Agile

Scrum

CONSULTING BSTORM

Concepts de SCRUM

Origine

SCRUM a été formalisé en 1995 par:

- Jeff Sutherland
  - http://jeffsutherland.com/
- Ken Schwaber
  - http://kenschwaber.wordpress.com/

Scrum

Définition

“A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value”

Scrum Guide

Scrum est :

- Léger
- Facile à comprendre
- Difficile à maîtriser

Big Picture

Scrum est un Framework d'Agile qui produit dans un laps de temps (**Sprint** de 1 à 4 semaines) une version de l'application qui fonctionne

- Les exigences et les priorités sont définies dans le **Product Backlog**.
- L'équipe Scrum s'auto-organise.
- A chaque fin de Sprint, la décision de livrer le produit dans l'état ou de continuer à l'améliorer est prise.

Agile vs Scrum

Agile	Scrum
philosophie	**framework**

Scrum is A Framework, Not A Process

Process

- Normatif : dicter étape à étape.
- Répétable de projet en projet.
- Idéal pour une utilisation dans un contexte bien défini.

Framework (“*cadre*”)

- Non normatif.
- Définition des tâches clés et d’une routine.
- Plus adaptable aux environnements volatiles.

SCRUM framework + notre environnement = notre process

CONSULTING BSTORM

Scrum is A Framework, Not A Process

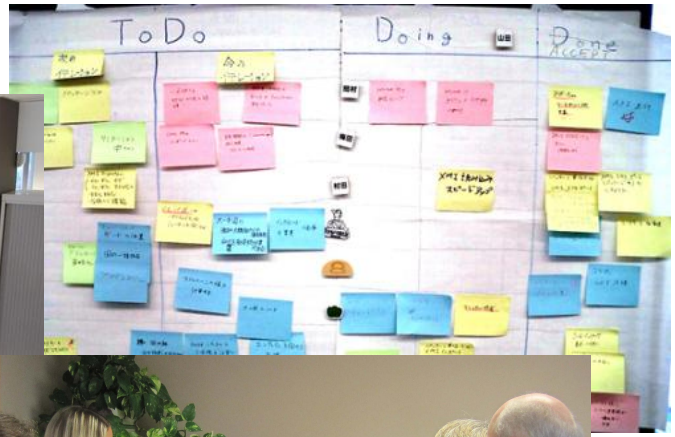
mon
SCRUM

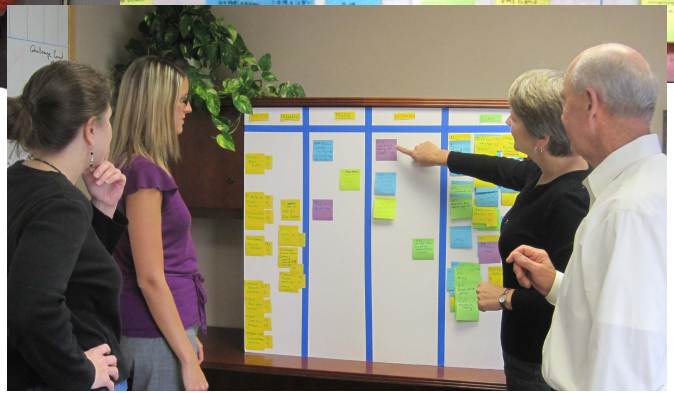
votre
SCRUM

BSTORM CONSULTING

Scrum is A Framework, Not A Process

SCRUM en images





SCRUM en images

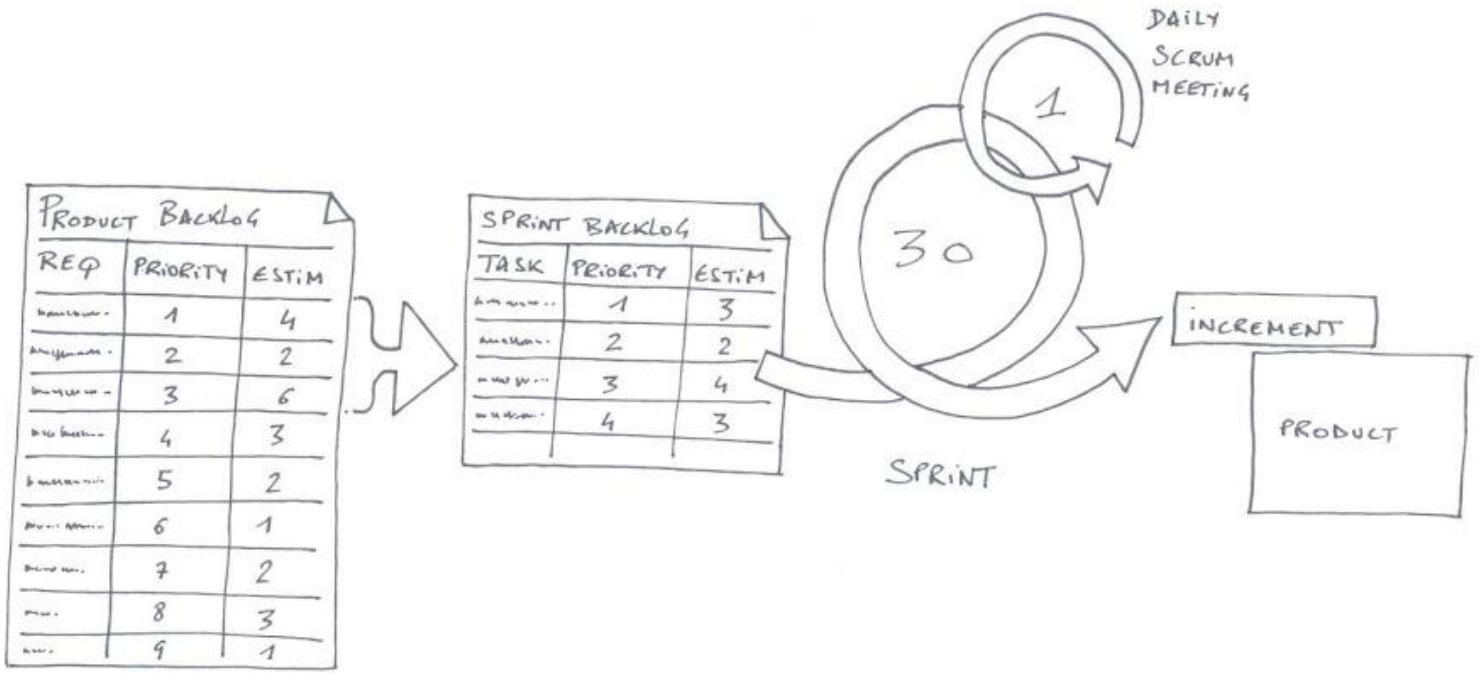
Le cycle de vie d’un projet SCRUM

Le Processus

Un processus SCRUM consiste en 3 phases

1. **La phase Initiale**
   - Phase d'analyse.
   - Création d'un Product Backlog.
2. **La phase de Sprints**
   - Phase de développement de 1 à 4 semaines.
   - Création d'un Sprint Backlog.
3. **La phase de Clôture**
   - Préparation du produit pour une livraison.

Le Processus



Phase Initiale

La phase initiale aboutit à :

1. La conceptualisation et l'analyse du système.
2. La mise en place d'un Product Backlog.
    - Liste de tâches restant à effectuer.
3. La définition approximative de la date de livraison.
4. La formation et la constitution de l'équipe de développement.
5. L'analyse du risque et des coûts.

Phase Initiale

Phase courte mais qui requiert des compétences variées :

- connaissance du marché,
- des utilisateurs potentiels,
- des ressources techniques issues d'autres développement similaires,
- etc...

