

# Cursus SALESFORCE

M2I - Formation - 2024

---

Donjon Audrey



# Les Méthodes agiles

---



Donjon Audrey  
Formatrice Frontend

---

Contact :



[www.linkedin.com/in/audrey-djn](https://www.linkedin.com/in/audrey-djn)



[audrey\\_](#)



[audrey-donjon](#)

1. La gestion de projet
2. Introduction aux méthodes agiles
3. Les différentes méthodologies Agile
4. Introduction à SCRUM
5. Mise en pratique avec Scrum
6. Utilisation de Jira
7. Outils du développeur agile

# **La gestion de projet**

# La gestion de projet

1. La gestion de projet
2. Importance de la gestion de projet
3. Méthode traditionnelle de gestion de projet
4. Exemple d'application de méthode
5. Les limites de la méthode linéaire

# La gestion de projet

La gestion de projet est l'application de **connaissance**, **compétences**, **outils** et **techniques** pour répondre aux exigences d'un projet. Le projet peut prendre plusieurs formes selon l'objectif visé, il est conçue pour produire un résultat unique que ce soit un produit, un service ou un résultat spécifique.

La gestion de projet va donc impliquer de la **planification**, de **l'organisation**, de la **coordination** et de **contrôler** toutes les activités nécessaires pour atteindre les objectifs du projet dans les délais impartis et le budget prévu.

# Importance de la gestion de projet

Dans le monde des affaires et de la tech, la gestion de projet est cruciale pour assurer le succès des objectifs. Elle permet de :

- ❑ **Définir clairement les objectifs :** Identifier ce qui doit être accompli et les critères de succès
- ❑ **Gérer les ressources :** Utiliser efficacement les ressources humaines, financières et matérielles
- ❑ **Respecter les délais et le budget :** Planifier et contrôler les coûts et les calendriers
- ❑ **Communiquer efficacement :** Faciliter la communication entre les membres de l'équipe et les parties prenantes
- ❑ **Gérer les risques :** Identifier, analyser et atténuer les risques potentiels

# Méthode traditionnelle de gestion de projet

L'une des méthodes connues existant bien avant l'arrivée des méthodes agiles est la **méthode en cascade**. La méthodologie **Waterfall** « **en cascade** » est considérée comme l'approche de référence à la gestion de projet.

Cette méthode est de **nature linéaire**, elle décompose un projet en **une série de phase** où **chaque phase doit être complétée** avant de **passer à la suivante**. **Les principales étapes** sont :

- ❑ **Initiation** : cartographie et analyse des exigences et besoins liés à l'application, au produit ou au projet à livrer, consolidation des résultats dans un cahier des charges qui devra servir de document de référence au cours de la réalisation du projet.
- ❑ **Planification** : développer un plan détaillé pour atteindre les objectifs du projet et définir les ressources nécessaires.
- ❑ **Exécution** : mettre en œuvre le plan et réaliser le travail nécessaire pour atteindre les objectifs du projet.
- ❑ **Suivi et contrôle** : implication de bêta-testeurs et de professionnels du contrôle qualité pour détecter, signaler et corriger d'éventuels problèmes ou bugs.
- ❑ **Clôture** : déploiement et mise en service du livrable achevé, prestations d'assistance et de maintenance.



# Exemple d'un projet web avec la méthode Waterfall

Projet de développement d'un site web pour une petite entreprise qui souhaite vendre ses produits en ligne :

## 1. Initiation : définir le but du projet / Identifier les parties prenantes / Obtenir l'approbation initiale.

- Rédaction d'un document expliquant les bénéfices attendus, les coûts et le retour sur investissement (ROI).
- Identification des fonctionnalités clés du site web, comme le catalogue de produits, le panier d'achat, et le système de paiement.
- Document formel décrivant les objectifs, les parties prenantes, les responsabilités, les délais, et le budget.
- Présentation du business case et de la charte de projet aux sponsors et parties prenantes pour approbation.

## 2. Planification : développer un plan détaillé pour atteindre les objectifs du projet / Définir les ressources nécessaires.

- Réalisation de réunions avec les parties prenantes pour recueillir et documenter les exigences détaillées.
- Création d'un document de spécifications détaillant les fonctionnalités du site web.
- Définition de l'architecture du site web, incluant les bases de données, les serveurs, et les technologies utilisées.
- Développement d'un plan de projet incluant le calendrier, le budget, les ressources, et les jalons.
- Identification des risques potentiels et planification des réponses appropriées.

## 3. Exécution : mettre en œuvre le plan et réaliser le travail nécessaire pour atteindre les objectifs du projet.

- Création des pages web, design UI/UX, et intégration des éléments graphiques.
- Développement des fonctionnalités côté serveur, base de données, et API.
- Intégration des systèmes tiers comme les passerelles de paiement, les services de livraison, etc.
- Test des composants individuels du site web pour s'assurer qu'ils fonctionnent correctement.
- Réunions régulières pour suivre l'avancement du projet et résoudre les problèmes.

# Exemple d'un projet web avec la méthode Waterfall

4. **Suivi et contrôle** : implication de bêta-testeurs et de professionnels du contrôle qualité pour détecter, signaler et corriger d'éventuels problèmes ou bugs.
  - Comparer l'avancement réel aux prévisions et ajuster le plan de projet si nécessaire.
  - Traiter les demandes de changement et évaluer leur impact sur le projet.
  - Générer des rapports réguliers pour informer les parties prenantes de l'avancement du projet.
  - S'assurer que le travail accompli répond aux normes de qualité définies.
  
5. **Clôture** : déploiement et mise en service du livrable achevé, prestations d'assistance et de maintenance.
  - Impliquer les utilisateurs finaux pour tester le site web et valider qu'il répond à leurs besoins.
  - Mettre le site web en ligne et le rendre accessible aux utilisateurs.
  - Former les utilisateurs à l'utilisation du nouveau site web.
  - Fournir la documentation complète du projet, incluant le manuel utilisateur, les guides de maintenance, etc.
  - Réaliser une rétrospective pour évaluer les succès et les leçons apprises.
  - Archiver les documents du projet et libérer les ressources.

# Les limites de la méthode linéaire

Cette approche est bien adaptée aux projets où les **exigences** sont **claires** et **peu susceptibles de changer**.

Par contre elle peut être **moins efficace** dans des **environnements dynamiques** où les besoins peuvent **évoluer au fil du temps**.

Dans les **limites** que peut avoir cette méthode nous avons :

- **Sa rigidité** : il est difficile d'apporter des modifications une fois qu'une phase est terminée
- **De longs cycles de développement** : les projets peuvent prendre beaucoup de temps avant de produire des résultats tangibles
- **Une communication limitée** : interaction limitée entre les différentes phases du projet
- **Réactivité réduite** : Moins de flexibilité pour s'adapter aux changements imprévus

# **Introduction aux méthodes agiles**

# Introduction aux méthodes agiles

1. Origines des méthodes agiles
2. Que sont les méthodes agiles
3. Les méthodologies agiles
4. Le manifeste agile
5. Les 12 principes du Manifeste Agile

# Origines des méthodes agiles

L'origine de la **méthode Agile** remonte au **années 1990** pour répondre aux **limitations des approches traditionnelles** de gestion de projets comme la méthode en **cascade**.

Vis-à-vis des approches traditionnelles qui étaient assez **rigides, lentes** et **moins efficaces** conduisant souvent à des **retards** et à une **insatisfaction des clients**, les **méthodes agiles** ont alors émergés.

# Que sont les méthodes agiles

Contrairement aux **méthodes linéaires**, les **méthodes agiles** s'adaptent mieux aux changements et favorise la collaboration pour satisfaire les besoins évolutifs des clients.

