

Initiation aux méthodes agiles

Gestion de projets :

Méthodes "prédictives" : Tout est planifiable avant le début du projet

- Cycle en cascade
- Cycle en V
- Cycle en Y
- Modèle en spirale

▼ Cycle en cascade

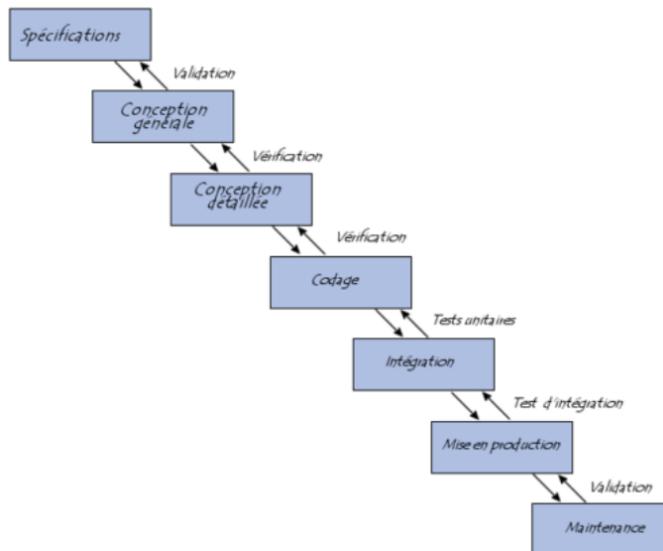
→ Présenté en 1970, ce modèle est hérité du BTP, où les projets doivent suivre une séquence très précise et rigide.

Ce modèle se base sur deux principes :

- Une étape ne peut pas être débutée avec la fin de la précédente
- La modification d'une étape du projet a un impact important sur les étapes suivantes

Le modèle en cascade se compose de 7 phases, qui se succèdent :

- Spécification → Conception générale → Conception détaillée → Codage
→ Intégration → Mise en production → Maintenance

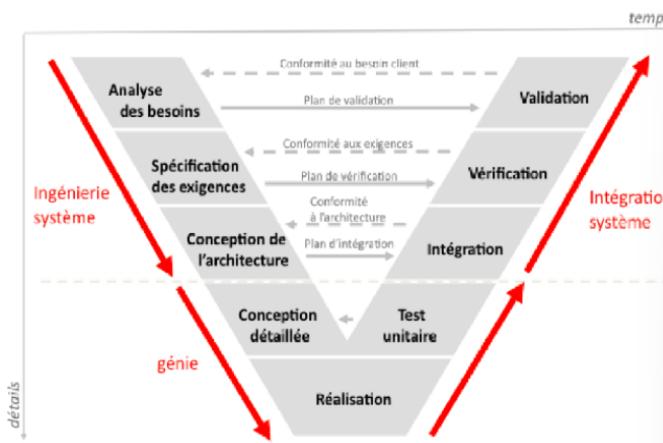


▼ Le cycle en V

→ Créé dans les années 80 est une variante du modèle en cascade, qui cherche à résoudre certaines de ses limitations majeures

Comment le cycle en V s'inspire du modèle en cascade ?

- Le cycle en V est une approche séquentielle. Chaque phase de conception doit être terminée avant de passer à la suivante.
- Les étapes de conception et de développement sont très similaires dans les deux modèles.



La MOA (Maîtrise d'Ouvrage) :

- La **MOA** (Maîtrise d'Ouvrage) commande le projet et définit les besoins et objectifs.

- Elle valide la solution proposée et représente le client, sans être l'utilisateur final.
- Elle est souvent assistée par l'**AMOA** (Assistance à Maîtrise d'Ouvrage) pour clarifier les besoins et suivre la réalisation du projet.

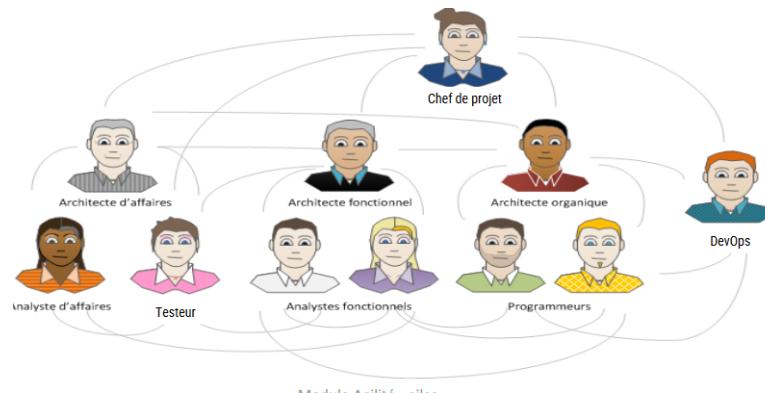
La MOE (Maîtrise d'Œuvre) :

- est l'entité qui réalise le projet pour le compte de la MOA.
- Elle est représentée par le chef de projet.

Le comité de pilotage :

- est une instance transverse qui vérifie l'atteinte des jalons du projet.
- Il est composé de représentants de la MOA, de la MOE et du client.

→ Les différents acteurs du projet ont des rôles et des responsabilités différentes



Avantages :

- Simplicité du modèle
- Le planning est prévu à l'avance
- Facilement contractualisable
- Documentation complète
- Idéal pour des projets à exigences stables

Inconvénients :

- Faible tolérance à l'erreur
- Tout est anticipé, aucune possibilité de s'adapter
- Aucune innovation possible
- Réalisation qui peut s'éloigner de l'objectif du client
- Risque de découverte tardive de spécifications fausses, incomplètes

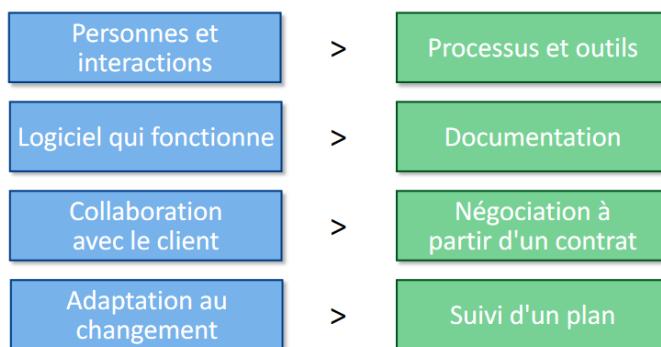
▼ Pourquoi l'Agilité

- Le modèle agile a été introduit pour répondre aux limites des modèles traditionnels comme le cycle en cascade.
- L'agilité est une méthode de gestion de projet qui privilégie la flexibilité, l'adaptation aux changements et la collaboration avec les parties prenantes. Elle repose sur des cycles courts (sprints) pour livrer progressivement des produits de valeur, tout en recueillant régulièrement des retours.

Les 4 valeurs fondamentales :

- **Les individus et leurs interactions plutôt que les processus et les outils**
 - Équipe idéalement colocalisée
 - Communique au maximum et bonne entente
 - Pluridisciplinaire
 - Collaboration au quotidien
 - Amélioration continue
- **Des logiciels opérationnels plutôt qu'une documentation exhaustive**
 - Livraisons fréquentes et incrémentales
 - Documentation légère mais suffisante
 - Améliorations continues basées sur les retours
 - Prototypes utilisables
 - (*Fonctionnel et répond au besoin client (ni plus ni moins)*)
- **La collaboration avec les clients plutôt que la négociation contractuelle**

- Échanges réguliers et constructifs
- Adaptation aux priorités changeantes
- Partenariat plutôt que relation fournisseur/client
- Validation continue
- **L'adaptation au changement plutôt que le suivi d'un plan**
 - Revoir les priorités à chaque itération
 - Flexibilité face aux imprévus
 - S'adapter rapidement aux nouvelles opportunités
 - Plans flexibles et révisables
 - Anticipation des risques



Vous avez raison, voici la version complète et structurée avec l'ajout de ce point important :

12 Principes Agiles

1. Satisfaction du client

- **Notre plus haute priorité est de satisfaire le client** en livrant rapidement et régulièrement des fonctionnalités à forte valeur ajoutée.