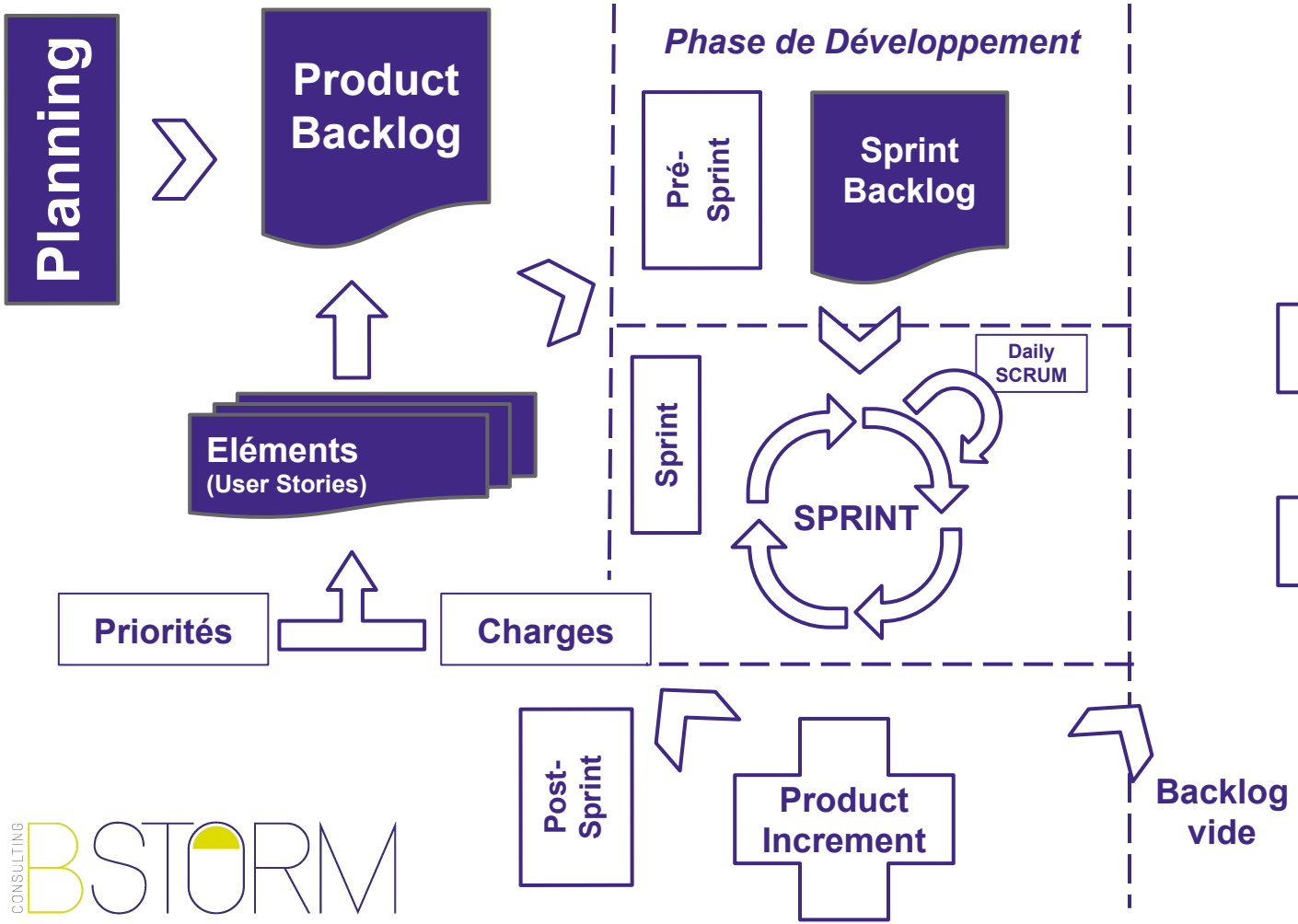
Phase de Sprint

- Phase où le développement est, à proprement parlé, réalisé.
  - Analyse, conception, implémentation... de chaque élément.
- Les sprints sont guidés par une liste de tâches provenant du Product Backlog formant le **Sprint Backlog**.
- Durée de 1 à 4 semaines.
- Isolation de l'équipe de toute influence extérieure.

Phase de clôture

Préparation du produit pour une livraison :

- intégration,
- tests systèmes,
- documentation utilisateur,
- préparation de supports de formation,
- préparation de supports marketing,
- etc...


Planning
Product Backlog
Phase de Développement
Phase Finale
Pré-Sprint
Sprint Backlog
Eléments
(User Stories)
Sprint
Daily SCRUM
SPRINT
Livraison finale
Documentation
Priorités
Charges
Post-Sprint
Product Increment
Backlog vide

Les rôles SCRUM

Les rôles

1. **Product Owner**
   - Objectif : maximiser la valeur du produit.
   - Définit et partage la vision globale du produit.
2. **SCRUM Master**
   - Aide l'équipe à respecter les règles du Framework Scrum.
3. **SCRUM Team**
   - Responsable du déroulement de chaque Sprint (créer et livrer le produit).

Product Owner

Les rôles du Product Owner

- Est responsable de la gestion du Product Backlog.
- S'assure que le Product Backlog soit compris pour chacun.
- S'assure de la valeur du travail fourni par l'équipe.

Les caractéristiques

- Un seul PO par projet.
  - Une seule Vision du Produit (référant).
- Leader (communication) et Team Player (négociation).
- Visionnaire, Disponible et Qualifié.

3 facettes du Product Owner

1. **Product Management**
   - Définir et communiquer la Vision du Produit.
   - Définir la stratégie de livraison des différentes versions du produit sur le marché.
2. **Business Analysis**
   - Identifier les besoins et exigences des utilisateurs.
   - Mise à jour du Product Backlog pour refléter ces besoins et exigences.
3. **Project Management**
   - Gérer le budget du projet.
   - S’assurer du respect de chaque échéance.

Product Owner

Collaboration avec l'équipe SCRUM

- Lui même membre de la SCRUM Team
- Le Product Owner doit se trouver sur le même site que la SCRUM Team

Collaboration avec le SCRUM Master

- Rôles complémentaires
- Product Owner est responsable du "QUOI"
- SCRUM Master est responsable du "COMMENT"
- Une personne ne doit pas cumuler les 2 rôles

Un Product Owner pour deux équipes

Product Owner est une activité à plein-temps. Dans de rares cas, un PO peut s'occuper de 2 Scrum Team en même temps seulement si 3 conditions sont réunis :

1. Les deux équipes travaillant dans le même domaine.
2. Les deux équipes travaillant dans le même produit.
3. Les deux équipes ont de l'expérience avec Scrum.

Ex : une équipe de développement et l'autre sur la maintenance d'un même produit.

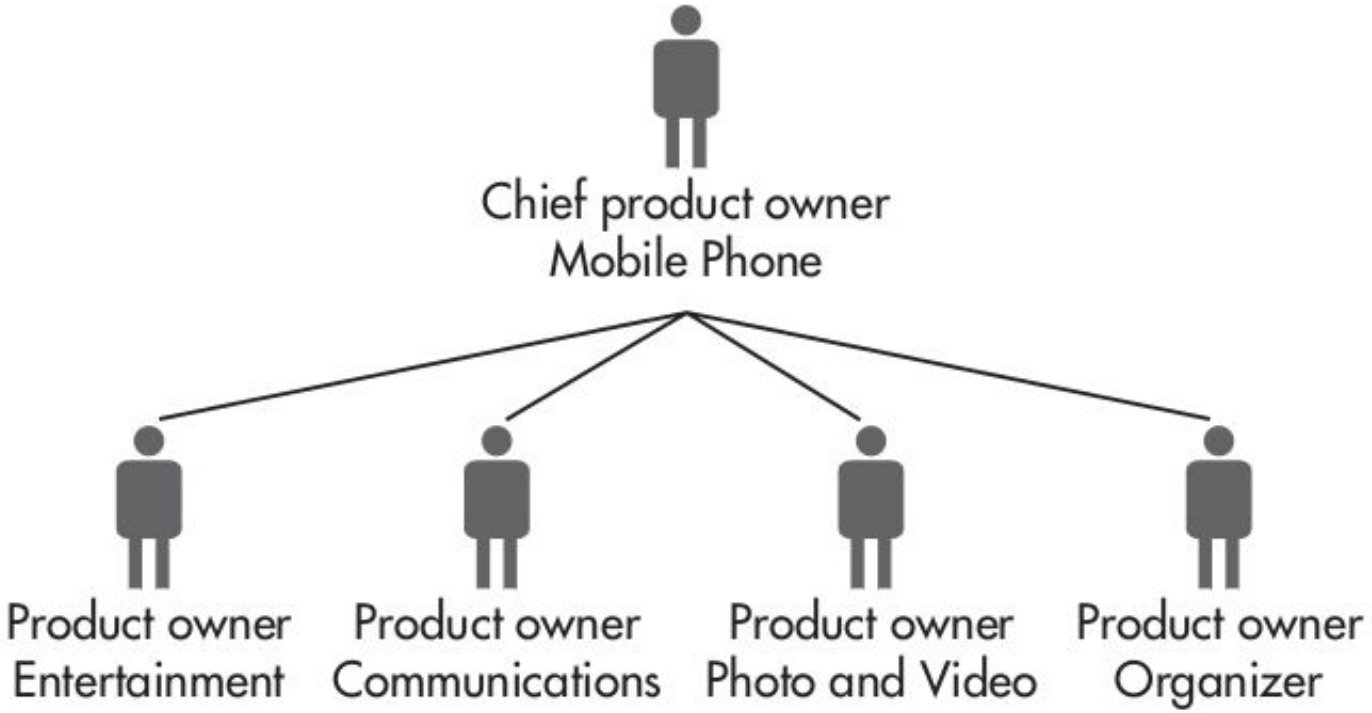
Chief Product Owner

Difficile pour un Product Owner de s'occuper de 2 SCRUM Team. Quid de projets comprenant plus de 2 SCRUM Team alors que le Product Owner est unique ?

Introduction du rôle du **Chief Product Owner**

- Responsable de coordonner différents Product Owner.
- Responsable de la vision du produit.

Chief Product Owner



Erreurs courantes

1. Ne pas donner assez de pouvoir au Product Owner
    - Il ne dispose pas assez d'autonomie
2. Surcharger le Product Owner
    - Il risque de se focaliser ce sur quoi il sera jugé, à savoir le Project Management (budget et temps), en mettant de côté l'écoute des clients et d'analyse.
3. Ne pas séparer (transversalement) les tâches du Product Owner entre différentes personnes

SCRUM Master - rôles

- Assure que l'équipe Scrum fonctionne aussi efficacement que possible en respectant les règles du Framework Scrum.
- Aide l'équipe à travailler de façon autonome et à s'améliorer constamment.
- Anime les réunions de Pré-Sprint, Scrum, Post-Sprint.
- Elimine les obstacles qui ralentissent l'équipe.
- Communique avec le management (ex. rapports d'avancement).

SCRUM Master - caractéristiques

- **Bien connaître SCRUM**
- **Bien organisé**
  - Garde l'équipe sur la bonne voie.
- **Talent de communication**
  - Pouvoir résoudre les conflits.
- **Etre respecté par l'équipe et le Product Owner**
  - Etre capable de guider sans imposer.
- **Etre respecté par l'organisation**
  - Protège l'équipe des influences extérieures.
- **Idéalement, est élu par la Scrum Team**
  - Pour les équipes expérimentées.

“Servant Leaders”

- Le SCRUM Master n'est pas un chef d’équipe ou de projet
  - Ce n’est pas la Scrum Team qui est aux services du Scrum Master mais l’inverse.
- Il ne dirige pas, il n'impose pas, il n’est pas craint.
  - Il fait partie de l'équipe.
  - Il aide l’équipe à mieux travailler.
  - Ceux qui font le travail sont les mieux placés pour déterminer *comment* faire le travail.

