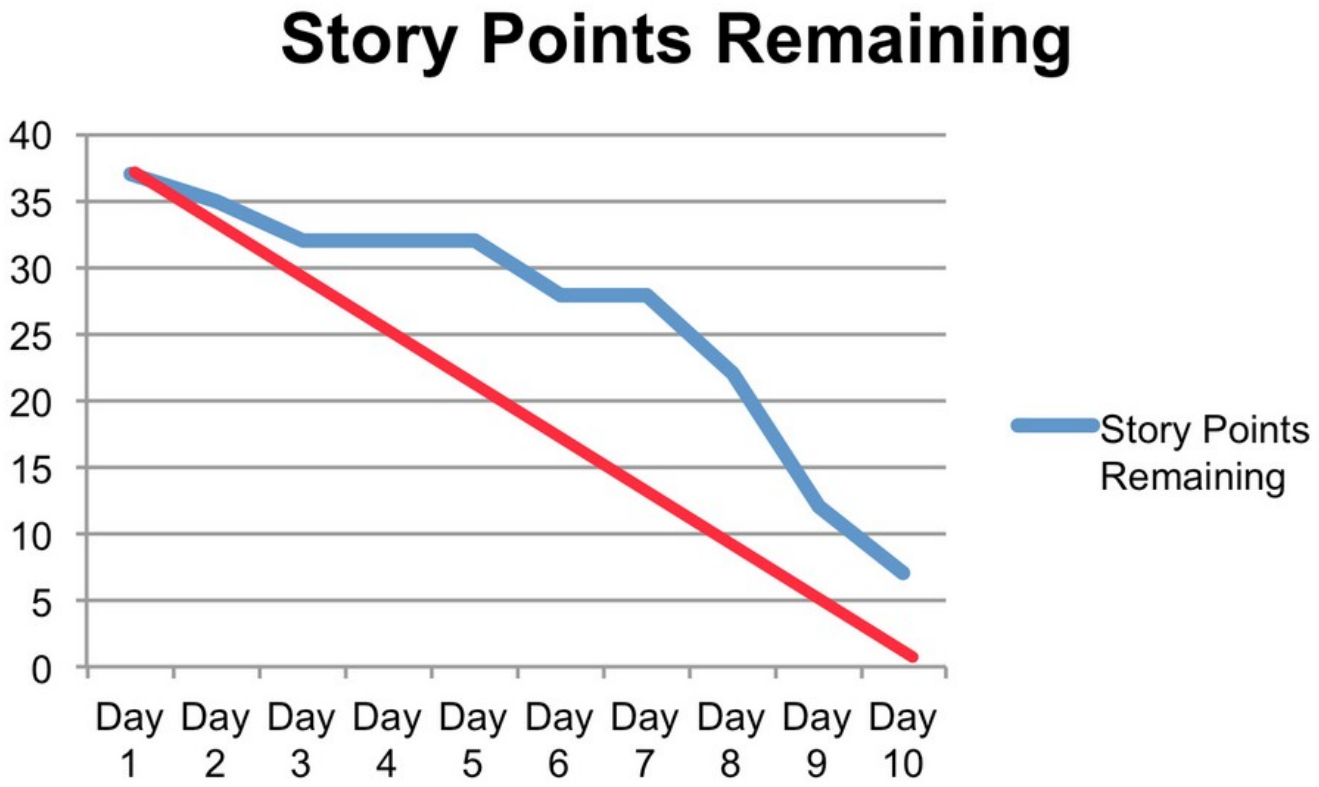
3.2.2.2 Les différences :

- Kanban n'a pas de cycle, à proprement parler. Dès qu'une fonctionnalité est prête, une démo est construite.
- Kanban n'imposant pas de limite sous forme de cycle, la limitation du workflow (flux de travail) se fait sur la quantité d'éléments présents dans la colonne « build » : c'est le **Work In Progress** (WIP).
- Il n'y a pas de « Sprint backlog » alimenté par le Scrum Master et le Product Owner : les tâches sont donc placées directement dans la colonne « build » par le coach et l'équipe.



3.2.3 BURNDOWN CHART (GRAPHIQUE D'AVANCEMENT)

C'est un graphique qui montre le nombre de tâches (stories) initialement, jusqu'à la fin programmée du projet. Mais il faut reporter la réalité sur le graphique pour déterminer le déroulement (avancement) du projet.