## Méthodes linéaires vs agiles

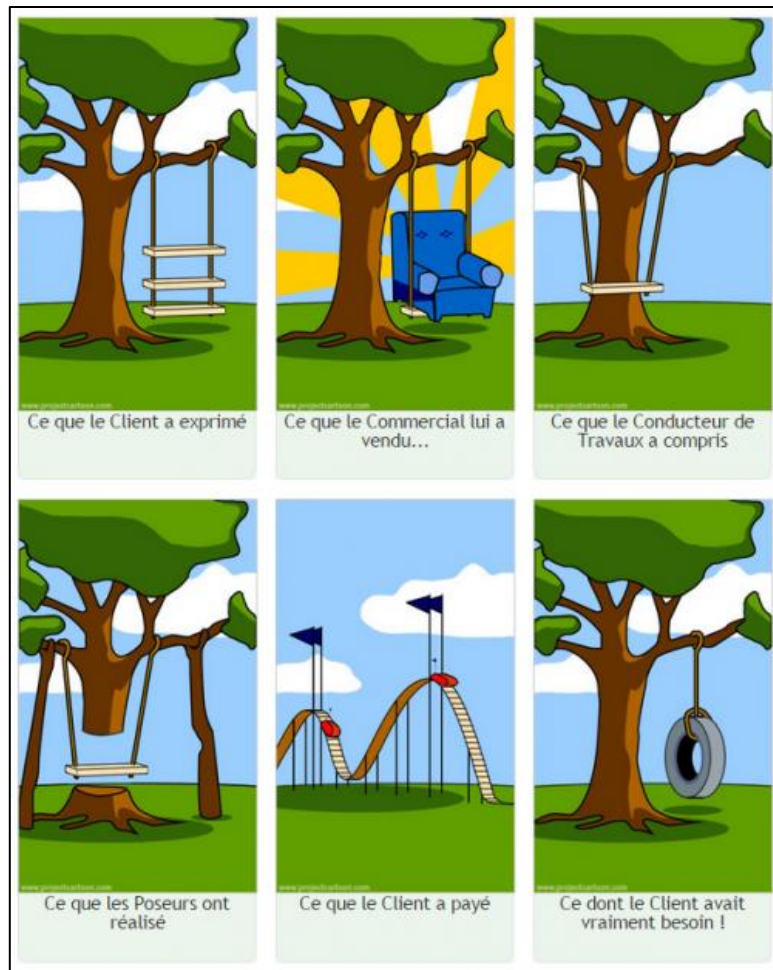
Voici un mini-jeu illustrant l'intérêt de la méthode agile par rapport aux méthodes linéaires :

<https://www.boxuk.com/battleships/>

Le problème des **méthodes linéaires** tel que celui en **cascade**, c'est le **manque de flexibilité** et de **communication**.

Les **méthodes agiles** s'adaptent mieux aux **changements** et **favorise la collaboration** pour satisfaire les besoins évolutifs des clients.

Que veut le client ?



# Les méthodologies de développement Agile

Dans les **années 1990** et au **début des années 2000**, plusieurs méthodologies de développement Agile ont émergé, telles que **Scrum**, **Extreme Programming** (XP) et **Feature-Driven Development** (FDD).

Bien qu'elles aient été développées indépendamment, ces méthodologie partageaient des principes communs, tels que la **flexibilité**, **l'adaptabilité** et la **collaboration** étroite avec le client.

Sont considérés comme étant des méthodes agiles, les approches qui s'inscrivent dans la philosophie du **manifeste pour le développement agile de logiciels**.



# Le manifeste Agile

En **février 2001**, 17 développeurs (experts en dev logiciel) se sont réunis à Snowbird, dans l'Utah, aux États-Unis pour discuter de **nouvelles approches de développement logiciel**. De cette réunion est né le **Manifeste Agile**, un **document fondateur** qui a **formalisé les valeurs** et **principes des méthodes agiles**.

Les **quatre valeurs principales** sur lequel est fondé le Manifeste Agile sont :

- ☐ Les **individus** et les **interactions** plutôt que les processus et les outils
- ☐ Des **logiciels opérationnels** plutôt que de la documentation exhaustive
- ☐ La **collaboration avec les clients** plutôt que la négociation contractuelle
- ☐ L'**adaptation au changement** plutôt que le suivi d'un plan

# Les 12 principes du Manifeste Agile

Le **Manifeste Agile** définit également 12 principes pour guider le développement logiciel Agile.

Ces principes encouragent la **satisfaction** du **client**, la **collaboration**, l'**auto-organisation des équipes**, la **communication** face à face, la **simplicité** et la **réflexion** sur les **méthodes de travail** pour les **améliorer continuellement**.

# Principe 01 : Satisfaction du client

**Principe :** Satisfaire le client par des livraisons rapides et continues de logiciels opérationnels.

Ce principe explique qu'il faut s'assurer que les **clients** soient **satisfait** en leur fournissant des **fonctionnalités utilisables dès que possible**.

# Principe 02 : Accueillir le changement

**Principe** : Accueillir favorablement les changements de besoins, même tard dans le développement.

Ce principe explique qu'il faut être prêt à **adapter le produit** en fonction des nouvelles informations et des besoins changeants.

## Principe 03 : Livraison fréquente

**Principe** : Livrer fréquemment des logiciels opérationnels, avec une préférence pour des cycles courts.

Ce principe explique qu'il faut fournir des **mise à jour régulières** du logiciel pour recevoir rapidement des retours et apporter des améliorations.

## Principe 04 : Collaboration quotidienne

**Principe** : Les utilisateurs et les développeurs doivent travailler ensemble quotidiennement tout au long du projet.

Ce principe explique qu'il faut encourager une **communication constante** entre **l'équipe de développement** et **les utilisateurs** pour mieux comprendre les besoins et ajuster les fonctionnalités.

# Principe 05 : Individus motivés

**Principe** : Construire les projets autour d'individus motivés. Leur fournir l'environnement et le soutien dont ils ont besoin et leur faire confiance pour accomplir le travail.

Ce principe explique qu'il faut **créer un environnement de travail favorable** où les développeurs se sentent valorisés et soutenus..

# Principe 06 : Communication en face à face

**Principe** : La méthode la plus efficace pour transmettre de l'information à une équipe de développement est la conversation en face à face.

Ce principe explique qu'il faut **privilégier les discussions directes** plutôt que les emails ou les documents pour une meilleure compréhension et une communication plus rapide.



# Principe 07 : logiciel opérationnel

**Principe** : Un logiciel opérationnel est la principale mesure de progrès.

Ce principe explique qu'il faut **évaluer l'avancement du projet** en fonction des **fonctionnalités réellement utilisables** plutôt qu'en fonction de documents ou de plans.

## Principe 08 : Développement durable :

**Principe** : Les processus agiles favorisent un développement durable. Les sponsors, les développeurs et les utilisateurs devraient être capables de maintenir un rythme constant indéfiniment.

Ce principe explique qu'il faut adopter un **rythme de travail soutenable** qui évite le surmenage et permet une productivité à long terme.

# Principe 09 : Excellence technique

**Principe** : Une attention continue à l'excellence technique et à une bonne conception renforce l'agilité.

Ce principe explique qu'il faut **maintenir un haut niveau de qualité technique** pour faciliter les modifications et les améliorations du logiciel.

# Principe 10 : Simplicité

**Principe :** La **simplicité**, c'est-à-dire l'art de maximiser l'efficacité en éliminant le travail inutile est essentiel

Ce principe explique qu'il faut se concentrer sur les fonctionnalités vraiment nécessaires et **évitez les complexités inutiles**.

# Principe 11 : Équipes auto-organisées

**Principe** : Les meilleures architectures, exigences et conceptions émergent d'équipes auto-organisées.

Ce principe explique qu'il faut donner à l'équipe la **liberté de s'organiser et de décider** comment accomplir le travail pour tirer parti de leur expertise collective.

# Principe 12 : Réflexion et ajustement

**Principe :** À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et ajuste son comportement en conséquence.

Ce principe explique qu'il faut prendre le temps de **revoir régulièrement les processus et les performances de l'équipe** pour identifier les améliorations possibles et les mettre en œuvre.

# Les 12 principes du Manifeste Agile

En respectant ces **12 principes**, les équipes Agile sont en mesure de **créer un environnement de travail flexible, collaboratif et axé sur la satisfaction des besoins du client.**

Ces principes encouragent les équipes à s'adapter rapidement aux changements, à travailler en étroite collaboration et à améliorer continuellement leurs processus et résultats.

# Exercice 01 : méthodes agiles vs méthodes traditionnelles

Prenez 10 minutes pour lister les différences clés entre les méthodes agiles et les méthodes traditionnelles.





# **Les différentes méthodologies Agiles**

# Principales méthodologies agiles

1. Méthodologie Scrum
2. Méthodologie Kanban
3. Extreme Programming (XP)
4. Lean Software Development
5. Comment choisir la bonne méthode ?

# Méthodologie Scrum

**Scrum** est l'une des méthodologies Agile les plus populaires largement adoptées pour la gestion de projets complexes et le développement logiciel.

Cette méthodologie a été créée en **1995**, elle a été développée par **Jeff Sutherland** et **Ken Schwaber**. Les deux créateurs ont présenté Scrum pour la première fois lors de la **conférence OOPSLA** (Object-Oriented Programming, Systems, Languages & Applications) de 1995.

Le nom **Scrum** provient du **rugby**, où un "**scrum**" (ou "**mêlée**" en français) est une **formation spécifique** utilisée pour redémarrer le jeu après une infraction mineure. Dans ce contexte, les joueurs des **deux équipes se rassemblent et travaillent ensemble de manière coordonnée** pour gagner la possession du ballon.