Backgrounds pour être Scrum Masters

1. **Développeur**
   - Respecté par ses collègues.
   - Bonne connaissance des processus de développement.
2. **Testeur**
   - Vision globale du projet.
   - Habitué au relation 1-to-1 avec chaque membre de l'équipe.
   - "Spécialiste" des bugs et erreurs de développement.
3. **Project Manager**
   - Bien organisé.
   - Bonne relation avec toute l'organisation.
   - Attention aux habitudes autoritaires !

À plein temps ?

1. **Full-time Dedicated Scrum Master**
   - Plein temps focalisé sur une seule équipe.
   - Permet d'être focus sur le projet.
   - Permet un suivi plus approfondi de l'équipe.
2. **Full-time Shared Scrum Master**
   - Plein temps partagé entre plusieurs (2 à 3) équipes.
   - Permet de partager le coût d'un Scrum Master entre plusieurs équipes.
3. **Part-time Scrum Master**
   - Temps partiel avec une seule équipe.
   - Plus proche des problématiques de l'équipe.
   - Idéal pour une équipe Scrum débutante.

L’équipe de développement

Responsable du déroulement de chaque Sprint

- Ce qui est mis en oeuvre pour atteindre les objectifs du Sprint.

Impliqué dans :

- L'estimation de la charge de travail.
- La création du Sprint Backlog.
- L'identification des freins à l'avancement du projet.

L’équipe de développement

Responsabilités	Caractéristiques
Créer le produit	Apprécie participer à la création du produit Dispose d’au moins une compétence nécessaire à la création du produit
S’auto-organise et s’auto-gère	Fait preuve d’initiative et d'indépendance
Est pluri-disciplinaire	A de la curiosité Désire contribuer au delà de sa zone de maîtrise Apprécie apprendre de nouvelles compétences Partage volontier sa connaissance
Est dédiée et rassemblée	

L’équipe de développement - pluridisciplinaire

Une Scrum Team doit être pluridisciplinaire pour être autonome, sans dépendance externe.

- Mettre de côté les titres qui inhibent les compétences.
  - Testeurs, développeur .NET, développeur Java…
  - Tous sont appelés développeur.
- Travailler à étendre ses compétences.
  - Apprendre quelque chose de nouveau chaque sprint.
- Se proposer pour aider un membre qui fait face à un obstacle.
- Être flexible.

L’équipe de développement - s’auto-organise

Afin de tirer avantage de la connaissance et l’expérience des membres de l’équipe

- S’engage vis-à-vis des objectifs du sprint
- Identifie ses tâches
- Estime l’effort nécessaire à chaque tâche
- Se focalise sur la communication
- Collabore
- Prend des décisions sur base de consensus
- Participe activement

L’équipe de développement - s’auto-gère

Contrôle la manière dont elle travaille afin d’assurer le succès du projet

- Autorise la fluctuation du leadership
- S’appuie sur les outils et processus agile
- Rapport régulièrement et de manière transparente l’avancement du projet
- Règle les problèmes au sein de l’équipe
- Créer une convention d’équipe
- Inspect and Adapt

First Among Equals

Le Product Owner et Scrum Master sont des rôles importants mais ils ne sont pas les chefs de la Team Scrum.

- La Team Scrum est autonome.
  - Ce qui permet au PO et SM de se concentrer sur le travail plutôt sur la gestion au quotidien de l'équipe.
- Le Product Owner est considéré comme "**First Among Equals**" (premier parmi ses pairs)
  - A le dernier mot, en cas de conflit ou indécision..
  - Puisque seul responsable de la vision produit.

Taille réduite (recommandée) de l'équipe

3 à 9

membres

Facilite la bonne communication

Maintien l'unicité de l'équipe

Évite la création de sous-équipes

Encourage la pluridisciplinarité, la communication en face-à-face…

Prise d'initiatives

Réussi ou échoue en tant qu'équipe.

Autres: Management et Client

Management

- Responsable de la décision finale
- Impliqué dans le choix du Product Owner
- Surveille l'avancement du projet
- Peut augmenter / réduire le Backlog en concertation avec le Product Owner

Client

- Participe à l'élaboration du Product Backlog

Les meetings

Meetings

4 types de meetings

1. Sprint Planning Meeting
   - Planification en début de chaque sprint
2. Daily Scrum
   - Réunion quotidienne informelle de toute l'équipe
3. Sprint Review
   - Présentation aux clients
4. Sprint Retrospective
   - Evaluation de l'efficacité du sprint

Sprint Planning Meeting

Réunion de pré-Sprint organisée par le SCRUM Master en 2-temps

- **1ère Phase**
  - Clients, utilisateurs, management, product owner et Scrum Team établissent le Sprint Backlog
  - Décider des objectifs et de la fonctionnalité du prochain Sprint
- **2ème Phase**
  - Le Scrum Master et la Scrum Team organisent le déroulement du Sprint
  - Décide de la manière dont l'incrément de produit est mis en œuvre pendant le sprint.

TIMEBOX: 2 heures par semaine de SPRINT

Daily Scrum

Quoi ?

- Réunion quotidienne informelle de toute l'équipe

Quand et où ?

- Toujours à la même heure et au même endroit

Qui ?

- N'interviennent que les personnes qui travaillent effectivement sur le développement
- Les extérieurs y sont invités pour suivre l'avancement du projet

Daily Scrum

Objectifs

1. Partager les connaissances acquises
2. Faire un point sur l'avancement
3. Donner au management une certaine visibilité sur la progression du projet

Contrôle continu et empirique via 3 questions quotidiennes

1. *Qu'est ce qui a été fait pendant la journée ?*
2. *Que reste-t-il à faire ?*
3. *Quels sont les obstacles qui gênent l'avancement du projet ?*

Daily Scrum

Le Scrum Master prend les décisions que l'équipe est incapable de prendre par elle même et s'engage à apporter une solution à tout ce qui entrave le développement du projet

TIMEBOX: 15 à 30 minutes max

Sprint Review

- Réunion informelle
- L'équipe présente au client ce qui a été développé pendant les 30 jours précédents via une démonstration
- Confronter les résultats du travail de l'équipe avec la complexité et le chaos de l'environnement dans lequel l'application sera utilisée
- Décision de release ou non

TIMEBOX: 1 heure par semaine de Sprint

Sprint Retrospective

Réunion ayant lieu après le Sprint Review et avant le Sprint Planning Meeting.

- Le but du Sprint Rétrospective est de :
  - Inspecter la manière dont s'est déroulée le Sprint précédent quant aux personnes, relations, processus et outils utilisés.
  - Identifier et ordonner les éléments majeurs qui se sont déroulés ainsi que les améliorations potentielles.
  - Créer un plan pour implémenter des améliorations quant à la manière de travailler de l'équipe Scrum.

TIMEBOX: 45 minutes par semaine de Sprint.

User Story

Détaillé de manière appropriée

Les éléments du product backlog sont des user story

En tant que [rôle], **je veux** [action] **afin de** [but]

Exemple:

“En tant qu'utilisateur qui ferme l'application, je souhaite être invité à enregistrer tout ce qui a changé depuis la dernière sauvegarde afin de pouvoir conserver le travail utile et de supprimer le travail erroné.”

Workshop d'écriture des User Stories

- Participants: le Product Owner, partenaires additionnels (ex. utilisateurs), (la SCRUM Team)
- Pas forcément à chaque Sprint
- Génération sur base de Brainstorming
  - On commence avec des "**epics**" et on découpe
- Ecrire un maximum de stories possible
  - Certaines directement implémentable
  - Certaines resteront temporairement des "epics"

Bonnes pratiques

1. Ecrire pour un seul utilisateur.
2. Ecrire à la forme active.
3. Ne pas numéroter les cartes.
4. Ecrire des user stories fermées.

Pourquoi utiliser des User Stories ?

1. Suscite la communication verbale.
2. Compréhensible pour chacun.
3. Taille appropriée pour la planification.
4. Fonctionne bien dans des processus itératifs.

INVEST

User Story doit être :

1. **I**ndependent.
2. **N**egotiable.
3. **V**aluable.
4. **E**stimable.
5. **S**mall.
6. **T**estable.

Independent

La user story doit être autonome, de manière à ce qu'il n'y ait pas de dépendance inhérente à une autre user story.

Négociable

Stimule la conversation

L'utilisateur peut payer avec une carte de crédit.	L'utilisateur peut payer avec une carte de crédit.
Remarque: Acceptez Visa et MasterCard. Considérez American Express. Sur achats de plus de 100 $ demander l'identifiant de la carte le dos de la carte. Le système peut dire quel type de carte il est des deux premiers chiffres du numéro de carte. Collecter le mois et année d'expiration.	Remarque: Acceptez Visa et MasterCard. Considérez American Express. Interface utilisateur: pas besoin d'un champ séparé pour le type de carte.
TROP DE DÉTAIL	**JUST ENOUGH**

Valuable

Aux utilisateurs et aux acheteurs, PAS aux développeurs.

Estimable

Unités utilisées

- Points Story (relatif)
- Jour-hommes (absolut)

Qu'est-ce qui rend une histoire difficile à estimer?

- Manque de connaissance du domaine
- Manque de connaissances techniques
- La taille est trop grande

SMALL

- Un utilisateur peut planifier des vacances.
  - Trop grand
- Un utilisateur peut poster son CV.
  - Trop grand

- Un utilisateur peut ajouter son CV.
- Un utilisateur peut mettre à jour son CV.
- Un utilisateur peut marquer les CV comme inactifs.