2. Amélioration continue

- À intervalles réguliers, l'équipe réfléchit à ses méthodes de travail pour devenir plus efficace, puis ajuste son comportement en conséquence.

3. Auto-organisation des équipes

- Les meilleures architectures, spécifications et conceptions émergent d'équipes auto-organisées, qui sont autonomes dans leurs décisions.

4. Simplicité

- L'agilité valorise la simplicité, c'est-à-dire l'art de minimiser le travail inutile, afin de se concentrer sur l'essentiel.

5. Excellence technique et bonne conception

- Une attention continue à l'excellence technique et à la qualité de la conception renforce l'agilité et permet de livrer des produits de meilleure qualité.

6. Réactivité aux changements

- Les processus agiles accueillent positivement les changements de besoins, même tard dans le projet, et utilisent ces changements pour offrir un avantage compétitif au client.

7. Livraison fréquente

- L'agilité priviliege des cycles de livraison courts (de quelques semaines à quelques mois), avec une préférence pour les cycles les plus courts, afin de livrer rapidement des logiciels opérationnels.

8. Collaboration continue

- Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet, favorisant ainsi la communication et l'adaptation aux besoins.

9. Motivation de l'équipe

- L'équipe est motivée et soutenue par l'organisation, qui lui fait confiance pour atteindre les objectifs fixés.

10. Communication efficace

- Le dialogue en face-à-face est la méthode la plus simple et la plus efficace pour transmettre des informations à l'équipe de développement et au sein de celle-ci.

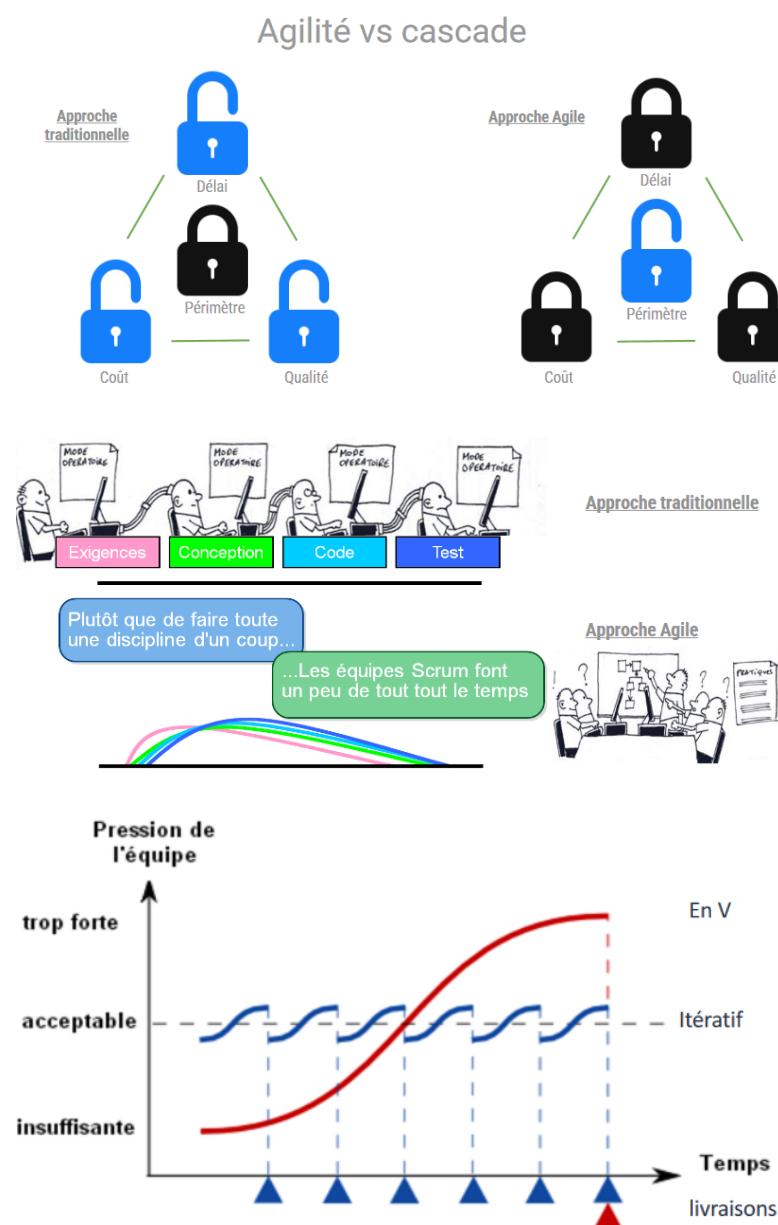
11. Rythme soutenable

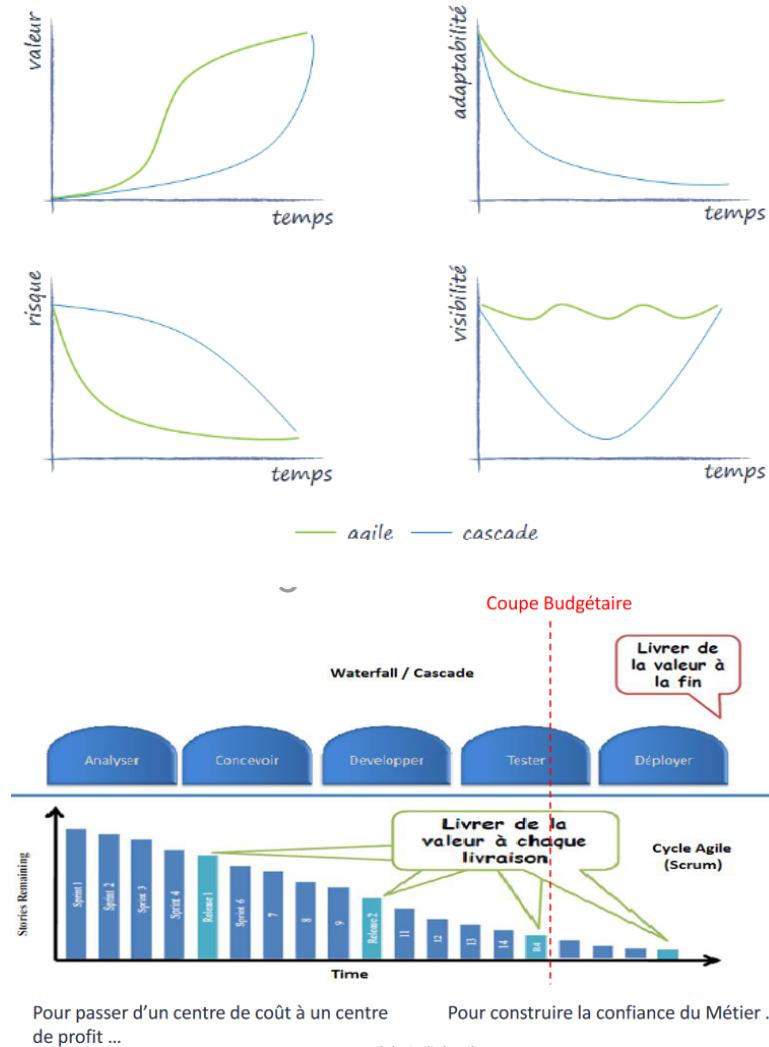
- Les processus agiles encouragent un rythme de développement soutenable, permettant à l'équipe, aux commanditaires et aux utilisateurs de maintenir un rythme constant sur le long terme.

12. Mesure de l'avancement

- Un logiciel opérationnel est la principale mesure d'avancement, plutôt que des documents ou des spécifications.