**Jeff Sutherland et Ken Schwaber**, ont emprunté ce terme pour illustrer l'idée d'une **équipe** de développement **travaillant de manière cohésive** et **collaborative** pour **atteindre un objectif commun**. L'analogie avec le rugby **souligne l'importance** de la **collaboration**, de la **communication** et de l'**auto-organisation**, qui sont des principes fondamentaux de la **méthodologie Scrum**.

# Méthodologie Scrum

Dans la méthodologie **Scrum**, il existe **trois rôles principaux** : le **Product Owner**, le **Scrum Master** et **l'équipe de développement**.

Il y également ce qu'on appelle les **artéfacts** Scrum qui sont le **Product Backlog**, le **Sprint Backlog** et le **Product Increment**.

Les **artéfacts Scrum** sont des **outils** utilisés pour **planifier** et **suivre le travail** dans un projet **Scrum**.

Enfin il existe aussi les **événements Scrum** qui sont les **étapes clés** qui ont lieu tout au long du processus de développement Scrum pour planifier, suivre et améliorer le travail de l'équipe.

Il y a **5 événements dans Scrum** :

- ☐ Le sprint planning
- ☐ Le sprint
- ☐ Le daily scrum
- ☐ La sprint review
- ☐ La sprint retrospect

# Méthodologie Kanban

**Kanban** est une **méthode de gestion du travail agile** visant à améliorer **l'efficacité** et la **flexibilité** des équipes. Elle utilise un **système visuel** pour gérer les tâches, généralement sous la forme de **tableaux** avec des **colonnes** représentant les différentes étapes du **flux de travail** (par exemple, "À faire", "En cours", "Terminé").

**Kanban** (mot japonais qui signifie « **carte visuelle** » ou « **panneau** ») a été développé par **Taiichi Ohno** chez **Toyota** dans les années **1940** et **1950**. **Kanban** a été initialement conçu dans le cadre du **système de production Toyota** pour améliorer **l'efficacité** de la chaîne d'assemblage automobile en utilisant un **système de cartes** pour signaler la nécessité de réapprovisionnement en matériaux.

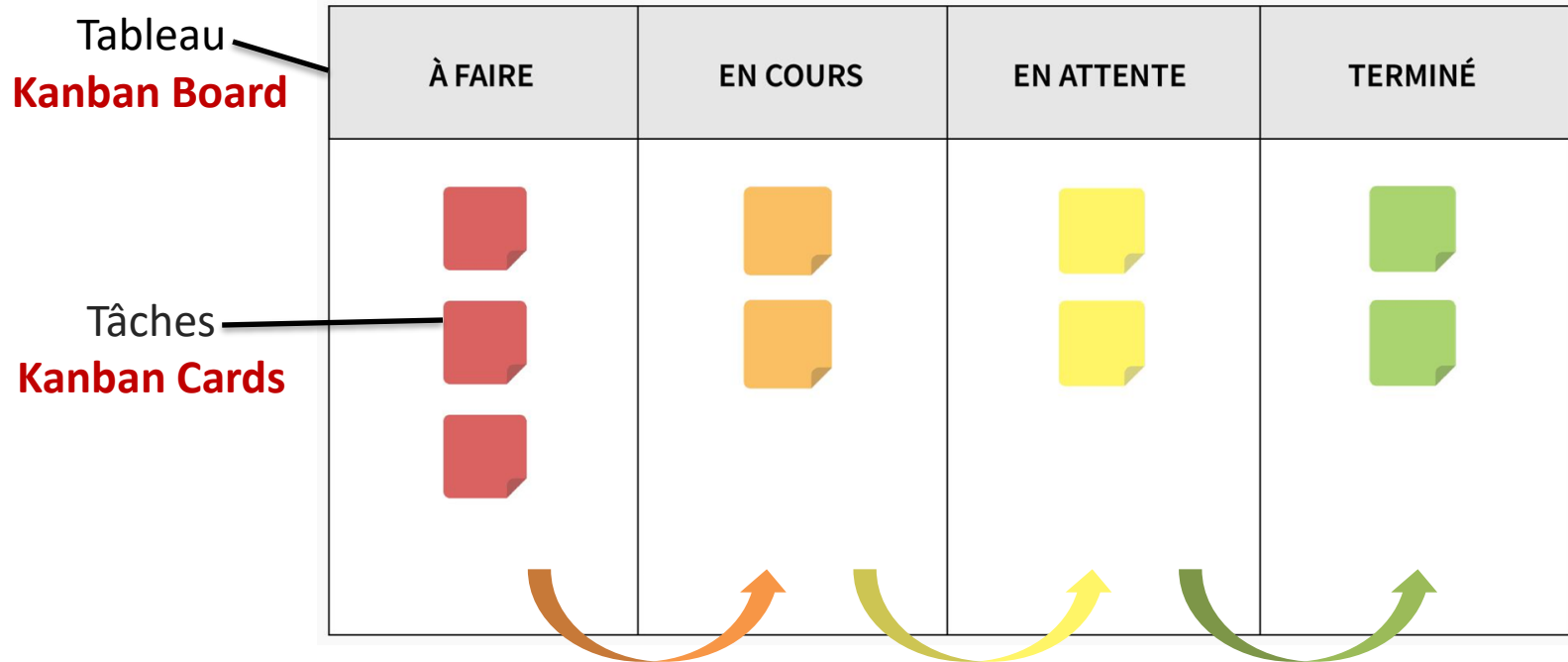
Il a été adapté aux environnements de développement logiciel et de gestion de projet à partir des années 2000.

Kanban est axé sur la **visualisation du flux de travail**, la **limitation du travail en cours** et **l'amélioration continue**.

# Méthodologie Kanban : visualisation du travail

Dans les **principes de base de Kanban**, on utilise un **tableau** pour représenter visuellement le **flux de travail** et les **tâches**.

Quand une tâche avance, on la change de colonne.



# Méthodologie Kanban : limitation du travail

Dans les **principes de base de Kanban**, on fixe des limites au nombre de tâches pouvant être en cours à un moment donné (**WIP**) pour éviter la surcharge.

Le but est d'encourager la **concentration** sur les **tâches en cours** avant d'en commencer de nouvelles. De ce fait nous avons une **réduction des goulots d'étranglement**, une **amélioration de la qualité du travail** et une **accélération du délai de livraison**.

# Méthodologie Kanban : Amélioration continu

Dans les **principes de base de Kanban**, on recherche constamment des moyens d'améliorer les processus et les pratiques de travail par l'amélioration continue.

Kanban encourage les équipes à surveiller et à analyser constamment leur flux de travail pour identifier les opportunités d'amélioration.



# Méthodologie Kanban : les avantages

**Kanban** met l'accent sur la **visualisation du flux de travail**, la **limitation du travail en cours** et **l'amélioration continue**.

Les équipes utilisent un tableau Kanban pour **gérer et suivre les éléments de travail**, en s'assurant que les priorités sont claires et que le travail progresse efficacement à travers le processus de développement.

**Grande flexibilité et adaptabilité**, ce qui permet aux équipes de **répondre rapidement aux changements** et **d'optimiser** constamment **leur performance**.

**Kanban** est particulièrement **adapté aux projets où les priorités peuvent changer fréquemment**.

# Méthodologie Extreme Programming (XP)

**XP** est une approche Agile axée sur la **qualité du code** et **l'efficacité du développement**, qui met l'accent sur des pratiques comme la **programmation en binôme**, **l'intégration continue** et le **test automatisé**.

Extreme Programming (XP) est née officiellement en **octobre 1999** avec la parution du livre « *Extreme Programming Explained* » de **Kent Beck**. Cette méthode a été conçue en réponse aux **défis rencontrés** lors du développement de logiciels, tels que les **changements fréquents** des exigences et la **nécessité d'améliorer la qualité du code**. Il a été initialement mis en pratique dans un projet chez **Chrysler** (C3 project) pour **améliorer l'efficacité du développement logiciel**.

"**Extreme Programming**" vient de l'idée de **pousser les bonnes pratiques** de développement logiciel à l'**extrême**. Par exemple, si le **test de code** est une bonne pratique, XP préconise de **tester en continu**. Si la révision du code est bénéfique, XP pousse cette pratique à l'extrême en utilisant le **pair programming**.

# Méthodologie XP : les cinq valeurs fondamentales

L'**extreme programming** repose sur cinq valeurs fondamentales :

**Communication** : Encourage une communication continue entre les membres de l'équipe et les parties prenantes.

**Simplicité** : Concevoir des solutions simples et éviter les complexités inutiles.

**Feedback** : Obtenir des retours réguliers des clients et des utilisateurs pour s'assurer que le produit répond à leurs besoins.

**Courage** : Prendre des décisions audacieuses pour améliorer le projet, même si cela signifie revoir ou refactorer le code existant.