Stratégies de décomposition

1. Par étapes d'un Workflow
   - L'utilisateur accomplit une tâche selon un workflow bien établi. On découpe les stories par étapes qui seront développées de façon incrémental.
2. Par scénario
   - On obtient une User Story pour le scénario principal, le cas où tout se passe bien, on en a d'autres pour les cas d'erreurs ou les scénarios alternatifs : quand il se passe x, quand il se passe y.
3. Par séquence dans un scénario
   - Le cas est plus précis, on découpe cette fois-ci une séquence au sein d'un scénario…

Stratégies de décomposition

4. Par opérations
   - Souvent le mode de décomposition le plus évident … Le CRUD (create, Retrieve, Update, Delete) est un bon exemple. Il est souvent utile de découper ou d'en faire deux en même temps…créer un compte, le consulter, le modifier et le supprimer.
5. Par format ou type de données
   - On joue sur le type d'objet (ex : compte titres, espèces… messages en français, anglais, espagnol…)
6. Par type d'entrée, sortie ou configuration
   - Des variations d'un point de vue matériel ou non, selon les configurations mais aussi en termes de moyens de saisie. Cela peut se jouer aussi au niveau de l'interface…

Stratégies de décomposition

7. Par Persona ou rôle
    a. Cette fois-ci, on décompose les user stories en fonction du rôle et de celui qui va utiliser le produit, le fameux « en tant que… ». Pour cela, on peut s'appuyer le user story mapping, une activité menée au début du projet pour définir le backlog de produit et la roadmap produit.
8. Par niveau de connaissance
    a. Le niveau de connaissance acquis sur une fonctionnalité est un bon critère de décomposition…Une story pour ce qui est connu, une autre là où c'est moins maîtrisé. Cela peut déboucher sur un spike (une user story un peu particulière orientée exploration)

Stratégies de décomposition

9. Par niveau de complexité
   - Une user story va par exemple décrire une fonctionnalité dans son mode de réalisation le plus simple, d'autres suivront par un niveau de complexité plus grand
10. Par niveau de qualité attendu
   - Performance, Sécurité, Utilisabilité… ces exigences non fonctionnelles constituent le plus souvent des conditions de satisfaction pour des user stories spécifiques mais elles peuvent permettent également de distinguer des user stories entre elles (ex : afficher en moins de 60 sec, moins de 30 sec ; données en temps réels ou non…)

Testable

- Critères de test concrets.
- Habituellement écrit au verso de la carte.
- La plupart des tests devraient être automatisés.

- L'écran de démarrage disparaît peu de temps après l'exécution du programme.
  - Pas assez concret
- L'écran de démarrage disparaît dans les 4 secondes.
  - Concret & Testable

Testable

- Les tests sont utilisés pour :
  - La spécification
  - Déterminer qu'une user story est terminée
- Ecrire les tests avant de coder
  - Lors des discussions Product Owner / Développeurs
  - Au début du Sprint
- Les tests sont principalement spécifiés par le responsable du produit
- Les tests font partie du processus

Exemple

Une entreprise peut payer une offre d'emploi avec une carte de crédit

- Testez avec Visa, MasterCard, American Express
- Testez avec les bons, les mauvais et les numéros manquants
- Test avec des cartes expirées
- Testez avec différents montants d'achat (y compris une limite supérieure à la carte)

Définition du Done

Afin d’éviter des problèmes de compréhension, il est important de se mettre d’accord de ce qu’est une taché terminé.

La **définition de Done** est une liste d'exigences auxquelles une User Story doit adhérer pour que l'équipe puisse la qualifier de terminé.

Définition du Done

Les critères sur la définition du Done sont définis avant le premier Sprint par :

1. Le Product Owner
2. Le Scrum Master
3. L'équipe de développement

Un élément du product backlog est considéré comme achevé lorsque :

1. Il est implémenté
2. Il est testé
3. Il est intégré
4. Il est documenté

Définition du Done

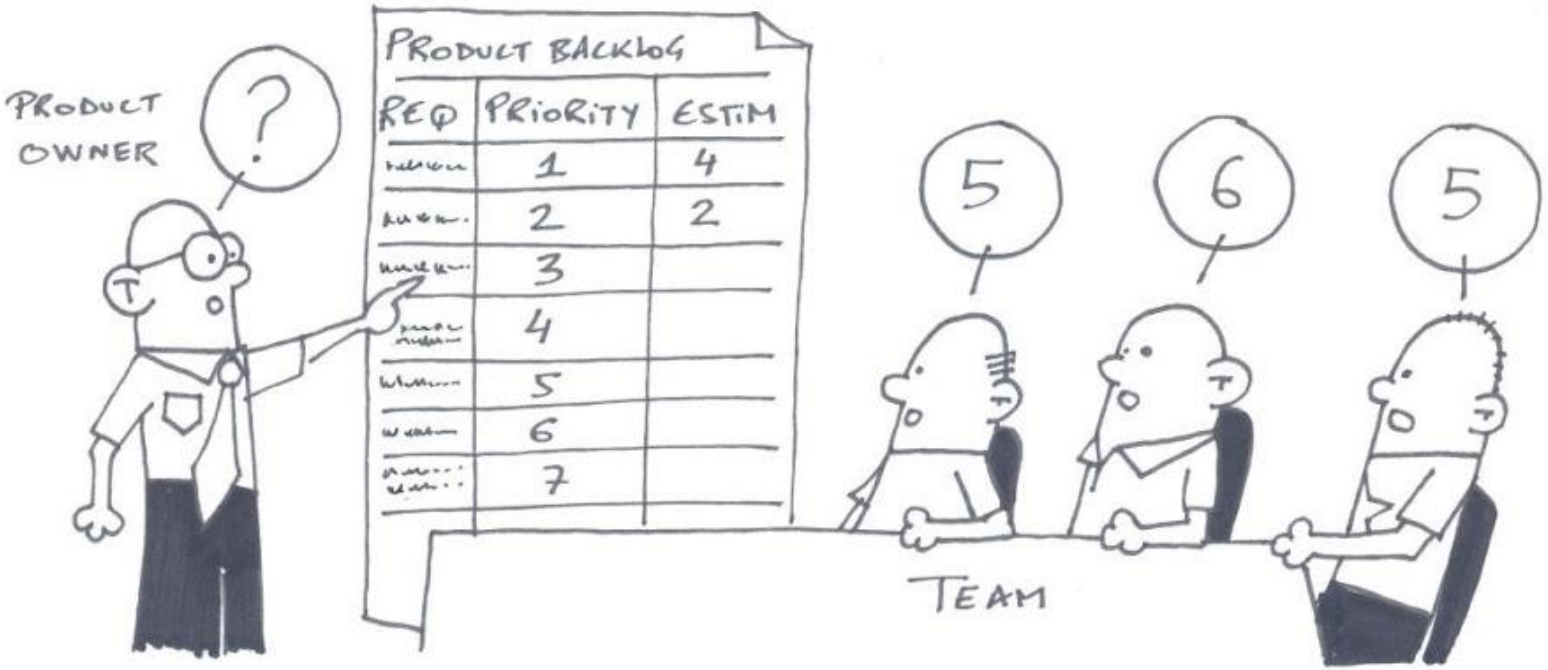
Une bonne définition doit :

1. **Être atteignable**
   - Représente de façon réaliste les capacités de l'équipe
   - Ex: s'assurer que l'équipe ait accès à toutes les ressources nécessaires
2. **Être fait de manière collaborative**
   - Tous les participants de la tâche doivent avoir la possibilité de donner leurs avis
3. **Être flexible**
   - Ouvert au changement à mesure que l'équipe trouve de meilleures façons de travailler

Product Backlog

Product Backlog - Big Picture

1. Une liste de tous les éléments nécessaires à la bonne livraison du produit.
2. Le Product Owner est responsable de la création et du suivi du bon déroulement de cette liste.
3. Tout membre de l'équipe Scrum peut ajouter des éléments à la liste en collaboration avec le Product Owner.



CONSULTING BSTORM

Product Backlog

Les qualités du Product Backlog

4 Qualités: **DEEP**

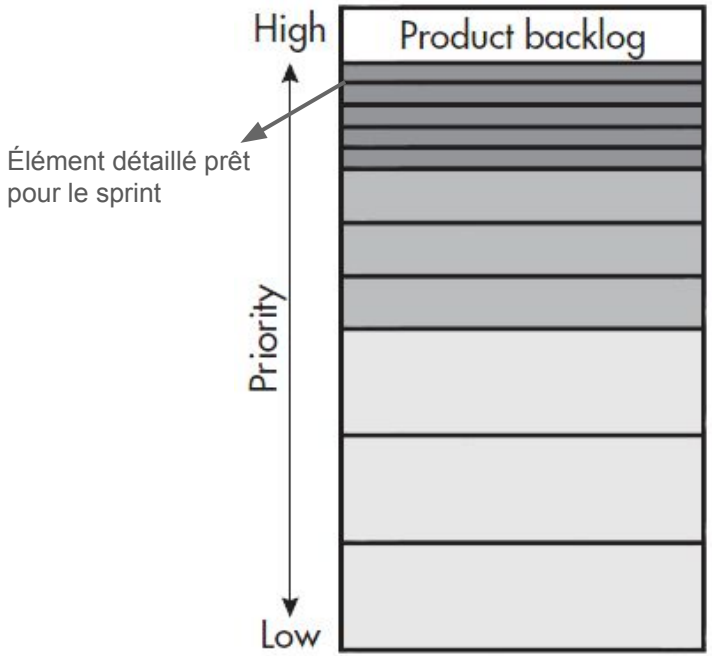
1. **D**etailed Appropriately
   - Les éléments hautement prioritaires sont décomposés et raffinés en vue de la réunion du planning de Sprint (Detailed Appropriately)
2. **E**stimated
   - Estimation des éléments via le planning poker
3. **E**mergent
   - Ajout, modification, suppression d'éléments
4. **P**rioritized
   - Prioritisation (plus important en haut)

Détaillé de manière appropriée

Les éléments les plus prioritaires sont les éléments les plus détaillés

Implique que les exigences soient décomposées et raffinées durant l'entièreté du projet

Les éléments du PB sont des **User Stories**



Emergent

Le contenu du product backlog change continuellement :

- De nouveaux éléments sont ajoutés
- Des éléments sont modifiés
- Des éléments sont supprimés

Priorisés

1. **Valeur apportée aux clients / utilisateurs**
   - *« Quel est le bénéfice financier à développer cette fonctionnalité ? »*
2. Coût de développement estimé
   - *« Quel est le coût de développement ? De maintenance ? De formation des utilisateurs ? »*
   - *« Quel est le coût d'un changement ? »*
3. Opportunité d'apprentissage pour l'équipe
   - *« L'implémentation de cette fonctionnalité nous permet-elle d'apprendre ou de développer de nouvelles compétences ? »*
   - *« Cette fonctionnalité va-t-elle améliorer la productivité de mon personnel ? »*

Triage des exigences

- **Problème**
  - Nombre important d'exigences élicitées
  - Ressources limitées
  - Impossibilité de faire tout, tout de suite
- **Solution**
  - Donner un ordre de priorité (prioritisation)
  - Différentes approches possibles
    - Approche Qualitative
      - **MoSCoW**
      - **Kano Model**
      - **Now-How-Wow Matrix**
    - Approche Quantitative
      - **Matrix Prioritization / VOLERE**

Modèle Kano

Le succès d'un produit ou service dépend de sa capacité à résoudre efficacement un ou plusieurs problèmes des clients.