▼ Comparaison des méthodes prédictives et des méthodes agiles - Agilité vs Cascade





- **Transformation du rôle de “chef” vers un rôle de coach**
 - Commander → Faire confiance
 - Contrôler → Faciliter
 - Diriger → Accompagner



**Représentation de l'évolution
du rôle de 'chef' vers un rôle de coach**

Représentation de l'évolution du rôle de 'chef' vers un rôle de coach

- **Changement dans le rôle du client**

- Commander → Co-construire
- Spécifier → Echanger
- Diriger → Accompagner



Le client est présent tout au long de la production

- Transformation du rôle de développeur

- Exécuter → Produire
- Impliquer en fin → Impliquer dans tout le processus
- Main-d'œuvre → Autonome

Software Developer



L'équipe de développement est pluridisciplinaire

Comparaison Méthodes Agiles vs Traditionnelles

Critères	Agiles	Traditionnelles
Processus	Itératif, incrémental et adaptatif	Linéaire, étapes séquentielles, sans rétroaction
Livraison	Livraison continue de petites parties fonctionnelles	Livraison unique en fin de projet
Flexibilité	Très flexibles, adaptation à chaque itération	Peu flexible, difficile à modifier
Implication du client	Participation active et régulière	Participation limitée au début et à la fin du projet
Planification	Planification évolutive, adaptée à chaque sprint	Planification rigide définie dès le départ
Structure de l'équipe	Pluridisciplinaire et auto-organisée	Hiérarchique, rôles clairement définis
Gestion des changements	Facile, intégrée à chaque itération	Difficile, changements couteux
Qualification/Test	Le plus régulièrement possible (chaque fin de sprint)	À la fin du projet
Documentation	Documentation minimale, juste suffisante	Documentation complète et détaillée
Objectifs	Satisfaction du client par livraison de valeur métier	Respect de l'engagement

L'agilité face à l'échec des projets :

- Mauvaise définition des besoins
- Manque de flexibilité face aux changements
- Problèmes de communication dans l'équipe
- Découverte tardive des erreurs
- Risque de sur-engagement des ressources
- Difficultés à gérer les priorités

▼ Les principales approches agiles

1. Agile Scrum Methodology
2. Scaled Agile Framework (SAFe)
3. Lean Software Development
4. Kanban
5. Extreme Programming (XP)
6. Feature Driven Development (FDD)
7. Dynamic Systemes Development Method (DSDM)
8. Crystal Methodology

▼ Kanban

- est une méthode agile de gestion de projet qui utilise un tableau visuel pour suivre et gérer le flux des tâches.
 - Les tâches sont représentées par des cartes, qui se déplacent à travers différentes colonnes correspondant aux étapes du processus.
- Les phases par défaut sont les suivantes :

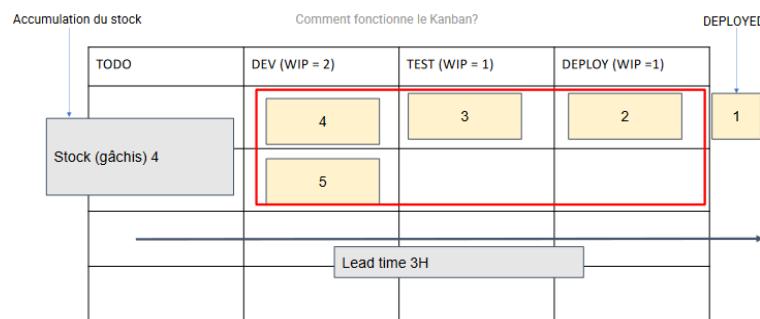
- **À faire** (Backlog ou To Do)
- **En cours** (Work In Progress - WIP ou Doing)
- **Terminé** (Done)

Comment fonctionne le Kanban?

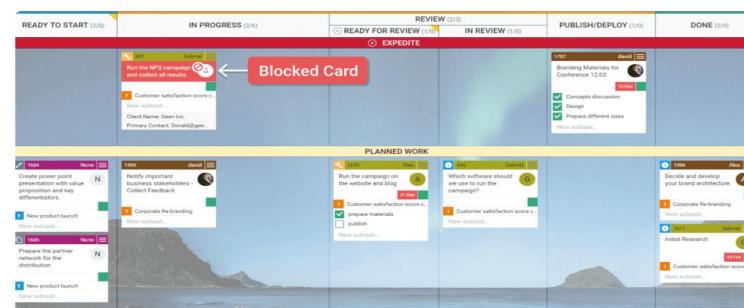
TODO	DEV (WIP = 2)	TEST (WIP = 1)	DEPLOY (WIP =1)
	3	2	1
5	4		

Comment fonctionne le Kanban?

TODO	DEV (WIP = 2)	TEST (WIP = 1)	DEPLOY (WIP =1)	DEPLOYED
	4	3	2	1
	5			



- Exemple d'un Tableau Kanban dans un vrai projet:



Objectifs et principes de Kanban :

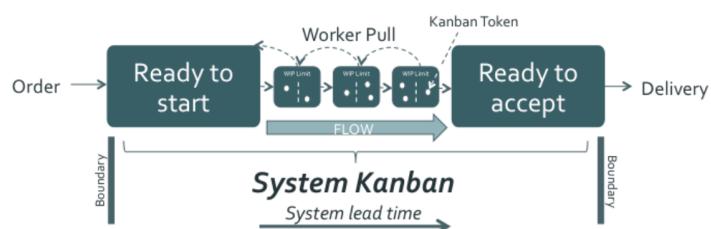
- Objectifs

- Améliorer l'efficacité du flux de travail
- Maximiser la productivité
- Amélioration continue
- Réduire les temps de cycle

- Principes

- Commencer avec ce que vous faites actuellement
- S'engager dans une amélioration continue
- Respecter les processus et les rôles actuels
- Visualiser le travail
- Limiter le travail en cours (WIP) et gestion du flux

Workflow global de la méthode kanban



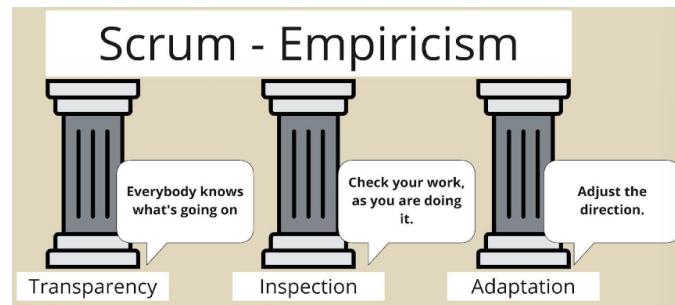
▼ Scrum

Qu'est ce que c'est Scrum ?

Scrum est une méthode agile visant à délivrer la plus grande valeur métier dans les délais les plus courts.

- Un produit fonctionnel est livré à la fin de chaque **sprint**, qui dure généralement de 2 à 4 semaines.
- Les priorités métier sont définies par le client ou le propriétaire du produit, tandis que l'équipe s'organise de manière autonome pour décider de la meilleure façon de répondre à ces priorités.
- À la fin de chaque sprint, le produit en cours est **présenté et évalué**. L'équipe et les parties prenantes peuvent alors décider soit de le **livrer tel quel**, soit de continuer à l'améliorer lors d'un sprint supplémentaire.

Les piliers de Scrum ?



1. Transparency :

- Les éléments clés du processus doivent être **visibles** à tous les participants.
- La transparence implique l'établissement d'un **standard commun**, notamment une définition partagée de ce qui est considéré comme "**Fini**" (Definition of Done).
- Scrum met un accent particulier sur l'utilisation d'un **langage commun** entre l'équipe et la direction, afin d'assurer une compréhension claire et partagée.

2. Inspection :

- Les utilisateurs de Scrum doivent régulièrement passer en revue les artefacts et l'état d'avancement par rapport aux objectifs afin de détecter les écarts indésirables.