**Respect** : Valoriser et respecter chaque membre de l'équipe et leurs contributions.

# Méthodologie XP : pratiques clés

Voici quelques pratiques de cette méthode :

**Pair Programming** : Deux développeurs travaillent ensemble sur une même tâche, partageant un seul poste de travail.

**Test-Driven Development (TDD)** : Écrire des tests automatisés avant de coder les fonctionnalités.

**Refactoring** : Améliorer continuellement le code pour le rendre plus clair et plus maintenable.

**Intégration Continue** : Fusionner fréquemment le code développé et exécuter des tests automatisés pour détecter rapidement les problèmes.

**Petites Versions Fréquentes** : Livrer des versions opérationnelles du logiciel fréquemment pour obtenir des retours rapides.

# Méthodologie XP : les avantages

Pour les **avantages** qu'offre cette méthode nous avons :

- ☐ Une communication claire avec les clients.
- ☐ Aucun travail de programmation inutile et un code clair à tout moment.
- ☐ Un logiciel stabilisé grâce à des tests continus.
- ☐ Une prévention des erreurs grâce au pair programming.
- ☐ des modifications qui peuvent être prises en charge à court terme.

# Méthodologie Lean

La méthodologie **Lean**, développée par **Toyota** en sortie de la seconde guerre mondiale, vise à maximiser la valeur pour le client tout en minimisant le gaspillage.

Le terme "**Lean**" (qui signifie "**maigre**" ou "**dépouillé**") a été utilisé pour la première fois par **John Krafcik** en **1988** dans un article intitulé "**Triumph of the Lean Production System**". Il décrit un système de production où tout ce qui ne crée pas de valeur est éliminé, rendant ainsi le processus plus "maigre" et plus efficace.

Au cours des années qui suivront 2000, il y aura plusieurs déclinaisons dont le **lean software development** dans le domaine du **développement logiciel**.

Ce modèle met en place sept principes :

1. **Éliminer les gaspillages** : comme pour le **lean**, le gaspillage est défini comme ce qui n'apporte pas de valeur au produit. La valeur étant définie du point de vue de l'utilisateur.
2. **Améliorer** l'apprentissage en favorisant un environnement qui encourage l'apprentissage et l'amélioration continue.
3. **Retarder** l'engagement en retardant les décisions importantes jusqu'à ce que l'on dispose de toutes les informations nécessaires pour faire des choix plus éclairés.
4. **Livrer** aussi vite que possible en réduisant le temps entre le début et la livraison d'une fonctionnalité.
5. **Donner** le pouvoir à l'équipe en leur fournissant les ressources, les responsabilités et la liberté de prendre des décisions.
6. **Intégrer** la qualité dès la conception comme les revues de code, et les tests automatisés.
7. **Considérer** le produit dans sa globalité et ainsi optimiser l'ensemble du système.

# Méthodologie Lean : avantages

Cette méthodologie de gestion de projet permet :

- ❑ Une réduction des gaspillages
- ❑ Une amélioration continue
- ❑ Des décisions retardée et éclairée
- ❑ Des livraisons rapides
- ❑ Une autonomisation de l'équipe
- ❑ Une intégration de la qualité dès la conception
- ❑ Une vision globale du produit

# Comment choisir la bonne méthode ?

Chaque **méthodologie Agile** présente des **avantages** et des **inconvénients**, et leur efficacité **dépend des besoins spécifiques de chaque projet et équipe**.

**Scrum** : Idéal pour les projets qui nécessitent des **livraisons rapides et régulières**, ainsi qu'une **structure bien définie avec des rôles clairement définis**. Scrum est particulièrement adapté aux équipes qui sont nouvelles dans le développement Agile et qui ont besoin d'une structure pour les guider.

**Kanban** : Convient aux projets où les **priorités peuvent changer fréquemment** et où la **livraison continue des fonctionnalités** ou des **produits est importante**. Kanban est également utile pour les équipes qui cherchent à **améliorer leur flux de travail** et à **réduire les goulots d'étranglement**.



# Comment choisir la bonne méthode ?

**Extreme Programming (XP)** : Idéal pour les projets où la **qualité du code** et la **capacité à s'adapter rapidement** aux changements sont essentielles. XP convient particulièrement aux équipes de développement qui ont déjà une certaine expérience avec les méthodologies Agile et qui cherchent à **améliorer leurs pratiques de développement et leur collaboration**.

**Lean** : Convient aux projets qui mettent l'accent sur la **maximisation de la valeur pour le client** et la **minimisation du gaspillage** dans les processus de développement et de gestion de projet.

# Comment choisir la bonne méthode ?

En définitive, une **méthode Agile est rarement utilisée complètement seule** :

Il est tout à fait possible **de combiner plusieurs méthodes agiles** pour adapter les pratiques à l'équipe et au projet. En fait, de nombreuses équipes combinent des éléments de différentes méthodes, **en fonction de leurs besoins spécifiques** et de leurs **objectifs**.

L'essentiel est de **rester fidèle aux principes agiles** de **collaboration**, de **flexibilité**, **d'adaptabilité** et **d'amélioration continue** tout en adaptant les méthodes aux besoins.

Par exemple, certaines équipes peuvent adopter le cadre Scrum avec ses rôles et événements, tout en intégrant les pratiques de visualisation du flux de travail et de limitation du travail en cours de Kanban.

# Exercice 02 : Choix de la méthodologie agile

Vous êtes chargé de **conseiller une entreprise** sur la méthodologie agile la plus adaptée pour différents projets de développement de sites web et de logiciels.

Pour chaque projet, **identifiez la méthode agile la plus appropriée** (Scrum, Kanban, Extreme Programming, Lean Software Development) et **justifiez votre choix**. Vous pouvez également **combiner plusieurs méthodologies** si cela est pertinent et justifié.



# Exercice 02 : projet 01

**Projet :** Maintenance et Mise à Jour d'un Logiciel de Gestion des Ressources Humaines (GRH).

**Objectif Principal :** Améliorer la qualité et la performance d'un logiciel existant en déployant régulièrement des correctifs et des mises à jour.



# Exercice 02 : projet 02

**Projet :** Optimisation des Processus de Développement d'un Site Web d'Actualités.

**Objectif Principal :** Identifier et éliminer les gaspillages dans le processus de développement pour réduire les coûts et améliorer l'efficacité globale.



# Exercice 02 : projet 03

**Projet :** Développement d'un Nouveau Site E-commerce.

**Objectif Principal :** Lancer rapidement une plateforme e-commerce avec des fonctionnalités de base et ajouter des fonctionnalités avancées au fil du temps en fonction des retours des utilisateurs.



# Exercice 02 : projet 04

**Projet :** Création d'un Logiciel de Gestion de Projet pour une Entreprise.

**Objectif Principal :** Développer un logiciel de gestion de projet robuste et scalable avec une forte collaboration entre les équipes de développement et les utilisateurs finaux.



# **Introduction à SCRUM**



# Introduction à SCRUM

1. Rôle dans Scrum
2. Artefacts Scrum
3. Évènements Scrum

# Rôles clés dans Scrum

Dans la méthodologie **Scrum**, il existe **trois rôles principaux** : le **Product Owner**, le **Scrum Master** et **l'équipe de développement**.

Chaque rôle a des responsabilités spécifiques et travaille en étroite collaboration avec les autres membres de l'équipe pour atteindre les objectifs du projet.

- ❖ **Product Owner** : Définit et priorise les besoins du client en faisant le lien entre ce dernier et l'équipe.
- ❖ **Scrum Master** : Facilitateur et coach. Le Scrum Master veille au respect des principes et pratiques de Scrum, cherche à améliorer l'efficacité de l'équipe.
- ❖ **Équipe de développement** : Composée de professionnels, ils conçoivent, construisent, testent et livrent les fonctionnalités.

# Rôles clés dans Scrum : Le Product Owner

Le **Product Owner** est responsable de **définir** et **prioriser** les exigences du projet, en s'assurant que l'équipe de développement travaille sur les fonctionnalités et les tâches qui offrent **la plus grande valeur ajoutée pour le client**.

Les **principales responsabilités** du **Product Owner** comprennent :

- ❖ **Gérer** et **prioriser** le **Product Backlog** selon la valeur ajoutée et les contraintes.
- ❖ **Collaborer** avec **clients** et **parties prenantes** pour définir les exigences.
- ❖ **Communiquer** clairement **les attentes** et **objectifs** à l'équipe de développement.
- ❖ **Prendre des décisions** sur les **priorités** en tenant compte des contraintes de temps et de ressources.
- ❖ **Participer aux revues de sprint** et **rétrospectives** pour **évaluer la progression** et les **améliorations**

# Rôles clés dans Scrum : Le Scrum Master

Le **Scrum Master** est responsable de faciliter et de guider l'équipe de développement dans l'adoption des pratiques Scrum et de veiller à ce que le processus Scrum soit suivi de manière efficace et cohérente.