Toutes ces exigences ne vont pas donner la même satisfaction aux clients.

Le modèle Kano est une représentation des besoins en 5 principales catégories en fonction de la satisfaction qu'ils procurent.

Kano Model

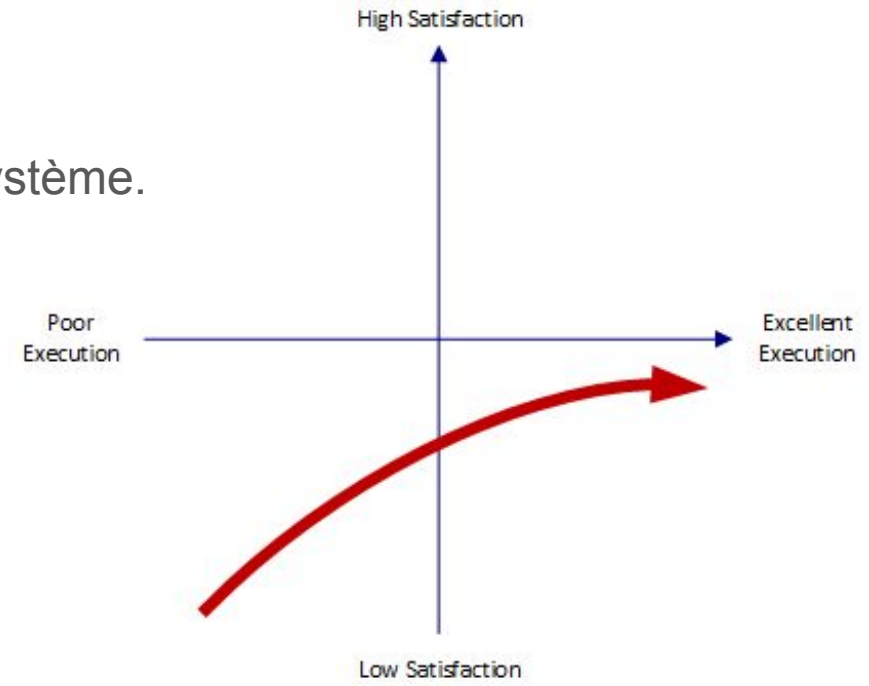
Kano a identifié cinq catégories différentes de produits basées sur la perception du client.

Author	Category 1	Category 2	Category 3	Category 4	Category 5
Kano (1984)	Must-Be	Attractive	One-Dimensional	Indifferent	Reverse
Cadotte & Turgeon (1988)	Dissatisfier	Satisfier	Critical	Neutral	
Brandt (1988)	Minimum Requirement	Value-Enhancing	Hybrid	Unimportant	
Venkitaraman and Jaworski (1993)	Flat	Value-Added	Key	Low	
Brandt and Scharioth (1998)	Basic	Attractive	One-Dimensional	Low-Impact	
Llosa (1997 and 1999)	Basic	Plus	Key	Secondary	
Pohl and Rupp (2011) [3]	Dissatisfier	Delighter	Satisfier		
KanoModel.com	Basic	Excitement	Performance	Indifferent	Reverse

Kano Model

Must-Be (Must)

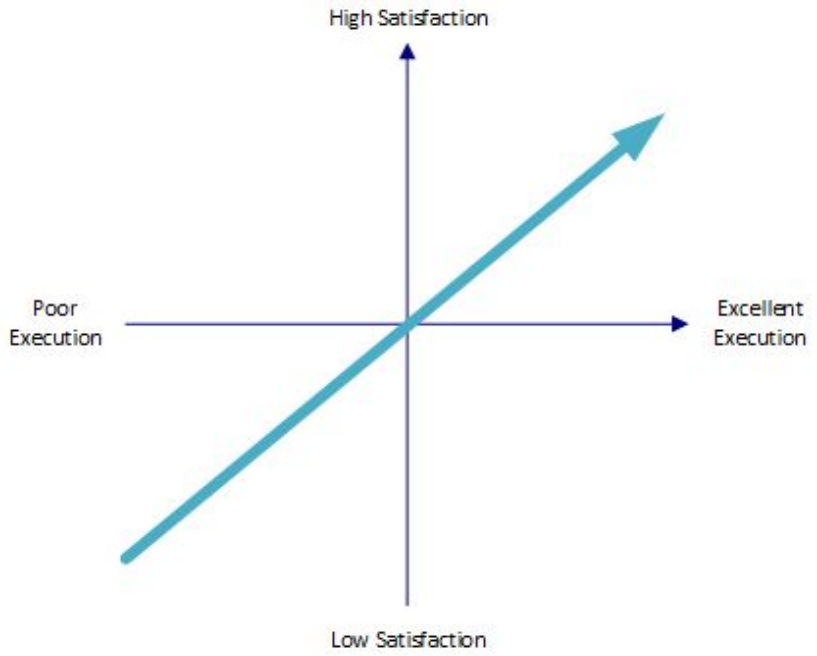
- Ce que le client prend pour acquis.
- Cost of entry pour pouvoir lancer le système.



Kano Model

One-Dimensional (Should Have)

- Ce que le client exige explicitement, et souhaite voir vérifié dans la solution finale.
- Engendre une augmentation proportionnelle de satisfaction.



Kano Model

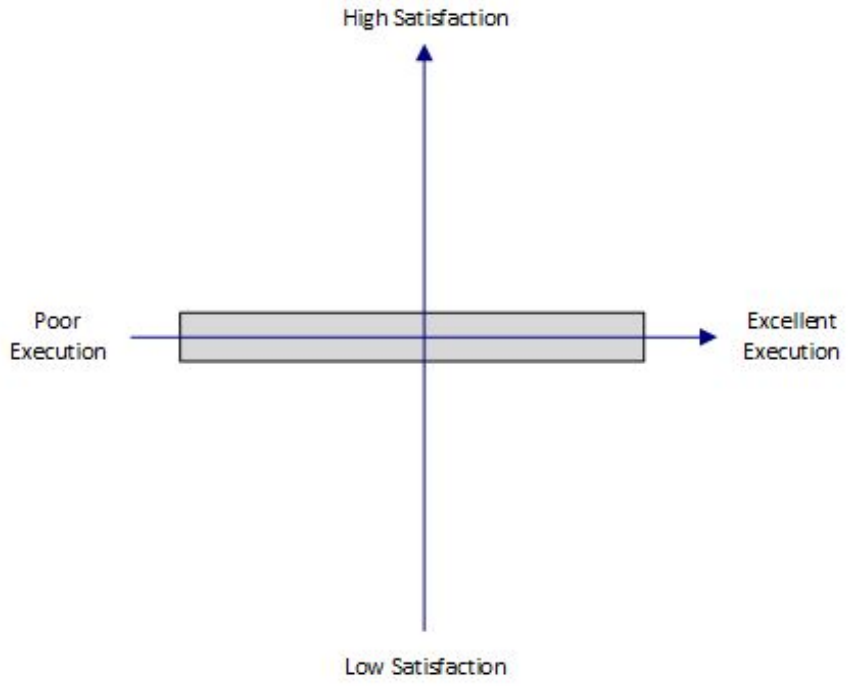
Attractive (Could Have)

- Ce que le client n'exige pas forcément, mais apprécie fortement.
- Engendre une augmentation non proportionnelle de satisfaction.

Kano Model

Indifferent

- Ce que le client n'exige pas forcément, et ne souhaite spécialement voir vérifié dans la solution finale.
- N'engendre pas d'augmentation de la satisfaction.



Kano Model

Reverse

- Ce que le client souhaite voir être évité dans la solution finale.
- Engendre une diminution proportionnelle de la satisfaction.

Modèle Kano

Fonctionnalité n

Si cette fonctionnalité était présente, que ressentiriez-vous ?	*« Ça me ferait plaisir »*	
	« Ce serait un minimum »	X
	« Je n'ai pas d'avis »	
	« Je l'accepterais »	
	« Ca me dérangerait »	
Si cette fonctionnalité n'était pas présente, que ressentiriez-vous ?	*« Ça me ferait plaisir »*	
	« Ce serait un minimum »	
	« Je n'ai pas d'avis »	
	« Je l'accepterais »	X
	« Ca me dérangerait »	

		Question dysfonctionnelle				
		Plaisir	Minimum	Neutre	Acceptation	Dérangement
Question fonctionnelle	Plaisir	?	L	L	L	E
	Minimum	R	I	I	I	O
	Neutre	R	I	I	I	O
	Acceptation	R	I	I	I	O
	Dérangement	R	R	R	R	?