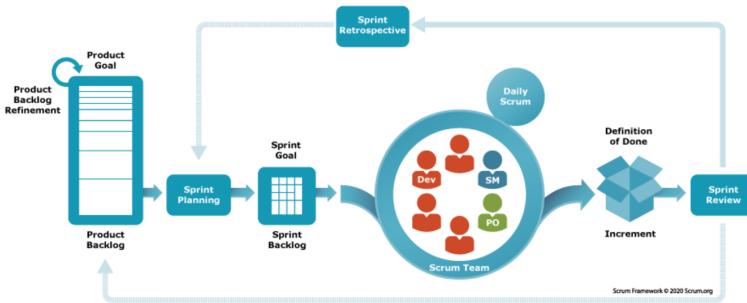
3. Adaptation :

Si une dérive est identifiée lors de l'inspection, le processus ou le produit doit être ajusté. Scrum prévoit plusieurs **rituels** permettant ces ajustements, à savoir :

- **La réunion de planification de sprint** (Sprint Planning)
- **La mêlée quotidienne** (Daily Scrum)
- **La revue de sprint** (Sprint Review)
- **La rétrospective de sprint** (Sprint Retrospective)

Ces réunions offrent des opportunités régulières pour adapter le travail et améliorer les processus.

- Cycle SCRUM :



- User Story:

Une **User Story** (histoire utilisateur) est une description simple d'une fonctionnalité ou d'un besoin du point de vue de l'utilisateur final.

Elle est utilisée en **Scrum** et autres méthodologies agiles pour capturer les attentes de l'utilisateur par rapport au produit.

Les User Stories suivent généralement ce format :

- "En tant que [rôle], je veux [objectif] afin de [bénéfice]"

Exemple :

"En tant qu'utilisateur enregistré, je veux pouvoir changer mon mot de passe afin de sécuriser mon compte."

▼ Le cadre Scrum

▼ Rôles :

- Product Owner
- Scrum Master
- Dev Team

▼ Product Owner

1. Le **Product Owner (PO)** dans Scrum est responsable de **maximiser la valeur du produit** et de ce qui sera fait par la

DevTeam en gérant le **product backlog**, dont il est le seul responsable.

Que comprend la gestion du product backlog ?

- **Exprimer clairement** les éléments du backlog.
- **Prioriser les items** pour atteindre les objectifs du projet.
- S'assurer que le backlog soit **visible, transparent et clair** pour tous, en indiquant ce sur quoi l'équipe travaillera ensuite.
- S'assurer que la **DevTeam** comprend bien les items du backlog avant de commencer à travailler dessus.

2. Le **PO** est responsable du "**Quoi**", c'est-à-dire de ce qui doit être fait, mais pas du "**Comment**", qui relève de la **DevTeam**.

En tant que PO, il ne faut pas :

- Estimer les items à la place de la DevTeam.
- Décider de la conception technique ou du découpage des tâches.
- Déterminer l'ordre des tâches dans le Sprint Backlog (cela revient à la DevTeam).
- Manager l'équipe (la DevTeam est auto-organisée).

▼ Scrum Master

Le **Scrum Master** a pour rôle de faciliter l'application de **Scrum** et de garantir que l'équipe respecte les valeurs et pratiques agiles.

Responsabilités principales :

- Maîtriser **Scrum** et l'agilité.
- Représenter le management du projet et gérer le **processus Scrum**, mais ne pas gérer la **DevTeam** (pas de rôle hiérarchique).

- Faire appliquer les **valeurs et pratiques Scrum** au sein de l'équipe.
- **Éliminer les obstacles** qui freinent la DevTeam dans son travail.
- S'assurer que l'équipe est **fonctionnelle et productive**.
- Faciliter la **coopération** entre tous les rôles du projet.
- **Protéger l'équipe** des interférences extérieures.
- Veiller à ce que les **cérémonies agiles** soient correctement réalisées (avec respect des time-box) et se déroulent dans de bonnes conditions (animation des réunions).

▼ Dev Team

L'équipe de développement est composée de professionnels responsables de la livraison d'un **incrément "Fini"** et potentiellement publiable à la fin de chaque sprint.

Principales responsabilités et caractéristiques :

- **Responsabilité** : L'équipe est responsable de la création d'un incrément **potentiellement publiable**.
- **Autonomie** : Elle est **auto-organisée** et gère son propre travail.
- **Cross-fonctionnelle** : L'équipe possède des compétences variées pour accomplir toutes les tâches nécessaires.
- **Collaboration** : Elle collabore avec le Product Owner et résout ses conflits de manière autonome.
- **Pas de hiérarchie** : Il n'y a pas de rôles hiérarchiques ou prédéfinis, chaque membre peut prendre en charge diverses activités.

Spécificités de l'équipe agile

- **Effectif** : L'équipe doit compter entre **3 et 8 personnes**.

- **Rôles diversifiés** : Architecte, Concepteur, Développeur, Spécialiste IHM, Testeur
- **Engagement** : Les membres de l'équipe devraient idéalement être **dedans à plein temps** sur le projet.
- **Auto-organisation** : L'équipe s'organise **de manière autonome** pour gérer son travail.
- **Stabilité** : La **composition de l'équipe** doit rester **invariable** pendant toute la durée d'un sprint.

▼ Les artefacts



▼ Backlog Produit

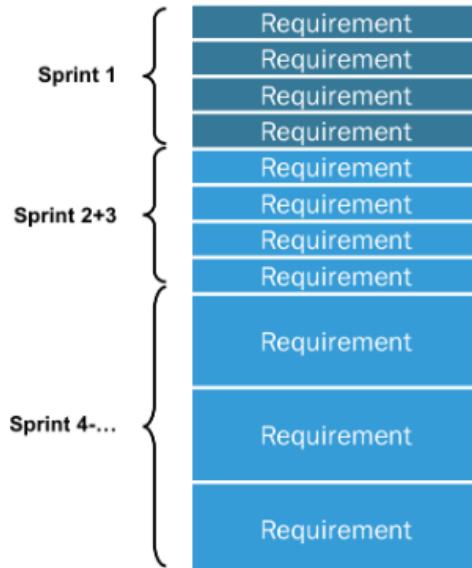
- Le **Backlog Produit** est une liste priorisée de toutes les fonctionnalités et tâches nécessaires au produit.
- C'est la **seule source d'exigences** pour les changements à apporter au produit.
- Chaque élément du backlog doit apporter **de la valeur** aux utilisateurs ou aux clients.
- Le **Product Owner** est responsable du **Backlog Produit**, y compris de son contenu, de sa mise à jour et de son ordre de priorité.

Exemple :

Elément de backlog	Estimation
Un invité peut faire une réservation	3
En tant qu'invité, j'annule une réservation	5
En tant qu'invité, je change les dates d'une réservation.	3
En tant qu'employé de l'hôtel, je produis les rapports de revenu par chambre	8
Améliorer la gestion des exceptions	8
...	30
...	50

▼ Backlog Sprint

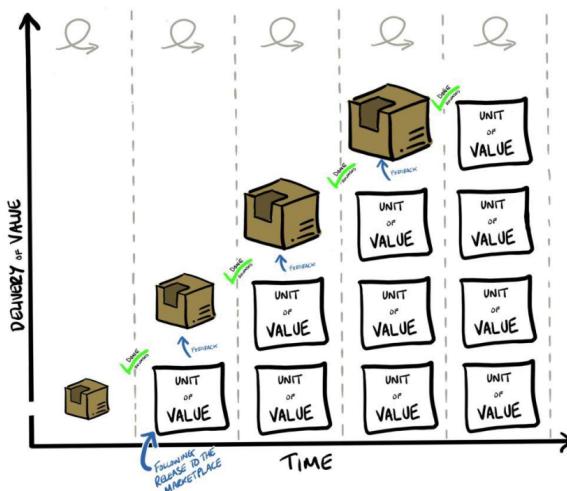
- Le **Backlog Sprint** est la liste des éléments sélectionnés pour le sprint, accompagnée d'un plan pour livrer l'incrément du produit et atteindre l'objectif du sprint.
- C'est une **prévision** faite par l'équipe de développement sur les fonctionnalités à intégrer dans le prochain incrément et le travail nécessaire pour les rendre "Fini".
- Le **Backlog Sprint** rend visible tout le travail identifié par l'équipe pour **atteindre l'objectif du sprint**.