Les **principales responsabilités** du Scrum Master comprennent :

- ❖ **Coacher** et **guider** tout le monde sur la **méthodologie Scrum** et les **bonnes pratiques Agile**.
- ❖ **Résoudre** rapidement les **obstacles** et **problèmes** entravant la progression de l'équipe.
- ❖ **Animer** les **réunions Scrum**.
- ❖ **Aider l'équipe** à s'**auto-organiser** et prendre des **décisions collectives** sur le travail et les priorités.
- ❖ **Protéger l'équipe** des **interférences extérieures** et **distractions**.

# Rôles clés dans Scrum : L'équipe de développement

L'**équipe de développement** est composée des professionnels qui sont directement impliqués dans la création et la livraison du produit, y compris les développeurs, les testeurs, les concepteurs et les ingénieurs.

Les **principales responsabilités** de l'équipe de développement comprennent :

- ❖ **Concevoir, développer, tester et livrer** les fonctionnalités et les tâches du Product Backlog dans le respect des objectifs et des délais du sprint.
- ❖ **Travailler** en étroite collaboration avec le **Product Owner** pour comprendre les exigences et aligner le travail sur les objectifs du projet.
- ❖ **S'auto-organiser** et **prendre des décisions collectives** concernant les approches techniques, les priorités et la répartition des tâches.
- ❖ **Participer aux Daily Scrum**, aux **Sprint Review** et aux **rétrospectives** pour communiquer sur l'avancement du travail, **identifier les obstacles** et **discuter des améliorations** possibles.

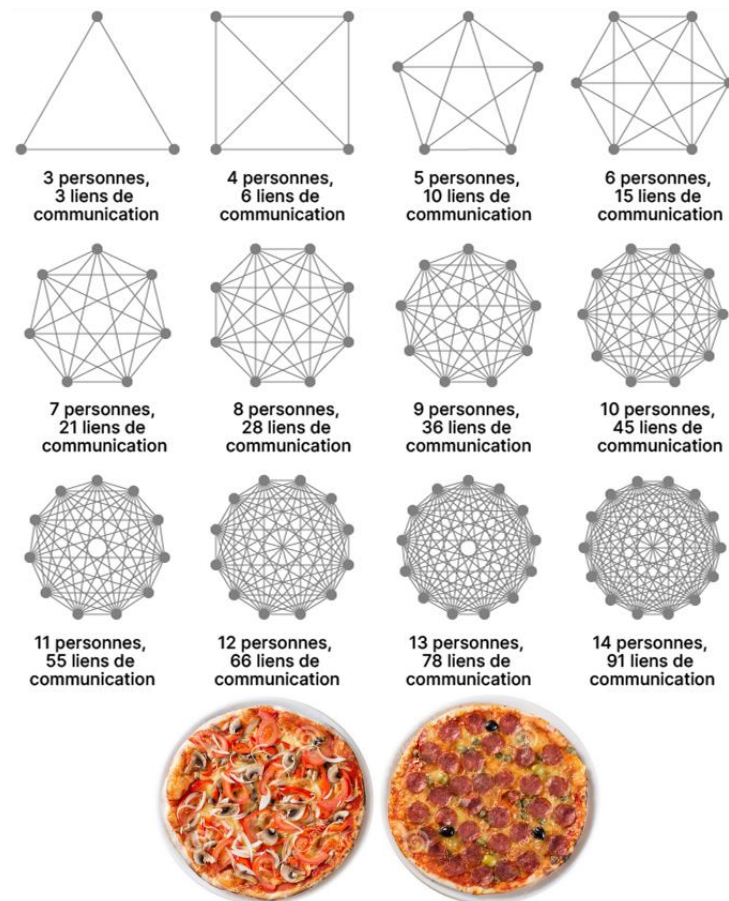
# Taille de l'équipe

Une **équipe Scrum** est généralement composée au maximum d'une **dizaine de personne**.

Il est important de maintenir une **taille d'équipe réduite** afin d'assurer une **communication efficace** et de **faciliter la collaboration** entre les membres.

**Exemple intéressant : Jeff Bezos**, fondateur et ex-PDG d'Amazon, appliquait dans son entreprise la "**règle des 2 pizzas**" pour souligner l'importance de limiter la taille des équipes et des groupes de travail. Selon cette règle, deux pizzas devraient suffire à nourrir l'ensemble de l'équipe lors d'une réunion ou d'un atelier.

En appliquant cette approche, les performances et l'efficacité de l'équipe sont optimisées au profit de l'entreprise.



# Artefacts Scrum

Les **artéfacts Scrum** sont des **outils** utilisés pour **planifier** et **suivre le travail** dans un projet **Scrum**.

Ils aident à **organiser** et à **visualiser** les **tâches**, les **priorités** et les **objectifs** du projet, et sont utilisés par l'équipe Scrum pour **communiquer** et **gérer** les **informations** tout au long du processus de développement.

Les **trois principaux artéfacts** Scrum sont le **Product Backlog**, le **Sprint Backlog** et le **Product Increment**.

# Artefacts Scrum : Le product Backlog

Le **Product Backlog** est une **liste ordonnée** de toutes les fonctionnalités, exigences, améliorations et corrections nécessaires pour le projet.

Il est **géré par le Product Owner**, qui est responsable de prioriser les éléments du **Product Backlog** en fonction de la valeur ajoutée pour le client, des contraintes du projet et des besoins des parties prenantes.

Le **Product Backlog** est un **document vivant qui évolue** et change tout au long du projet, à mesure que de nouvelles exigences sont découvertes et que les priorités changent.

Chaque **tâche** dans le document est appelée « **User Story** » et possède une priorité (définie par le **Product Owner**), ainsi qu'un **niveau de difficulté** (défini par l'équipe de développement).

User story	Story point(s)	Priority
As a user, I am able to search for documents so I can find them more easily	2	1
As a site visitor, I can compare different types of accounts to see which account type suites me best	1	2
As a user, I can submit questions through the website so I know how to better use the product	1	3
As a site visitor, I am shown what I can do in the product so I know whether or not this product will fill my needs	2	4
As a user, I want to be able to retrieve documents that were deleted so I can reclaim documents that were deleted on accident	3	5



# Le product Backlog : Les User Stories

Les **User Stories**, écrites par le **Product Owner**, sont des descriptions simples des fonctionnalités du produit, écrites du point de vue d'un utilisateur (En tant que [rôle], je veux [action] afin de [bénéfice]).

Elles aident à **définir les attentes** et **les objectifs**, et sont **estimées** en termes de difficulté **par l'équipe** pour faciliter la planification des sprints.

## Exemples de User Stories :

En tant que **client**, je veux **pouvoir rechercher des produits par catégorie** pour **faciliter mes achats**.

En tant qu'**administrateur**, je veux **pouvoir gérer les comptes utilisateurs** pour **assurer la sécurité du système**.

En tant qu'**utilisateur**, je veux **pouvoir réinitialiser mon mot de passe en cas d'oubli** pour **accéder à mon compte**.

En générale, la difficulté d'une **User Story** est définie soit avec un nombre de la **suite de Fibonacci** (1, 2, 3, 5, 8, 13, 21, 34, ...), soit avec une **notation de taille** (XS, S, M, L, XL, XXL).

User story	Story point(s)
As a user, I am able to search for documents so I can find them more easily	2
As a site visitor, I can compare different types of accounts to see which account type suites me best	1
As a user, I can submit questions through the website so I know how to better use the product	1
As a site visitor, I am shown what I can do in the product so I know whether or not this product will fill my needs	2
As a user, I want to be able to retrieve documents that were deleted so I can reclaim documents that were deleted on accident	3

# Artefacts Scrum : Le Sprint Backlog

Le **Sprint Backlog** est la version « Sprint » du **Product Backlog**. Ce dernier ne contient que les **User Stories** sélectionnées pour être **développées lors du prochain sprint**.

L'équipe de développement choisit les éléments du **Product Backlog** en fonction des priorités définies par le **Product Owner** et de la capacité de l'équipe à réaliser le travail dans le temps imparti pour le sprint.

Le **Sprint Backlog** est utilisé pour planifier et suivre le travail au cours du sprint, et pour s'assurer que l'équipe de développement se concentre sur les tâches prioritaires.