Il existe un outil en ligne pour générer un fichier PDF automatiquement.

http://www.burndowngenerator.com/

4 SOURCES

Génie Logiciel : principes et techniques (Pierre Gérard, IUT de Villetaneuse, 2007-2008)

http://www-lipn.univ-paris13.fr/~gerard/docs/cours/gl-cours-slides.pdf

https://www.groupeozitem.com/cycle-en-v/

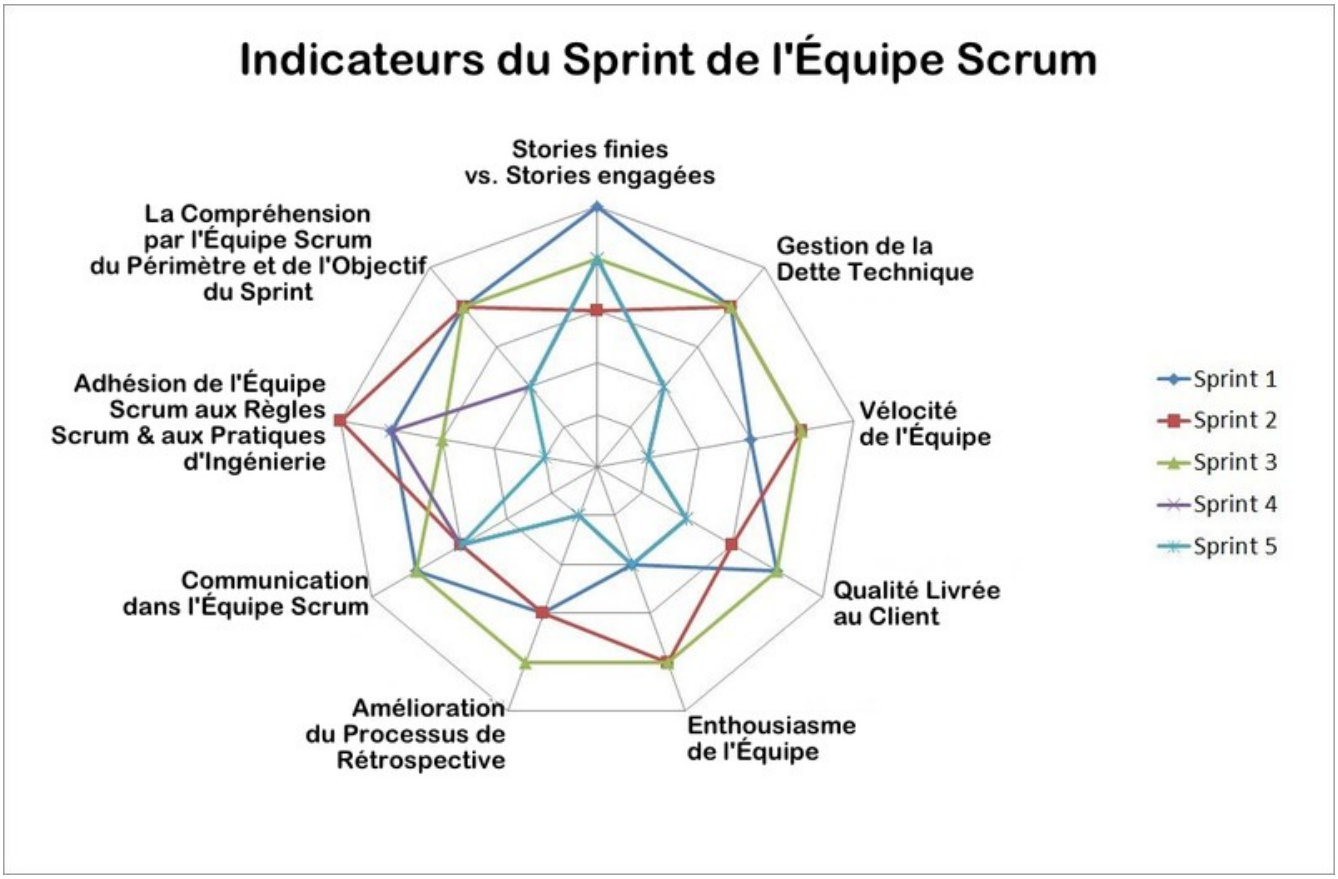
http://artisandeveloppeur.fr/les-12-principes-du-manifeste-agile/

http://www.responsive-mind.fr/cycles-developpement-informatique/

https://medina.developpez.com/cours/extreme-programming/

4.1 INDICATEURS SCRUM

Bien que SCRUM soit un travail d'équipe sans pression hiérarchique, il n'est pas rare que des indicateurs soient mis en place pour mesurer l'efficacité de l'équipe… ce qui revient à réintroduire la pression.



4.2 DIFFÉRENCE SCRUM VERSUS KANBAN

Gary, l'auteur de « Developement that pays » explique la différence entre les deux méthodes.

https://www.developmentthatpays.com/posts/68-difference-between-scrum-and-kanban?ref=cheatsheet