O Obligatoire **R** "Reverse" (Inversée)

E Exprimée **?** Incertaine

L Latente **I** Indifférente

MoSCoW

Technique de priorisation sur base d'une échelle nominale (pas de chiffres)

- Extrêmement fréquente car simple et multi-contexte
- Elle se base sur 4 grandes catégories
  1. **Must**
  2. **Should**
  3. **Could**
  4. **Won't**

Note : Les o dans MoSCoW sont ajoutés uniquement pour rendre l'acronyme prononçable.

MoSCoW

- **Must Have**
  - Les choses que le système doit vérifier afin d'aller de l'avant.
  - Si vous demandez à un client « Quid si X est manquant? » et que la réponse est « Dans ce cas nous ne continuons pas », alors vous avez trouvé un "must ».
- Si
  - Livrer à la date cible sans X n'a aucun sens.
  - Livrer sans X n'est pas légal.
  - Livrer sans X n'est pas sur (risque).

MoSCoW

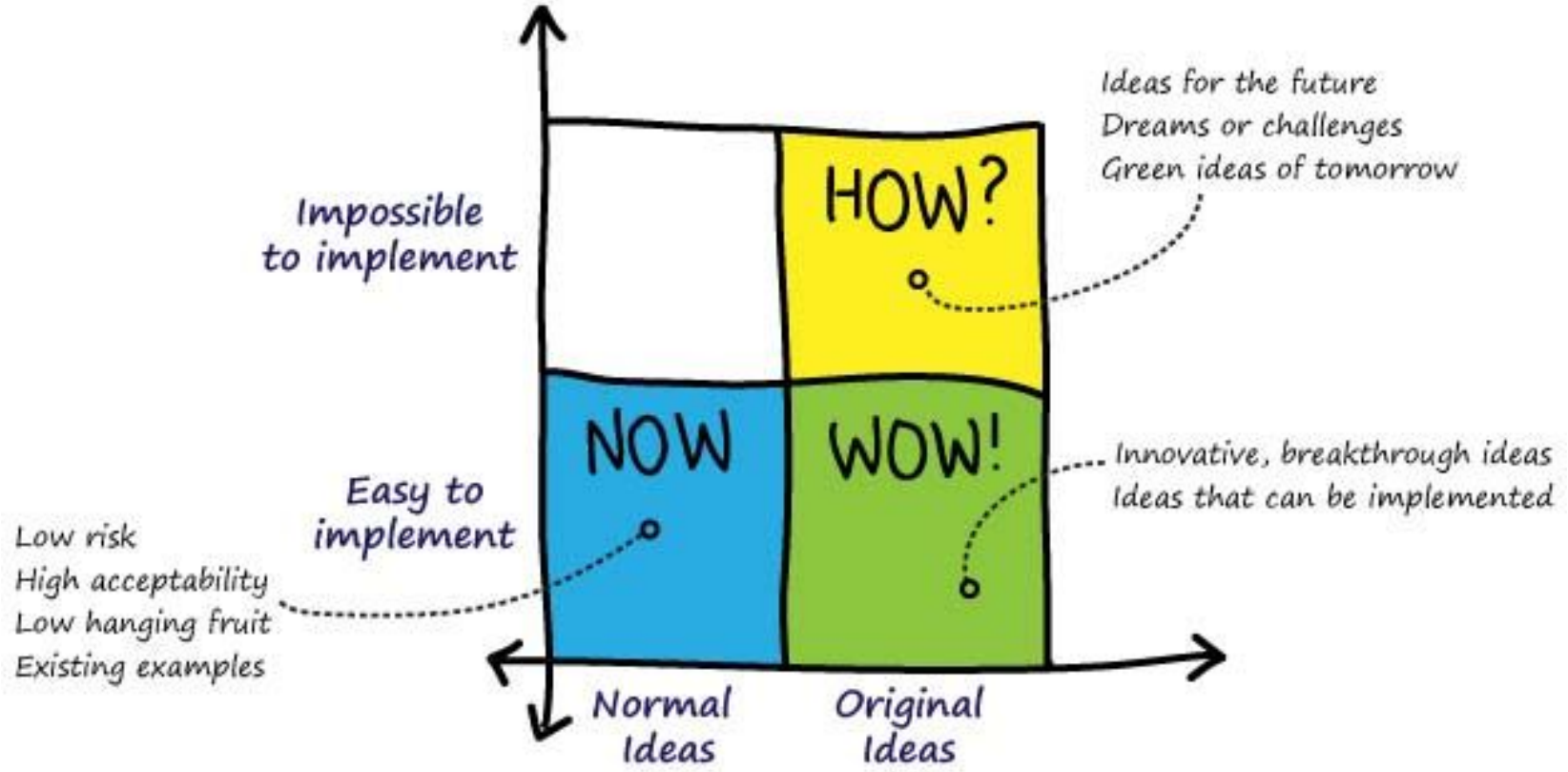
- **Should Have**
  - Diffère du Must Have dans le sens où il n'est pas absolument nécessaire pour aller de l'avant dans le projet, mais son absence entraînerait un niveau plus élevé d'insatisfaction de la part du client.
  - Important mais pas vital.
  - Peut être dommageable pour l'ensemble de laisser de côté, mais la solution est encore viable.
  - Peut nécessiter une sorte de solution de contournement, par exemple gestion des attentes, etc.

MoSCoW

- **Could Have**
  - Les éléments qui sont souhaitables, mais qui ont un impact plus faible si on les laisse sortir du scope du projet.
  - Recherché ou souhaitable, mais moins important.
  - Moins d'impact si laissé de côté (par rapport à Should).
- **Won't Have** (ou Would Have)
  - Les éléments identifiés comme étant souhaitables ou précieux, mais qui ne sont pas assez importants pour être implémentés dans le prochain release.
  - Ils ne sont pas invalides: ils sont tout simplement des éléments à ne pas prendre en compte "pour le moment ».

Now-How-Wow Matrix

- Préjugé
  - Les idées créatives sont souvent des idées qui sont difficiles à mettre en oeuvre
- Faux !
  - Certaines sont compliquées à implémenter.
  - D'autres offrent une valeur ajoutée très importante pour un coût relativement faible.
- Now-How-Wow aide à sélectionner et prioriser les idées selon leur faisabilité et leur degré d'innovation



BSTORM CONSULTING

Now-How-Wow Matrix

Volere Prioritisation Spreadsheet											
Copyright c The Atlantic Systems Guild 2002											
Requirement/Product Use Case/Feature	Number	Factor - score out of 10	%Weight applied	Factor - score out of 10	%Weight applied	Factor - score out of 10	%Weight applied	Factor - score out of 10	%Weight applied		Total Weight
		Value to Customer	40	Value to Business	20	Minimise Implementation Cost	10	Ease of Implementation	30	Priority Rating	100
Requirement 1	1	2	0.8	7	1.4	3	0.3	8	2.4	4.9	
Requirement 2	2	2	0.8	8	1.6	5	0.5	7	2.1	5	
Requirement 3	3	7	2.8	3	0.6	7	0.7	4	1.2	5.3	
Requirement 4	4	6	2.4	8	1.6	3	0.3	5	1.5	5.8	
Requirement 5	5	5	2	5	1	1	0.1	3	0.9	4	
Requirement 6	6	9	4	6	1.2	6	0.6	5	1.5	6.9	
Requirement 7	7	4	2	3	0.6	6	0.6	7	2.1	4.9	
Requirement											

Priorisés

1. Valeur apportée aux clients / utilisateurs
   - *« Quel est le bénéfice financier à développer cette fonctionnalité ? »*
2. **Coût de développement estimé**
   - *« Quel est le coût de développement ? De maintenance ? De formation des utilisateurs ? »*
   - *« Quel est le coût d'un changement ? »*
3. Opportunité d'apprentissage pour l'équipe
   - *« L'implémentation de cette fonctionnalité nous permet-elle d'apprendre ou de développer de nouvelles compétences ? »*
   - *« Cette fonctionnalité va-t-elle améliorer la productivité de mon personnel ? »*

Priorisés

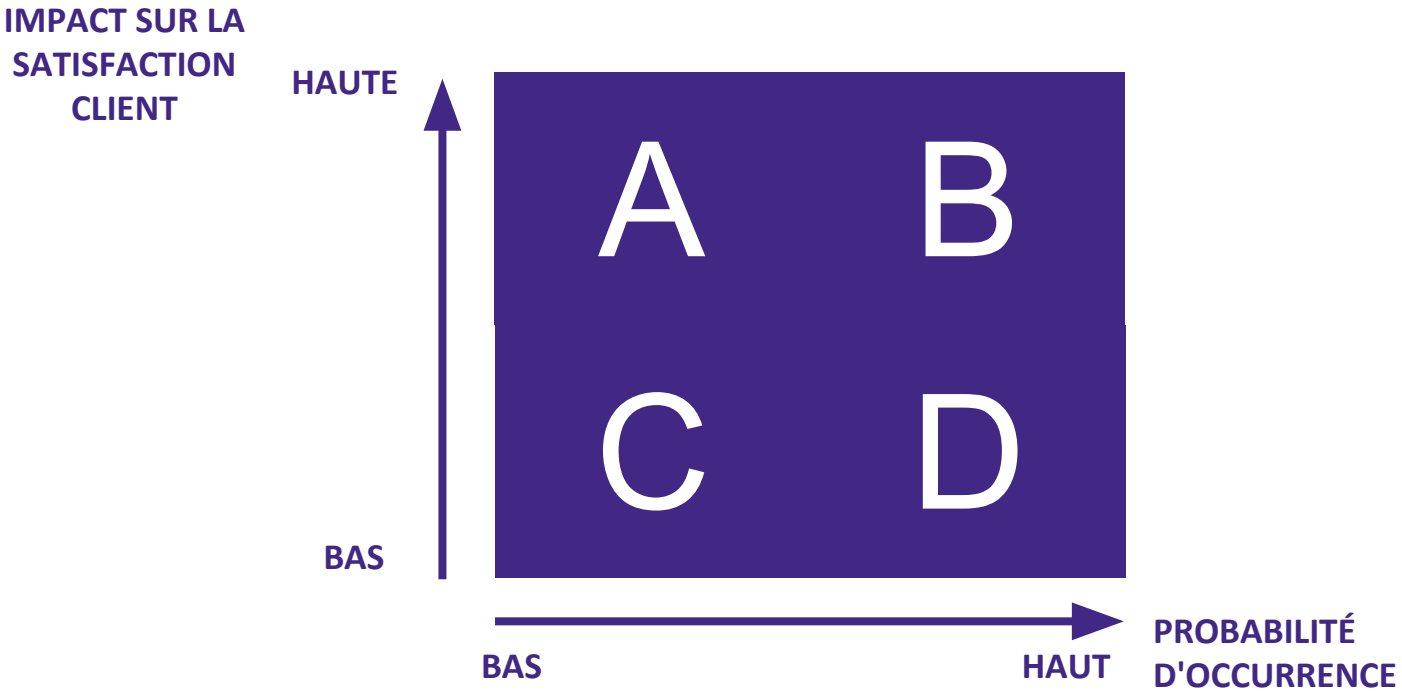
1. Valeur apportée aux clients / utilisateurs
   - *« Quel est le bénéfice financier à développer cette fonctionnalité ? »*
2. Coût de développement estimé
   - *« Quel est le coût de développement ? De maintenance ? De formation des utilisateurs ? »*
   - *« Quel est le coût d'un changement ? »*
3. **Opportunité d'apprentissage pour l'équipe**
   - *« L'implémentation de cette fonctionnalité nous permet-elle d'apprendre ou de développer de nouvelles compétences ? »*
   - *« Cette fonctionnalité va-t-elle améliorer la productivité de mon personnel ? »*