▼ Incrémentation du produit

- Un **incrément** dans Scrum est un résultat concret ou un ensemble de fonctionnalités complètes, développées et testées pendant un sprint.

- Chaque incrément représente un **ajout au produit**, **potentiellement livrable**, ce qui signifie qu'il peut être livré aux utilisateurs ou parties prenantes, même si ce n'est pas encore la version finale du produit.
- L'objectif de l'incrément est de permettre une **livraison régulière** à la fin de chaque sprint, avec de la valeur pour le **métier**.



▼ Les cérémonies

Il y a 4 cérémonies principales en Scrum :

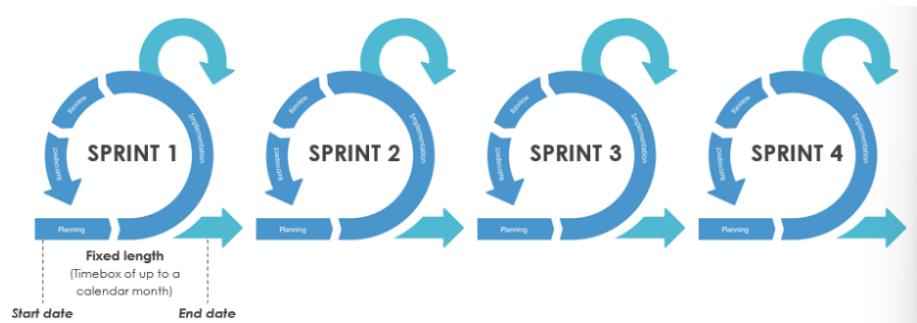
1. **Sprint Planning (Planification du sprint)** : Réunion pour définir les tâches et objectifs du sprint à venir.
2. **Daily Scrum (Mêlée quotidienne)** : Réunion rapide de 15 minutes chaque jour pour synchroniser l'équipe.
3. **Sprint Review (Revue de sprint)** : Réunion en fin de sprint pour examiner l'incrément produit et obtenir des retours.
4. **Sprint Retrospective (Rétrospective de sprint)** : Moment de réflexion pour analyser ce qui a bien fonctionné et ce qui doit être amélioré.

Bien que non officielle, le **Product Backlog Refinement** (affinage du backlog produit) est recommandé pour tenir à jour et bien prioriser le backlog.

▼ Sprint

Le **Sprint** est le cœur de Scrum. C'est une période de 2 à 4 semaines (time-box) durant laquelle un **incrément fonctionnel et "Fini"** est créé, et qui peut être potentiellement livré.

- La durée des sprints reste **constante** pendant la phase de développement.
- Un **nouveau sprint commence immédiatement** après la fin du précédent.
- Pendant le sprint, toutes les **cérémonies Scrum** sont réalisées.



▼ Sprint Planning

Le **Sprint Planning** est une cérémonie clé qui marque le début de chaque sprint.

→ Son objectif est de définir ce qui sera accompli pendant le sprint à venir et comment l'équipe va s'y prendre.

Lors de cette réunion, l'équipe se réunit pour :

- **Définir les objectifs du sprint**
- **Créer le Sprint Backlog**
- **Estimer le travail nécessaire**
- **Déterminer comment le travail sera accompli**



▼ Daily Scrum (mêlée quotidienne)

Le **Daily Scrum** est un événement **time-boxé de 15 minutes**, réservé à l'équipe de développement.

- **Fréquence** : Il a lieu chaque jour pendant le sprint.
- **Objectif** : L'équipe planifie le travail pour les **24 heures suivantes**, inspecte ce qui a été accompli depuis la dernière mêlée et anticipe le travail restant jusqu'à la fin du sprint.
- Le daily se tient **à la même heure et au même endroit** chaque jour pour simplifier l'organisation.
- Tout le monde est **invité**, mais seuls les membres de l'équipe de développement **parlent**.
- **But** : Ce n'est pas une réunion pour résoudre des problèmes, mais pour optimiser la collaboration et éviter la nécessité d'autres réunions.

Chacun répond à 3 questions

Qu'as-tu fait hier ?

Que vas-tu faire aujourd'hui ?

Y a t-il un obstacle qui te freine ?

▼ Sprint Review

La **Revue de Sprint** se tient à la fin de chaque sprint pour inspecter l'incrément réalisé et, si nécessaire, adapter le **Backlog Produit**.

- L'équipe Scrum et les parties prenantes échangent sur ce qui a été accompli et présentent une **démonstration** des nouvelles fonctionnalités ou de l'architecture.
- La revue est **informelle**, dure **moins de deux heures** et ne demande pas de préparation préalable.
- **Toute l'équipe** est invitée, ainsi que les **managers** et **autres équipes** pour discuter de l'avancement du projet.

▼ Sprint Retrospective

La **Rétrospective de Sprint** offre à l'équipe Scrum l'occasion de s'auto-inspecter et de définir un plan d'améliorations pour le sprint suivant.

- L'équipe réfléchit à ce qui a **bien fonctionné** et à ce qui peut être **amélioré**.
- Elle dure généralement entre **15 et 30 minutes**.
- Elle se fait à la **fin de chaque sprint**.
- **Toute l'équipe** participe : Scrum Master (SM), Product Owner (PO) et DevTeam.

Toute l'équipe collecte du feedback et discute sur ce qu'elle aimera :

Commencer à faire

Arrêter de faire

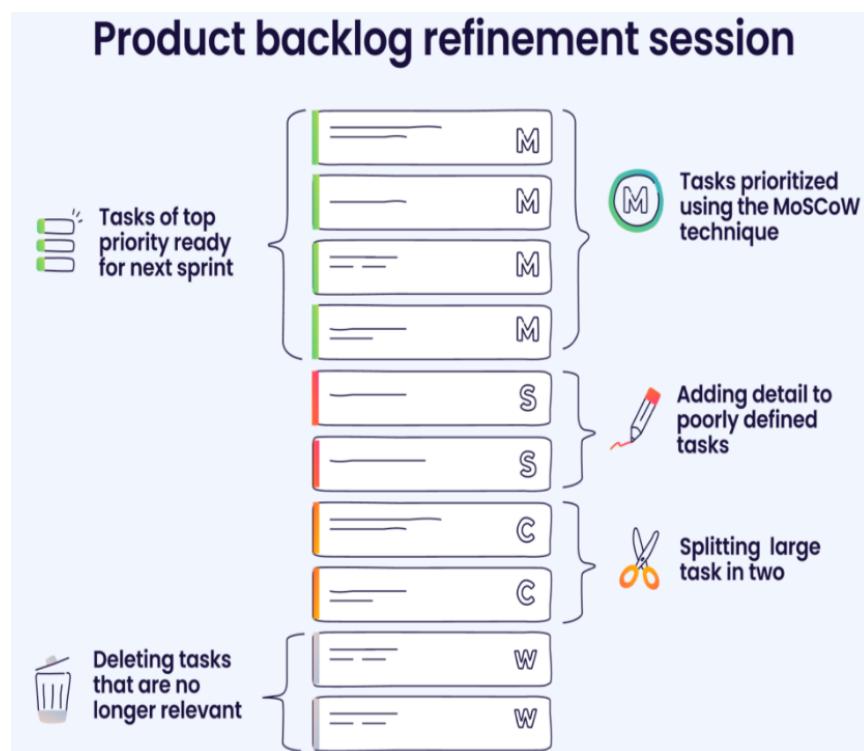
Continuer à faire

Juste une façon
parmi d'autres de
faire une
rérospective.