Backlog produit

User story	Story point(s)	Priority
As a user, I am able to search for documents so I can find them more easily	2	1
As a site visitor, I can compare different types of accounts to see which account type suites me best	1	2
As a user, I can submit questions through the website so I know how to better use the product	1	3
As a site visitor, I am shown what I can do in the product so I know whether or not this product will fill my needs	2	4
As a user, I want to be able to retrieve documents that were deleted so I can reclaim documents that were deleted on accident	3	5

Sprint Backlog

User story	Priority
As a user, I am able to search for documents so I can find them more easily	1
As a site visitor, I can compare different types of accounts to see which account type suites me best	2
As a user, I can submit questions through the website so I know how to better use the product	3

# Artefacts Scrum : Le Product Increment

Le **Product Increment** est le produit potentiellement livrable qui résulte de la réalisation des éléments du **Sprint Backlog** au cours d'un **sprint**.

Il s'agit d'une **version fonctionnelle du produit** qui intègre toutes les **fonctionnalités** et les **améliorations développées** lors des sprints précédents, ainsi que les **nouvelles fonctionnalités** développées lors du sprint en cours.

Le **Product Increment** doit être dans un état "**prêt à être livré**" à la fin du **sprint**, ce qui signifie qu'il doit être **testé, validé et conforme** aux critères d'acceptation définis par le **Product Owner**.

Pour résumer, si le projet concerne la **réalisation d'un site web** par exemple, le **Product Increment** sera **le site web** qui sera montré au client par exemple à la **fin du sprint**, avec les **nouvelles fonctionnalités ajoutées** lors du sprint.

# Évènements Scrum

Les **événements Scrum** sont les **étapes clés** qui ont lieu tout au long du processus de développement Scrum pour **planifier, suivre et améliorer** le travail de l'équipe.

Il y a **5 événements principaux dans Scrum** :

- ☐ Le sprint planning
- ☐ Le sprint
- ☐ Le daily scrum
- ☐ La sprint review
- ☐ La sprint retrospect



# Évènements Scrum : Le Sprint Planning

Dans la **méthode Scrum**, l'avancée du projet se fait en **itération** dont le **point de départ est le Sprint Planning**.

Le **Sprint Planning** est une **réunion** au cours de laquelle l'équipe Scrum **planifie le travail à réaliser** au cours du **sprint à venir**.

Le **Product Owner** présente les **éléments prioritaires du Product Backlog**, et l'équipe de développement choisit les éléments à inclure dans le Sprint Backlog en fonction de la capacité de l'équipe et des contraintes du sprint.

L'**équipe de développement** élabore ensuite un **plan** pour réaliser les éléments du **Sprint Backlog**, en déterminant les tâches et les responsabilités pour chaque membre de l'équipe.

C'est aussi durant cette réunion (ou dans une autre réunion séparée, le **Backlog Refinement**) que l'équipe de développement va **définir le niveau de difficulté** et **affiner les nouvelles « User Stories »** ajoutées par le **Product Owner**.

Pour résumer, le **Sprint Planning** sert à **définir le travail qui devra être réalisé par l'équipe de développement lors du sprint à venir**.

# Évènements Scrum : Le Sprint

Le **sprint** est la période de temps fixe, généralement comprise **entre 1 et 4 semaines**, au cours de laquelle l'équipe de développement **travaille pour accomplir l'ensemble des objectifs spécifiques** liés au projet, définis dans le **Sprint Backlog** (créé durant le Sprint Planning).

L'**objectif** d'un sprint est de produire un **Product Increment** potentiellement livrable à la fin de ce dernier.

Chaque jour, **avant de commencer à travailler**, l'ensemble de l'équipe se réunit pour le **Daily Scrum**.

# Évènements Scrum : Le Daily Scrum

Le **Daily Scrum** est une **réunion quotidienne** de courte durée (généralement **15 minutes**) au cours de laquelle l'équipe de développement **fait le point** sur l'avancement du travail, identifie les **obstacles** éventuels et coordonne les efforts pour la journée.

Chaque membre de l'équipe de développement partage brièvement **ce qu'il a accompli** depuis le dernier **Daily Scrum**, ce qu'il **prévoit d'accomplir** avant le prochain **Daily Scrum**, et les **obstacles** ou **problèmes** qu'il rencontre.

Le **Daily Scrum** permet de maintenir une **communication ouverte** et **transparente** au sein de l'équipe et de **résoudre rapidement les problèmes** qui pourraient entraver la progression du travail.

# Évènements Scrum : La Sprint Review

La **Sprint Review** est une réunion qui a lieu **à la fin de chaque sprint** pour présenter le **Product Increment terminé** à l'équipe, aux parties prenantes et aux clients.

L'équipe de développement **démontre les fonctionnalités** et les **améliorations** développées au cours du sprint, et **recueille les commentaires** et les **suggestions** des parties prenantes.

Le **Product Owner** utilise ces commentaires pour **affiner le Product Backlog** et **ajuster les priorités** pour les **sprints** futurs.

La **Sprint Review** est l'occasion de **valider le travail accompli**, d'obtenir des **retours** d'information précieux et d'**aligner l'équipe** et les parties prenantes sur les **objectifs** et les attentes du projet.

Pour résumer, la **Sprint Review** sert à présenter le **résultat du travail accompli** durant le **sprint** au client et à l'ensemble de l'équipe, afin de recueillir leurs retours et **valider les fonctionnalités développées..**



# Évènements Scrum : La Sprint Retrospective

La **Sprint Retrospective** est une **réunion qui a lieu après la Sprint Review** et **avant le début du prochain sprint**, au cours de laquelle l'équipe Scrum examine le processus de développement et discute des **améliorations** possibles pour les sprints futurs.

L'équipe **identifie** ce qui a **bien fonctionné**, ce qui n'a **pas fonctionné** et les domaines où des **améliorations** peuvent être apportées.

L'**objectif** de la **Sprint Retrospective** est de **favoriser** l'amélioration continue, l'apprentissage et l'adaptation au sein de l'équipe, et de s'assurer que les leçons tirées des sprints précédents sont intégrées dans les sprints futurs.

Pour résumer, la **Sprint Retrospective** est une **réunion** visant à **évaluer le déroulement du sprint actuel** et à **identifier des améliorations** pour les prochains sprints.

# Mise en pratique avec SCRUM

1. Création d'un backlog
2. Priorisation des users stories
3. Technique du planning poker

# Création d'un Backlog

**Atelier : Mettre Scrum en pratique - Partie 1 (30min)**

**Objectif :** Créer un Product Backlog.

**Groupe :** de 4 personnes

**Définir** qui est le **Scrum Master** et qui est le **Product Owner**. Le reste du groupe représentera **l'équipe de développement**.

**Outils possible :** fichier texte, trello (si vous connaissez bien) ou figjam (modèle du formateur)

**Ne pas estimer la priorité et la difficulté de développement des User Stories pour le moment.**

**Le projet :** Le **client** (le formateur) veut un site web. À l'équipe de s'organiser en respectant les **principes Scrum** pour **recueillir les informations** nécessaires à la création du **Product Backlog**. Le formateur passera dans chaque groupe pendant 5min pour vous donner ses besoins.

# Priorisation des user stories

## Atelier : Mettre Scrum en pratique - Partie 2 (20min)

**Objectif :** Reprendre le **Product Backlog précédent** puis **attribuer une priorité** à chacune des **User Stories**, en partant de « 1 » pour la plus importante.

**Outils possible :** fichier texte, trello (si vous connaissez bien) ou figjam (modèle du formateur)

### Attention à bien réfléchir aux priorités !

- ☐ Si une fonctionnalité dépend d'une autre pour fonctionner par exemple, elle devrait être moins prioritaire que cette dernière.
- ☐ Les fonctionnalités les plus importantes à forte valeur ajoutée pour le client sont à prioriser.
- ☐ Utiliser la technique de priorisation des **User Stories** « **MoSCoW** » pour vous aider :
  - **Must have (Doit avoir) :** Ces éléments sont absolument nécessaires pour le projet et doivent être livrés dans les délais impartis. Sans ces éléments, le projet serait considéré comme un échec.
  - **Should have (Devrait avoir) :** Ces éléments sont très importants pour le projet mais peuvent être reportés ou supprimés si nécessaire. Ils sont considérés comme critiques mais pas absolument nécessaires.
  - **Could have (Pourrait avoir) :** Ces éléments sont souhaitables mais pas essentiels pour le projet. Ils peuvent être inclus si le temps et les ressources le permettent.
  - **Won't have (N'aura pas) :** Ces éléments ne sont pas prioritaires pour le projet actuel, mais peuvent être envisagés pour des versions futures ou des itérations ultérieures.

# Technique du planning poker

**Atelier : Mettre Scrum en pratique - Partie 3 (30min)**

**Objectif :** Reprendre le **Product Backlog précédent** puis **estimer le niveau de difficulté** de chacune des **User Stories**.