Risque de développement

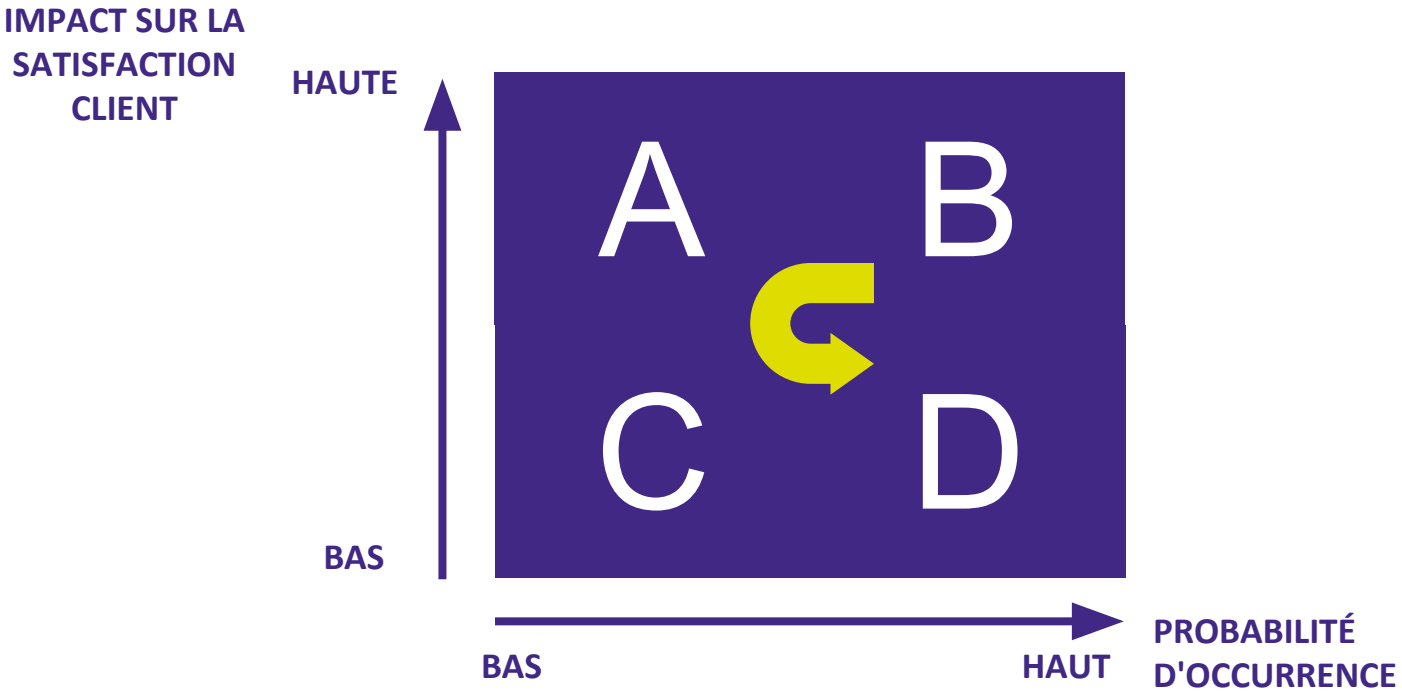
Le risque de développement

- *« Cette fonctionnalité nous expose-t-elle à davantage de risques ? Ou, au contraire, nous permet-elle de nous confronter au risque ? »*
- *« Cette fonctionnalité va-t-elle impacter les processus métier ? »*
- *« Cette fonctionnalité va-t-elle susciter de la frustration ou de la résistance ? »*
- *« Quel est le préjudice si cette fonctionnalité n'est pas implémentée ? »*

Analyse du risque



Analyse du risque



Exigences non-fonctionnelles

Ex.: performance, robustesse, utilisability...

Influence:

- Interface utilisateur
- Architecture
- Choix technologiques
- ...

Exigences non-fonctionnelles

Description sous forme de contraintes

- Ex.: Le système doit pouvoir répondre à n'importe quelle requête en moins d'une seconde

Distinction entre les exigences globales et locales

- Globales: doivent être définie avec la vision du produit
- Locales: peut être définie et attachée avec la user story affectée

Etendue du Product Backlog dans de larges projets

Utilisé un seul Product Backlog

- Permet d'éviter les problèmes de synchronisation

Préparer les éléments pour les 2 à 3 Sprints suivants

- Les nombres d'éléments détaillés est donc plus important

Proposer différentes vues pour les différentes teams SCRUM

Estimer l’effort et planifier

Eléments Estimés

Connaître la charge de travail aide à donner des priorités aux éléments.

Estimation grossière exprimée en nombre **Story Points**.

- Ne correspond pas à une unité de temps précise mais à une unité relative.
  - On compare le “temps” d’une tâche par rapport à une autre.
  - Cette tâche correspond à 2 ou 3 points ? On compare avec les autres tâches dont on a attribué 2 ou 3 points.
- Les estimations sont réalisées par les membres de la Scrum Team.
- Le Product Owner et le Scrum Master ne doivent ni participer, ni influencer l'estimation.

CURSE

L’un des principaux défis pour les équipes qui adoptent Scrum ou une autre méthodologie Agile est le concept de Story Point.

Généralement les équipes commencent à utiliser assimiler ces Points à une journée, donc 7 Points une semaine, etc.

On se rend très vite compte que c’est incomplet et trompeur.

CURSE

CURSE est un acronyme pour

- **C** pour la complexité.
  - C'est la complexité, la complexité, l'implication, l'intégration, etc. du travail d'une histoire.
- **U** pour l'incertitude ("*Uncertainty*") du travail à effectuer.
  - L'équipe peut ne pas bien connaître la technologie ou ne pas savoir comment un élément particulier de l'histoire peut avoir un impact sur l'architecture ou la conception.
- **R** est pour le risque.
  - Cela englobe le degré de dangerosité d'un changement ou d'une amélioration demandé pour l'ensemble du système, de la conception ou de l'architecture.

CURSE

CURSE est un acronyme pour

- **S** pour l'ampleur ("Scope").
  - La portée est la quantité de travail ou l'ampleur des changements particuliers.
- **E** pour l'effort.
  - C'est ce que nous associons généralement aux Story Points. Cela représente le travail réel qui va être fait, la mise en œuvre de la user story.

Suite de Fibonacci

Les Story Points sont des nombres de la Suite de Fibonacci

- Suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent.
- 0, 1, 1, 2, 3, 5, 8, 13, 21, etc.
- Pourquoi ?
  - Parce que au plus une tâche va prendre du temps, au moins on va être précis dans son estimation de temps.