▼ Product Backlog Refinement

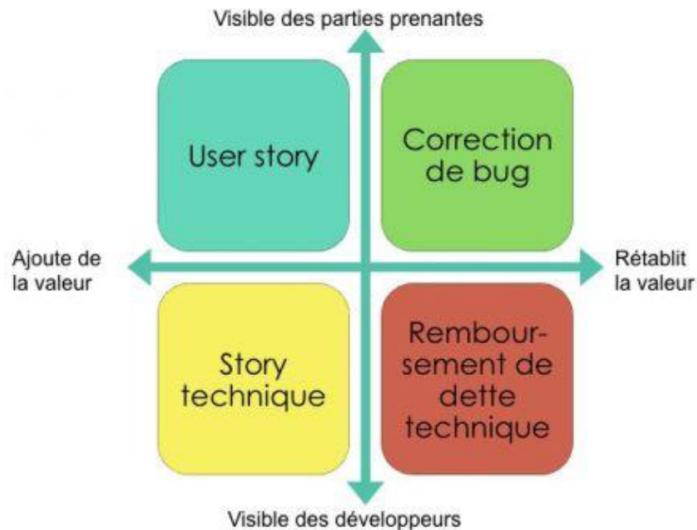
Le **raffinement du Backlog Produit** consiste à ajouter des détails, des estimations et à réorganiser les éléments du Backlog Produit.

- C'est une activité régulière où le **Product Owner** et l'équipe de développement collaborent pour affiner les éléments du Backlog.
- Pendant le raffinement, les éléments sont **revisités** et **révisés** à travers plusieurs actions :
 - Clarification des éléments
 - Priorisation des éléments
 - Estimation des efforts nécessaires
 - Division des éléments trop larges
 - Mise à jour des détails techniques
 - Révision des éléments obsolètes ou non pertinents



▼ Product Backlog

▼ De quoi est constitués le product backlog ?



Le **Product Backlog** est constitué de **tous les éléments nécessaires** à la création du produit. Il contient des fonctionnalités, des exigences, des améliorations et des corrections qui doivent être réalisées pour le produit. Voici les principaux composants d'un **Product Backlog** :

1. User Stories

- Descriptions des fonctionnalités ou besoins exprimés du point de vue de l'utilisateur final.
- Exemple : "En tant qu'utilisateur, je veux pouvoir réinitialiser mon mot de passe afin de sécuriser mon compte."

2. Éléments techniques

- Tâches ou améliorations techniques nécessaires pour maintenir ou faire évoluer le produit (ex : mise à jour de la base de données, refactoring du code, etc.).

3. Bugs / Corrections

- Problèmes identifiés qui doivent être résolus pour améliorer la qualité du produit.

4. Améliorations

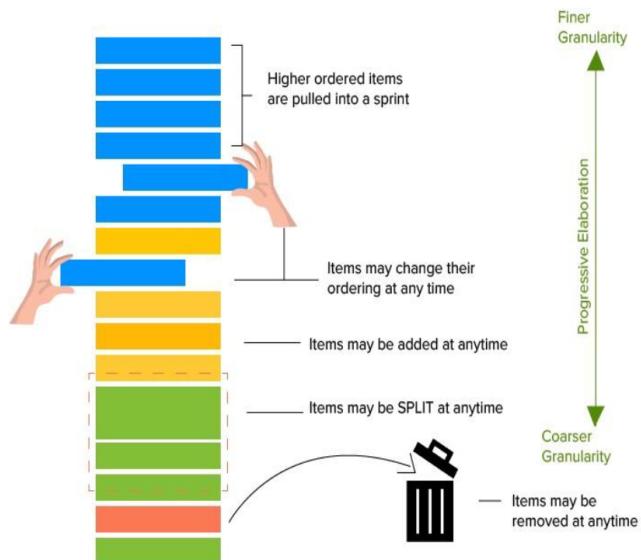
- Suggestions d'améliorations du produit existant, en termes de performance, d'ergonomie, de sécurité, etc.

▼ Comment constituer le product backlog

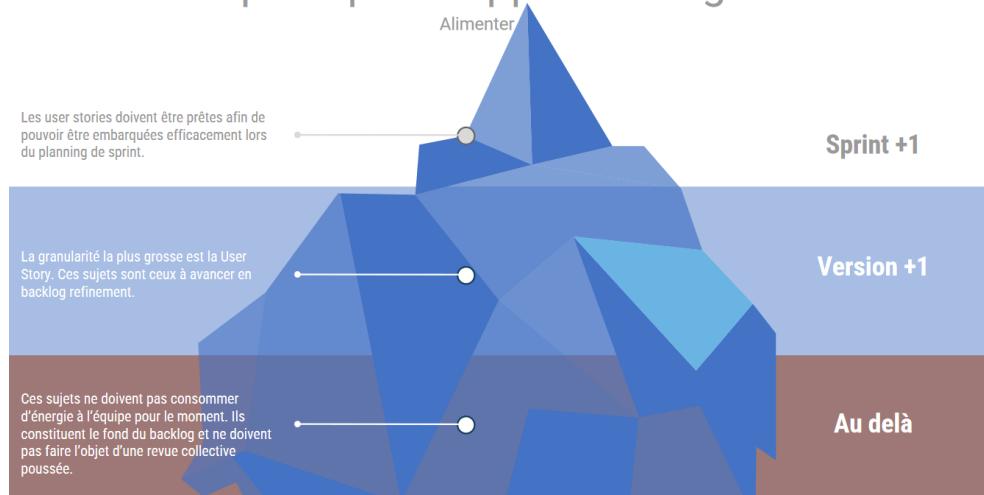
La création du **Product Backlog** implique une approche structurée pour identifier, prioriser et organiser les éléments nécessaires au produit. Le **Product Owner (PO)** est responsable de :

- **Collecter les besoins** des utilisateurs et parties prenantes.
- **Créer et organiser** les éléments dans le backlog.
- **Prioriser** les items en fonction de leur valeur.
- **Estimer l'effort** nécessaire pour chaque élément.
- **Réévaluer** régulièrement le backlog pour ajuster les priorités.
- **Impliquer les parties prenantes** pour valider les priorités et les choix.

Le backlog est un document **évolutif** qui reflète les changements et les priorités du produit au fur et à mesure du projet.



Les principales approches agiles



Sprint +1 : Les user stories doivent être prêtes afin de pouvoir être embarquées efficacement lors du planning de sprint.

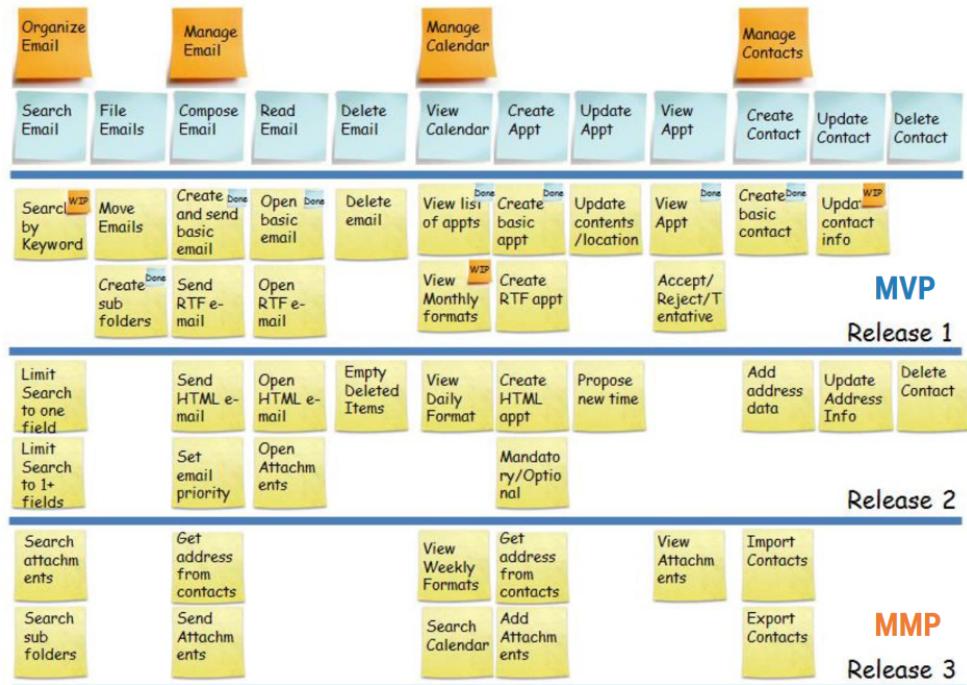
Version +1 : La granularité la plus grosse est la User Story. Ces sujets sont ceux à avancer en backlog refinement.