**Outils possible :** fichier texte, trello (si vous connaissez bien) ou figjam (modèle du formateur)

Pour **estimer** chaque **User Story**, nous allons utiliser **la technique du Planning Poker** :

Nous utiliserons les valeurs de la **suite de Fibonacci** (1, 2, 3, 5, 8, 13, 21, 34, ...) pour estimer la difficulté d'une User Story. **Plus la valeur est grande, plus la User Story est estimée complexe ou longue à mettre en place.**

**Chaque membre** de l'équipe de développement **choisi une valeur** pour estimer la difficulté d'une User Story, **puis tout le monde révèle son choix**. Si les **estimations sont cohérentes**, notez l'estimation pour cette User Story. Si les **estimations varient considérablement**, les personnes ayant voté les estimations les plus élevées et les plus basses **expliquent leur raisonnement**, puis **répétez le processus** jusqu'à ce qu'un consensus soit atteint.



# Utilisation de Jira

# Utilisation de Jira

1. Introduction sur Jira
2. Création de tâches
3. Gestion du Workflow
4. Le Backlog dans Jira



# Introduction sur Jira

## Qu'est- ce que Jira ?

Jira est un **outil de gestion de projet** et de **suivi des problèmes développé** par **Atlassian**. Il est couramment utilisé pour le suivi des bugs, la gestion des projets agiles et la gestion des tâches dans divers types d'organisations. **Jira** permet aux équipes de **planifier**, **suivre** et **gérer** les projets de manière **efficace** et **collaborative**.

Il s'agit d'un **outils très complet** et qui demandera beaucoup de temps d'adaptation mais il est très utilisés en entreprise. Nous allons donc survoler certains aspects de cet outil, libre à vous de l'approfondir par la suite.

Prise en main :

- 1 – **Inscrivez-vous** par ici : [Inscription sur Jira](#)
- 2 – Une fois inscrit, dans le menu burger en haut à gauche, **sélectionnez « Jira Software »**
- 3 – **Démarrez la version gratuite** (donnez un nom exemple : votre prénom) et poursuivez
- 4 - **Choisissez kanban**, donnez un nom au projet(ex : projet de test kanban), le niveau de connaissance du sujet "bas"

Le but ici sera **d'expérimenter Kanban** grâce à **Jira**, donc nous allons **créer des tâches**, les **configurer** et **agir** sur le comportement des **colonnes** et leurs liens via **la gestion du Workflow**.

# Création de tickets/tâches

L'interface de création de ticket dans Jira est utilisée pour ajouter de nouvelles tâches, bugs, user stories, ou autres types de tickets à notre projet.

1. **Projet** : Sélectionnez le projet auquel le ticket sera associé. Ici, "**Mon projet Kanban**". Cela permet d'organiser les tickets par projet.
2. **Type de ticket** : Choisissez le type de ticket, par exemple « **Tâche** ». Cela aide à catégoriser les tickets selon leur nature.
3. **État** : Définit l'état initial du ticket, par exemple « **To Do** ». Cela permet d'indiquer où en est le ticket dans le flux de travail dès sa création.
4. **Résumé** : Pour donner un **titre** ou un **résumé bref** de la tâche ou du problème. Cela donne une vue d'ensemble rapide de ce qu'est le ticket
5. **Description** : Pour donner des **détails complets sur le ticket**, y compris les objectifs, les spécifications ou les problèmes à résoudre. Cela permet de fournir toutes les informations nécessaires pour comprendre et traiter le ticket.

The screenshot shows the 'Créer un ticket' (Create Ticket) form in Jira. At the top right are icons for a minus sign, a cursor, and a close button. Below the title, a note states 'Les champs obligatoires sont marqués d'un astérisque \*' (Required fields are marked with an asterisk \*). The form includes several sections: 'Projet\*' (Project) with a dropdown menu showing 'Mon projet Kanban (KAN)'; 'Type de ticket\*' (Ticket type) with a dropdown menu showing 'Tâche' (Task); a link 'En savoir plus sur les types de ticket' (Learn more about ticket types); 'État' (State) with a dropdown menu showing 'To Do' and a note 'Il s'agit de l'état initial des tickets lors de leur création' (This is the initial state of tickets when they are created); 'Résumé\*' (Summary) with a text input field and a red error message 'Résumé doit être renseigné' (Summary must be filled in); and 'Description' with a rich text editor toolbar containing options like 'Texte normal', bold, italic, text color, bulleted list, numbered list, link, image, mention, emoji, table, code, help, and a plus sign for more options. The description area also contains a prompt: 'Les mots ne suffisent pas ? Saisissez : pour ajouter une émoticône' (Words aren't enough? Type : to add an emoji).

# Création de tickets/tâches

L'interface de création de ticket dans Jira est utilisée pour ajouter de nouvelles tâches, bugs, user stories, ou autres types de tickets à notre projet.

1. **Responsable** : Personne à qui la tâche est assignée. Par défaut, il est souvent "Non assigné". Cela permet d'identifier la personne responsable de la réalisation du ticket.
2. **Étiquettes** : Mots-clés ou tags pour classer et retrouver facilement les tickets. Cela permet de faciliter la recherche et le filtrage des tickets.
3. **Priorité** : Utilisé pour indiquer l'importance ou l'urgence d'un ticket par rapport aux autres. Cela aide les équipes à identifier quelles tâches ou bugs doivent être traités en premier.
4. **Story point estimate** : Utilisé pour estimer la quantité de travail nécessaire pour compléter un ticket, souvent utilisé dans les méthodologies agiles comme Scrum. Cela permet d'évaluer la complexité, l'effort et le temps requis pour une tâche ou une user story.
5. **Rapporteur** : Personne qui a créé le ticket. Cela permet de savoir qui a identifié le besoin ou le problème.

The screenshot shows the Jira ticket creation interface with the following fields:

- Responsable**: A dropdown menu showing "Automatique" with a person icon.
- Me l'assigner**: A blue link.
- Étiquettes**: A dropdown menu showing "Sélectionner une étiquette" with a downward arrow.
- Priorité**: A dropdown menu showing "Medium" with a yellow bar icon and a downward arrow.
- En savoir plus sur les niveaux de priorité**: A blue link.
- Story point estimate**: A text input field with a downward arrow.
- Measurement of complexity and/or size of a requirement.**: A small text label below the story point estimate field.
- Rapporteur\***: A dropdown menu showing "Test-formateur" with a person icon.

# Création de tickets/tâches

L'interface de création de ticket dans Jira est utilisée pour ajouter de nouvelles tâches, bugs, user stories, ou autres types de tickets à notre projet.

1. **Pièce jointe** : Permet de joindre des fichiers pertinents au ticket (images, documents, etc.). Cela fournit des informations supplémentaires ou des preuves visuelles.
2. **Ticket liés** : Option pour lier ce ticket à d'autres tickets, en définissant la relation (par exemple, blocks, relates to, etc.). Cela permet de montrer la dépendance entre tickets, facilitant la gestion des tâches liées.
3. **Flagged** : Option pour marquer le ticket comme bloqué ou nécessitant une attention particulière. Cela permet d'indiquer que le ticket a des obstacles à résoudre avant de pouvoir avancer.
4. **Créer un autre ticket** : Case à cocher si on souhaite créer un autre ticket immédiatement après celui-ci.



The screenshot displays the Jira ticket creation form with the following sections:

- Pièce jointe**: A dashed box containing a cloud icon and the text "Glisser-déposer des fichiers pour les joindre ou [parcourir](#)".
- Tickets liés**: Two dropdown menus. The first is set to "blocks". The second is labeled "Sélectionner un ticket".
- Flagged**: A checkbox labeled "Impediment" with the text "Allows to flag issues with impediments." below it.
- At the bottom right, there are two buttons: "Annuler" (grey) and "Créer" (blue).

# Gestion du Workflow

Un workflow dans Jira est une représentation visuelle des étapes (ou états) par lesquelles passent les tickets (tâches, bugs, etc.) au cours de leur cycle de vie, depuis leur création jusqu'à leur achèvement.

## 1. À quoi sert un Workflow ?

1. **Organiser le travail** : Le workflow aide à structurer et organiser les différentes étapes nécessaires pour compléter une tâche.
2. **Suivre le progrès** : Il permet de visualiser où en est chaque ticket dans le processus de travail.
3. **Définir les transitions** : Le workflow définit comment et quand un ticket peut passer d'un état à un autre.

## 2. Composants Clés d'un Workflow :

1. **États (ou Statuts)** : Les différentes phases par lesquelles un ticket passe, comme "À faire" (To Do), "En cours" (In Progress), "Terminé" (Done).
2. **Transitions** : Les actions ou chemins permettant de passer d'un état à un autre, par exemple, "Commencer à travailler" pour passer de "À faire" à "En cours".