Planning Poker

Planning Poker

- Objectif
  - Réussir à générer des estimations acceptables le plus rapidement possible

Planning Poker

1. Le Product Owner présente la User Story
   - Pas technique - uniquement fonctionnelle
2. La Scrum Team peut demander des éclaircissements
3. La Scrum Team estime et vote pour un nombre de Story Point
4. En cas de désaccord, on demande aux plus basses valeurs et plus hautes valeurs de justifier leur choix.
   - Objectif : lancer la discussion
   - Chaque discussion est dans une time-box. Toutes les x minutes, on revote jusqu'à ce qu'on ait un accord.
5. A la fin les éléments de même taille sont groupés ensemble pour valider leur valeur relative (triangulation)

Planning Poker

Planning Poker

Quelques bonnes pratiques:

1. Time-box : spécifier un intervalle de temps par tour de table (ex.: 2min max) et le chronométrer
2. Éviter d'influencer l'estimation avec des "Je pense que ce sera vite fait" ou "Ca risque de prendre des semaines… si pas des mois"
3. Ne jamais prononcer son estimation tant que tous les autres membres n'ont pas estimé
4. Toujours obtenir un consensus plutôt qu'une moyenne
5. Ceux qui votent sont ceux qui feront le boulot (le Product Owner ne vote pas)

Les Releases

La problématique de la planification

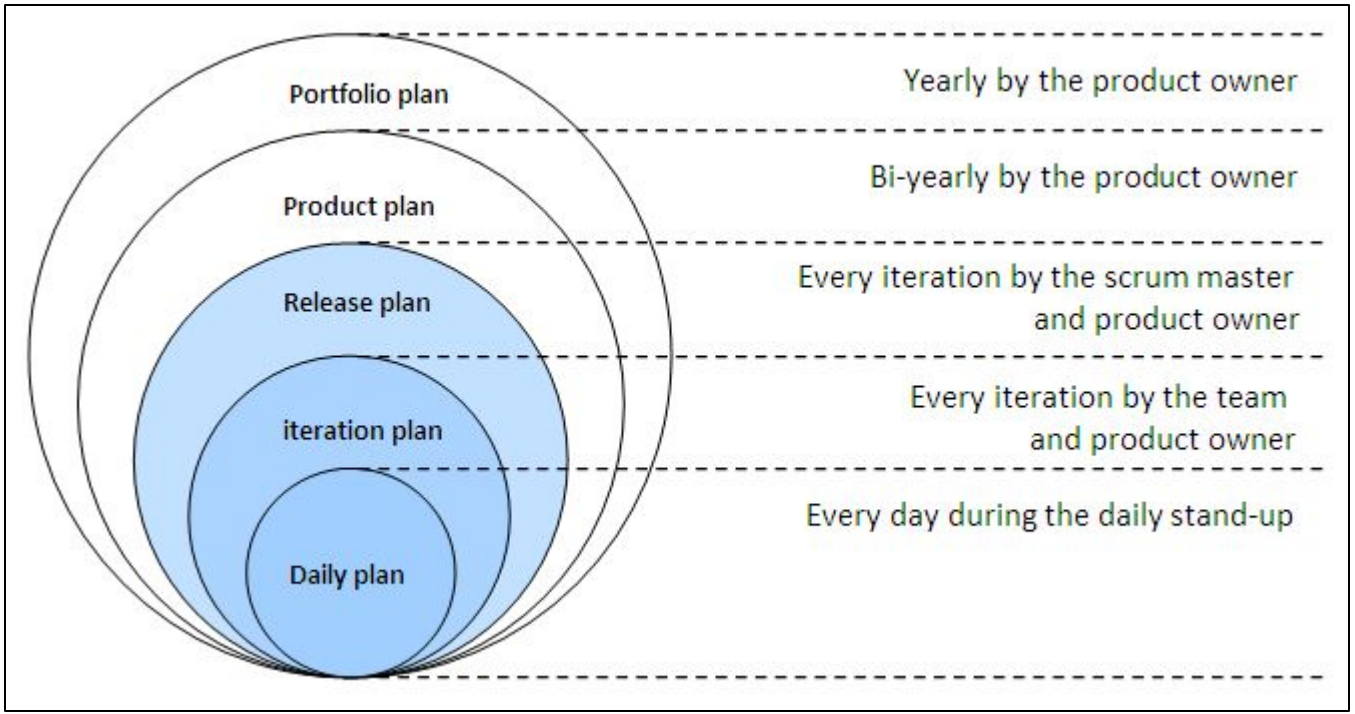
Pourquoi planifier :

1. Outil d'aide à la décision.
2. Outil de pilotage.
3. Support de communication.

5 niveaux de planification

1. **Etablissement de la vision**
   - Livraison finale
2. **Etablissement d'une roadmap**
   - Les différentes releases
3. **Planification d'une release**
   - Les différents Sprint aboutissant à la release
4. **Planification d'un SPRINT**
   - Les différentes user stories à réaliser durant cette itération
5. **SCRUM**
   - Réajustement du planning

Planning Onion



Release Plan

Major Feature	Sprint 1	Sprint 2	Sprint 3	Sprint 4 to 6	Quarter 2	Quarter 3	Year 2
Authen-tication	Login Screen		Security Questions		SSO Integration with Partner Sites		
	SSL Encryption						
Master	Product Master						RFID Implemen-tation
	Rate Master						
Order Entry		Product Selection				Recom-mended Products	Sponsored Look-ups
		Product Preview	Product Comparison				
				Product Review			
Checkout		Checkout		Coupons			Reward Points
				Order Track			

Durée des sprints

Entre 1 semaine et 4 semaines.

- Sprint court
  - Plus facile.
  - Réduction des risques...
  - … mais plus de réunion.
- Sprint long
  - Plus difficile.
  - Plus de risques...
  - … mais moins de réunion.

Estimer les durées

Principe de **vélocité** (v)

- La somme des story points (quantité de travail) qu'une équipe est capable de développer durant une itération
- Valeur hypothétique au début du projet et s'affine au fur et à mesure des itérations

Estimation durée du projet

- Après une première itération, on peut estimer une durée réaliste du projet
- Durée du projet = Somme Story Point / Vélocité

Les releases

Trois variables de décision à considérer pour planifier une release

1. **Temps**
   - *La date de lancement est-elle fixe ?*
2. **Coûts**
   - *Le budget de développement est-il fixe ?*
3. **Fonctionnalités**
   - *Un certain niveau de fonctionnalité doit-il être assuré pour une release ?*

Fixer les trois variables est impossible.

Fixer une date de lancement

- Déterminer une fenêtre d'opportunité.
- Déterminer une "*timebox*" identique pour toutes les releases.
  - Améliore l'innovation.
  - Améliore les estimations (coûts, planning...).
  - La détermination du budget est direct si la composition de l'équipe reste stable.
    - (Hypothèse: travail est coût principal).

Fixer un prix (enveloppe budgétaire)

Arrêt du développement lorsque :

1. Toutes les fonctionnalités sont implémentées.
2. Le budget est consommé.

Fixer les fonctionnalités

Ne prend pas en compte l'émergence des exigences.

Limite la capacité d'adaptation de l'équipe aux besoins du client.

Fixer un prix et un ensemble de fonctionnalités

- A éviter.
- Séparer le contrat en deux projets consécutifs.
  a. Créer la vision du produit et partiellement l'implémenter en 2 ou 3 sprints.
  b. Créer un planning de releases sur la base du Product Backlog avec un budget réaliste.

Burndown Chart

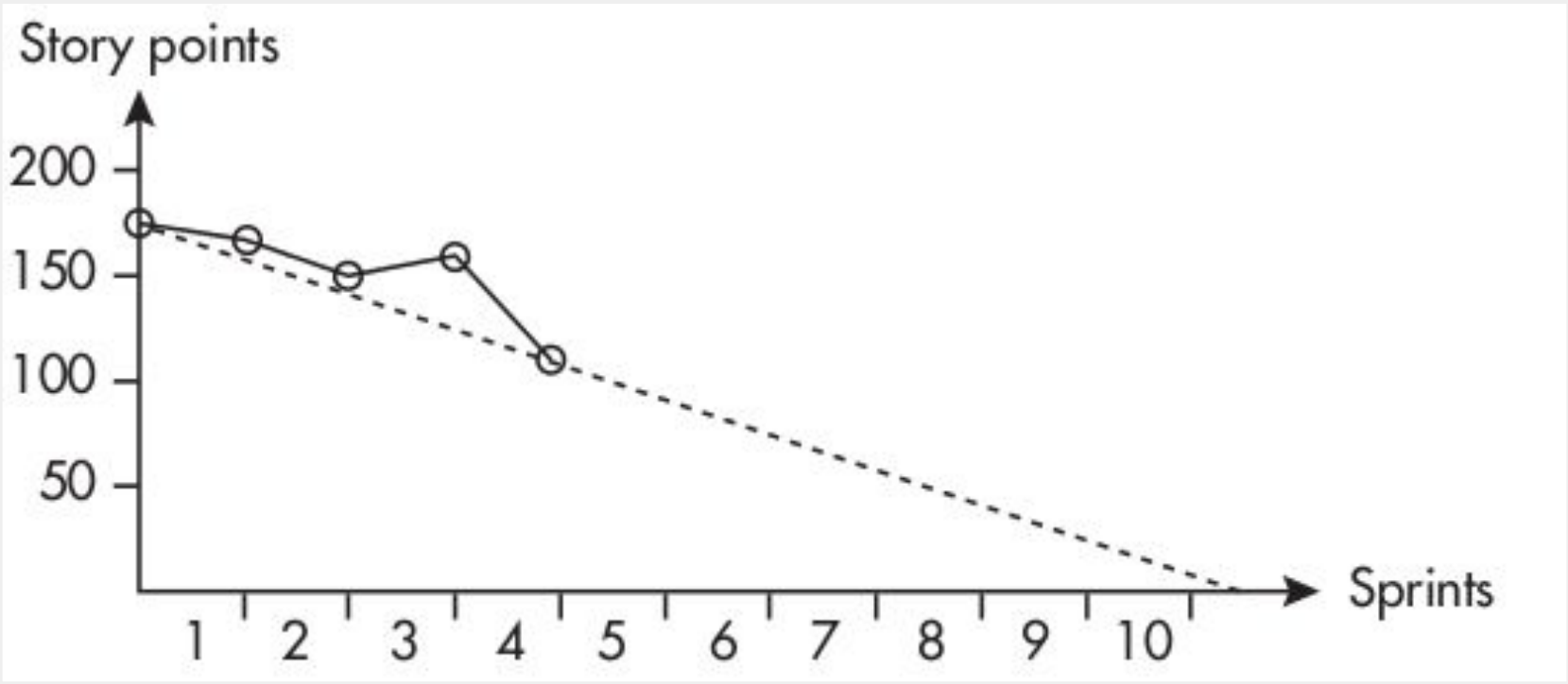
Représentation graphique de l'évolution de quantité de travail restante par rapport au temps sur une période de temps donnée.

- **Axe verticale** : quantité de travail restant (en Story Point)
- **Axe horizontale** : temps restant (en nombre de Sprint)

On peut faire un Burndown Chart par :

- Projet.
- Sprint.

Burndown Chart



Burndown Chart