Au delà : Ces sujets ne doivent pas consommer d'énergie à l'équipe pour le moment. Ils constituent le fond du backlog et ne doivent pas faire l'objet d'une revue collective poussée.

▼ Prioriser : MVP / MMP

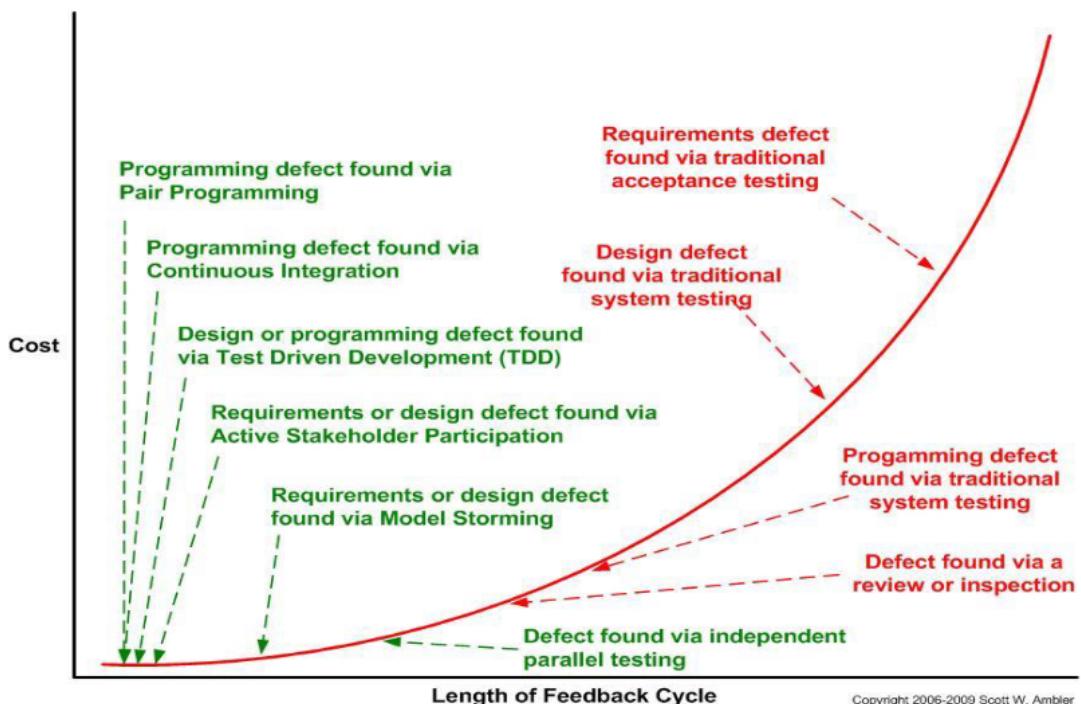
- **Viable Product (MVP)** : C'est l'ensemble minimum de fonctionnalités nécessaires pour fournir une valeur ajoutée et un service de base. Il est destiné à un public restreint, souvent des utilisateurs pilotes ou volontaires, pour tester l'idée du produit.
- **Minimum Marketable Product (MMP)** : C'est le minimum de fonctionnalités qui permet de lancer un produit sur le marché, de le rendre vendable et attrayant pour n'importe quel utilisateur potentiel.

En résumé, le **MVP** sert à valider une idée auprès des premiers utilisateurs, tandis que le **MMP** permet de mettre le produit sur le marché à grande échelle.



▼ La Qualité

▼ Différence des tests en agile et en cascade



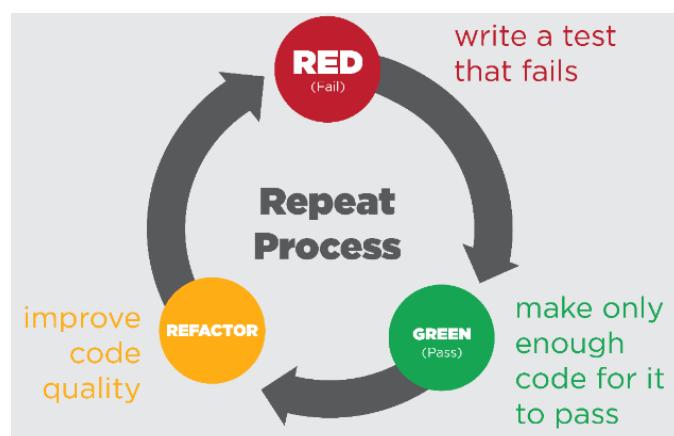
▼ TDD : Tests driven development

Le **Test-Driven Development (TDD)** est une approche où l'on écrit d'abord les tests, puis l'implémentation du code suit.

Les avantages de cette méthode sont :

- **Fonctionnalité opérationnelle** : On s'assure que la fonctionnalité fonctionne dès le départ.
- **Refactorisation sécurisée** : On peut améliorer ou refactorer le code sans risque, car les tests valident chaque changement.
- **Feedback rapide** : Les effets de bord et les erreurs sont détectés tôt, ce qui permet de corriger rapidement.

En résumé, le TDD garantit un code de qualité, stable et réactif aux changements.



The screenshot shows the progression of the TDD cycle:

- Step 1: Write a test that fails** (Red screen):
 - GreetingTest.java:

```
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;

public class GreetingTest {
    @Test
    public void test() {
        Greeting greeting = new Greeting();
        assertEquals(greeting.getMessage(), "Hello world!");
    }
}
```
 - Output: 1/1 Errors: 0 Failures: 1
 - Failure Trace:

```
java.lang.AssertionError: Expected: is "Hello world!"  
but: was null  
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)  
at GreetingTest.test(GreetingTest.java:12)
```
- Step 2: Make only enough code for it to pass** (Green screen):
 - Greeting.java:

```
public class Greeting {
    public Object getMessage() {
        // TODO Auto-generated method stub
        return "Hello world!";
    }
}
```
 - Output: 1/1 Errors: 0 Failures: 0
 - Failure Trace: None
- Step 3: Improve code quality** (Refactor):
 - GreetingTest.java:

```
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;

public class GreetingTest {
    @Test
    public void test() {
        Greeting greeting = new Greeting();
        assertEquals(greeting.getMessage(), "Hello world!");
    }
}
```
 - Greeting.java:

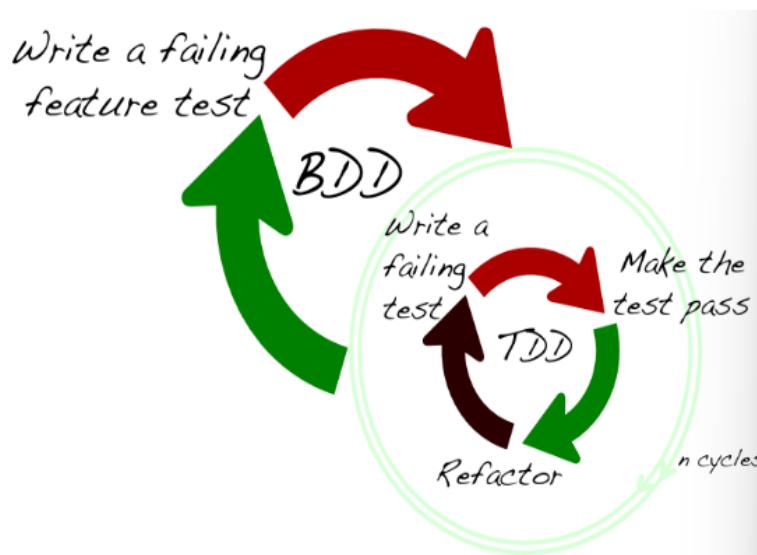
```
public class Greeting {
    public Object getMessage() {
        // TODO Auto-generated method stub
        return "Hello world!";
    }
}
```
 - Output: 1/1 Errors: 0 Failures: 0
 - Failure Trace: None

▼ BDD : Behavior Driven Development

Le **Behavior-Driven Development (BDD)** est une méthodologie qui favorise la collaboration entre **développeurs, testeurs et représentants métier** (comme les Product Owners).