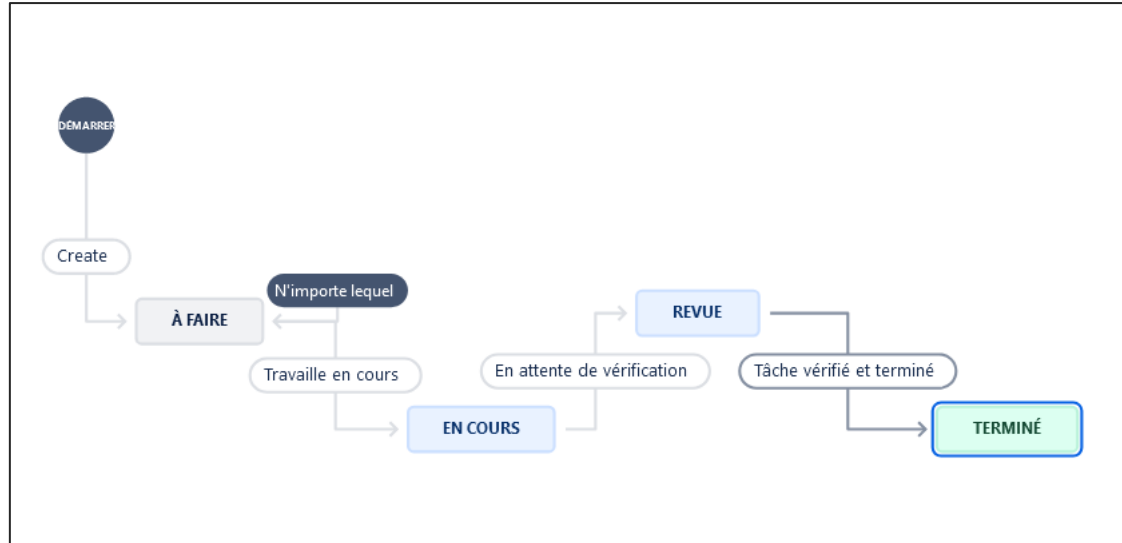


# Gestion du Workflow

## Exemple :

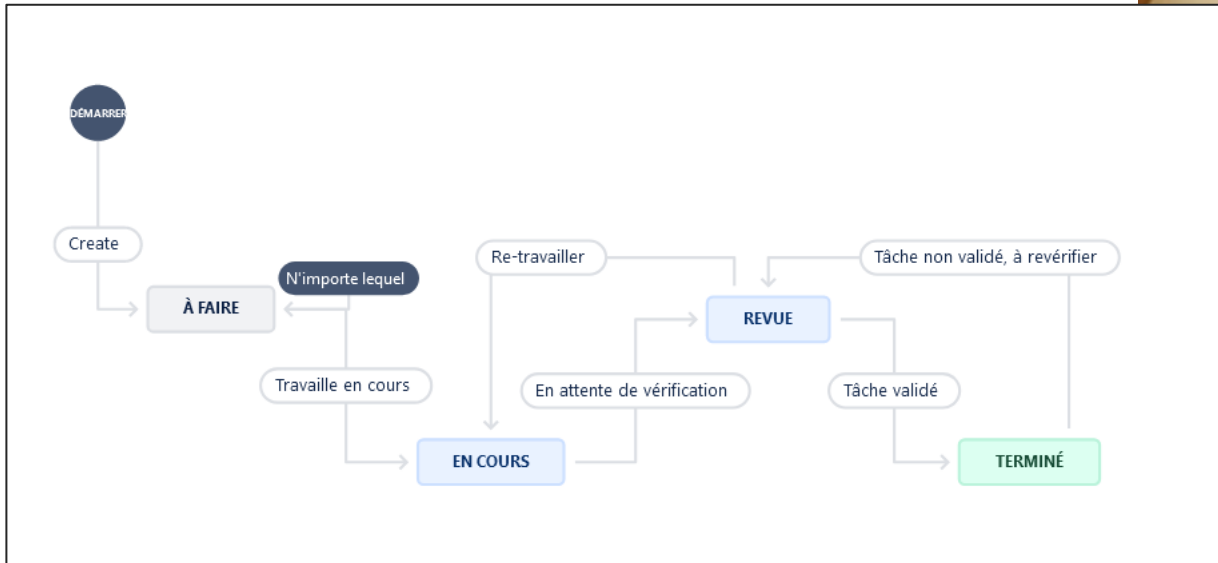
Imaginez une tâche simple comme "**Rédiger un rapport**". Le workflow pourrait ressembler à ceci :

1. **À faire (To Do)** : La tâche est créée mais pas encore commencée.
2. **En cours (In Progress)** : Quelqu'un travaille sur la tâche.
3. **Revue (Review)** : La tâche est terminée et en attente de vérification.
4. **Terminé (Done)** : La tâche a été vérifiée et approuvée.



## Exercice 03 :

À l'aide du screen, essayez de reproduire ce workflow :



# Le Backlog dans Jira

## Qu'est-ce que le Backlog ?

Le **backlog** dans Jira est une **liste priorisée de toutes les tâches, user stories, bugs**, et autres éléments de travail qui doivent être complétés pour **un projet**. Il est utilisé pour **organiser** et **gérer** le travail à effectuer par l'équipe.

## Types de Backlogs :

### 1. Backlog Produit :

- C'est la **liste principale** de **toutes les fonctionnalités, améliorations et corrections** nécessaires pour le projet.
- C'est géré par le **Product Owner** qui **ajoute, priorise et met à jour** les éléments.

### 2. Backlog de Sprint :

- C'est une **sélection de tâches** du **backlog produit** qui seront **complétées dans un sprint** (période de travail définie, généralement 2 à 4 semaines).
- C'est géré par **l'équipe de développement** lors de la **réunion de planification de sprint**.



# Le Backlog dans Jira

## Exemple de backlog dans Jira :

Projets / Mon projet Kanban

### Backlog

Rechercher



Epic ▾







☐ Tableau Sprint 1 22 juin – 6 juil. (4 tickets)

1 16 0

Terminer le sprint



<input checked="" type="checkbox"/>	KAN-6 Ticket B	REVUE ▾	3	=	
<input checked="" type="checkbox"/>	KAN-5 Ticket A	REVUE ▾	8	>	
<input checked="" type="checkbox"/>	KAN-7 Ticket C	À FAIRE ▾	1	<	
<input checked="" type="checkbox"/>	KAN-8 Ticket D	EN COURS ▾	5	=	

+ Créer un ticket

☐ Tableau Sprint 2 6 juil. – 20 juil. (2 tickets)

0 0 0

Démarrer un sprint



<input checked="" type="checkbox"/>	KAN-9 Ticket A (pour sprint2)	À FAIRE ▾	-	=	
<input checked="" type="checkbox"/>	KAN-10 Ticket B (pour sprint2)	À FAIRE ▾	-	=	

+ Créer un ticket






2 tickets | Estimation : 0

☐ Backlog (4 tickets)

0 0 0

Créer un sprint

<input checked="" type="checkbox"/>	KAN-11 Ticket C (pour sprint2)	À FAIRE ▾	-	=	
<input checked="" type="checkbox"/>	KAN-12 Ticket 001 (pour les sprints futur)	À FAIRE ▾	-	=	
<input checked="" type="checkbox"/>	KAN-13 Ticket 002 (pour les sprints futur)	À FAIRE ▾	-	=	
<input checked="" type="checkbox"/>	KAN-14 Ticket 066 (pour les sprints futur)	À FAIRE ▾	-	=	

+ Créer un ticket



# **Outils du développeur agile**

# Les outils du développeur Agile

Il existe un certain nombre d'outils utilisés pour **la gestion de projet**, dont voici une **liste non exhaustive** :



**Jira (Atlassian)** : Outil de gestion de projets et de suivi des bugs qui facilite la collaboration au sein des équipes Agile. Il offre des tableaux Kanban et Scrum personnalisables, ainsi que des fonctionnalités pour planifier et suivre les tâches, les User Stories et les problèmes en temps réel.



**YouTrack (JetBrains)** : idem que Jira.



**Trello (Atlassian)** : Outil de gestion de projets visuel et flexible basé sur le concept de tableaux Kanban. Il permet aux équipes de collaborer, d'organiser et de prioriser leurs tâches en utilisant des cartes, des listes et des tableaux pour une gestion de projet simplifiée et efficace.

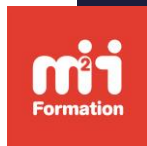
# Les outils du développeur Agile



**Slack :** Plateforme de communication et de collaboration pour les équipes, permettant d'échanger des messages, de partager des fichiers et d'intégrer d'autres outils pour améliorer la productivité et l'efficacité au travail.



**Microsoft Teams :** Solution de communication et de collaboration offrant des fonctionnalités de chat, de vidéoconférence, de partage de documents et d'intégration d'applications pour faciliter le travail en équipe et la coordination de projets.



# Fin du module

---



m2information.fr