- **Objectif** : L'objectif est d'écrire des **scénarios** qui décrivent le comportement attendu d'une fonctionnalité, vu du point de vue de l'utilisateur final.
- **Langage commun** : Ces scénarios sont rédigés dans un **langage compréhensible** par toutes les parties prenantes (souvent en **Gherkin**, un langage proche du langage naturel).
- **Tests basés sur les scénarios** : Les scénarios servent de base pour la création des tests.

En résumé, le BDD vise à garantir que le développement répond aux attentes des utilisateurs et à faciliter la communication entre toutes les parties impliquées.



Avantages du BDD

- **Meilleure collaboration** entre les équipes techniques et métier.
- **Exigences claires** et compréhensibles pour tous.
- **Automatisation des tests** facilitée, avec une réutilisation du code.
- **Alignement sur les besoins métier** pour un produit plus adapté.
- **Détection rapide des bugs** pour corriger les problèmes tôt.
- **Documentation vivante** qui évolue avec le projet.

BDD exemple

```

Un exemple de scénario BDD serait rédigé ainsi :
•Étant donné une certaine situation,
•Quand une action spécifique est effectuée,
•Alors un résultat attendu doit se produire.

Feature: Get greeting
As a consumer of the greetings resource
I should be able to get a greeting

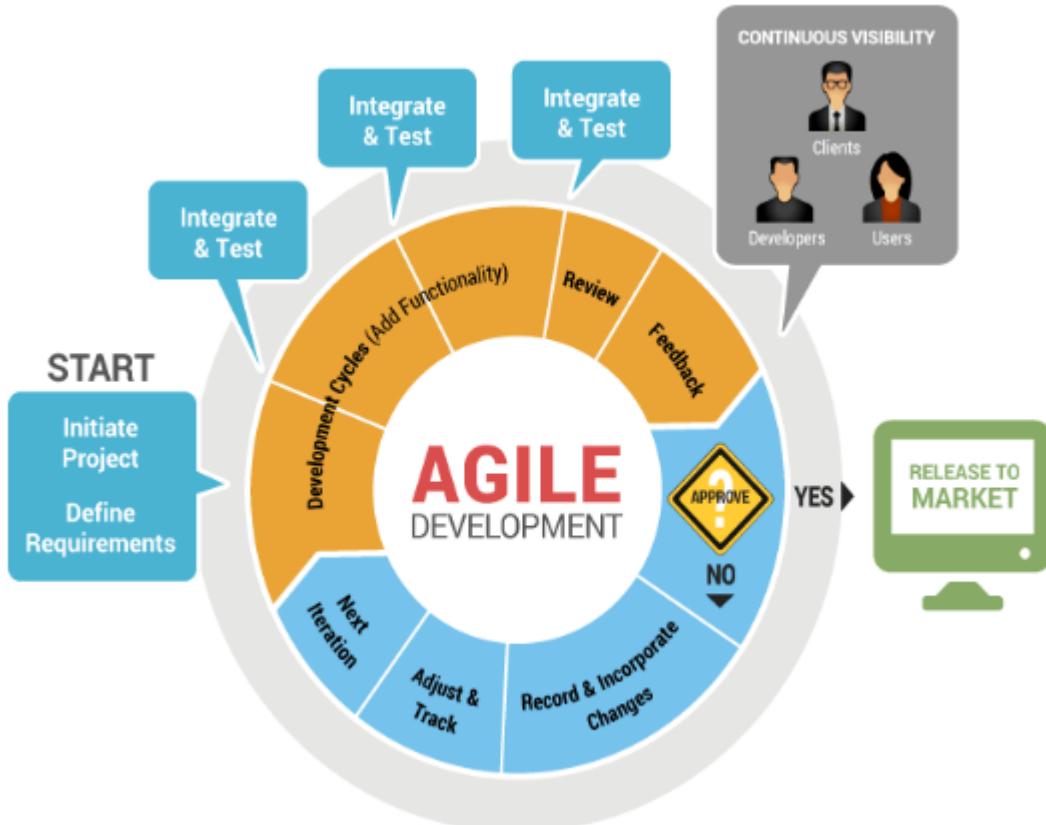
Scenario Outline: Get greeting using appropriate caller
Given I use the caller <caller>
When I request a greeting
Then I should get a response with HTTP status code <status>
And The response should contain the message <message>
Examples:
| caller | status | message |
| Duke   | 200   | Hello World, Duke |
| Tux    | 200   | Hello World, Tux |

Scenario: Get greeting using caller 0xCAFEBADE
Given I use the caller 0xCAFEBADE
When I request a greeting
Then I should get a response with HTTP status code 418

```

▼ L'importance des tests

- **Vérification de la solution** : Les tests garantissent que la solution répond aux attentes (performance, format de sortie, etc.).
- **Feedback sur les réalisations** : Ils fournissent des retours sur le travail accompli et permettent de valider les fonctionnalités.
- **Assurance qualité** : La stratégie de tests assure la qualité du produit tout au long du développement.

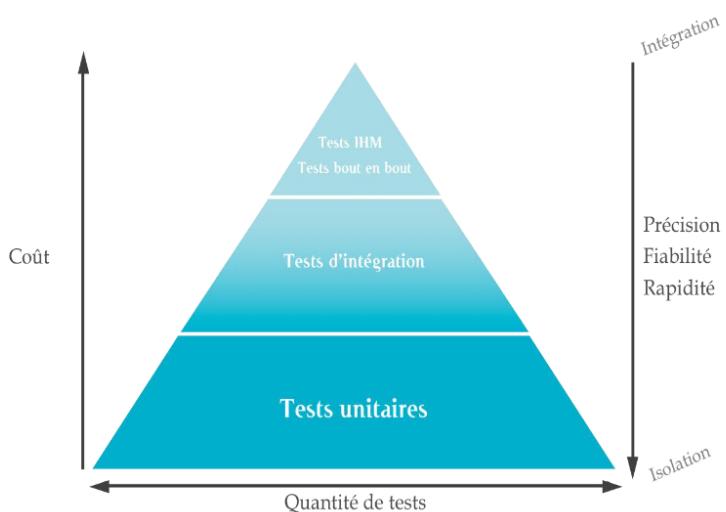


▼ Types de tests

Il existe différents types de tests, manuels ou automatisés :

- **Tests unitaires** : Vérifient le bon fonctionnement des plus petites unités de code (fonctions, méthodes).

- **Tests d'intégration** : S'assurent que plusieurs composants du système fonctionnent bien ensemble.
- **Tests E2E (End-to-End)** : Valident le comportement du système dans son ensemble, du début à la fin.
- **Tests d'acceptation** : Vérifient que le produit répond aux critères d'acceptation du client ou des utilisateurs.



▼ Couts des tests

Type de test	Feedback			Coût (création + maintenance)
	Précision	Fiabilité	Rapidité	
Test Unitaire	Très fine (au niveau fonction)	Très fiable (répétable à l'infini)	Très rapide (quelques secondes)	
Test d'Intégration Test Fonctionnel	Moyenne (partie de logiciel intégré)	Fiable (les dépendances peuvent provoquer des échecs)	Assez rapide (quelques minutes)	
Test de bout en bout Test d'IHM	Faible (logiciel pleinement intégré)	Peu fiable (relativement instable)	Lent (plusieurs dizaines de minutes)	

▼ Quadrants des tests

The Agile Testing Quadrants

