SCRUM


Synthex
UML 2
Pratique de la modélisation
3e édition
Benoit CHARROUX
Aomar OSMANI
Yann THIERRY-MIEG
PEARSON

Introduction aux méthodes agiles

Les méthodes agiles

- Les méthodes agiles sont apparues dans les années 1990 (Extreme Programming, Rapid Application Development, Scrum…) :
    - capacité à réagir au changement plutôt que de suivre un plan
    - collaboration accrue avec le client plutôt que de suivre les termes d'un contrat
    - livraison d'un logiciel fonctionnel est à privilégier plutôt que de rédiger une documentation complète
    - les personnes et leurs interactions priment sur l'utilisation d'outils ou le suivi des processus.

La manifeste agile 1/2

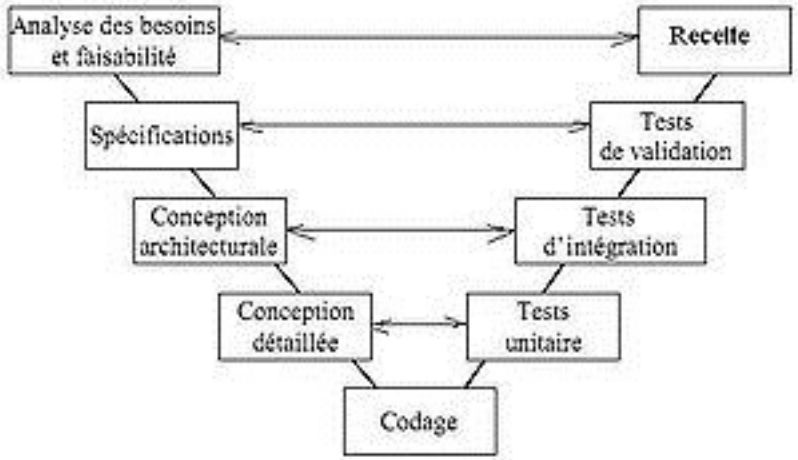
- Écrit en 2001 par 17 personnes : http://agilemanifesto.org

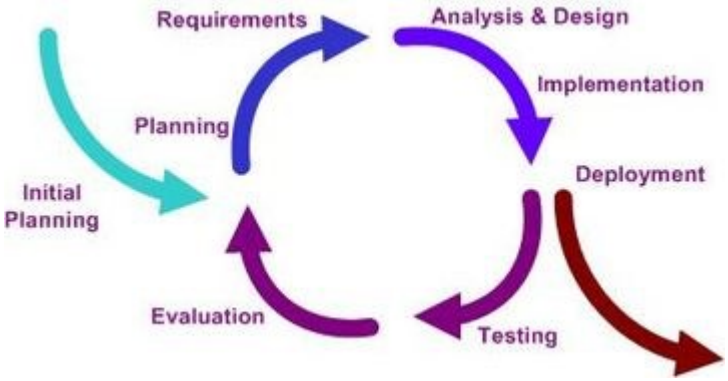
  - « Notre première priorité est de satisfaire le client en livrant tôt et régulièrement des logiciels utiles ».
  - « Livrer fréquemment une application fonctionnelle, toutes les deux semaines à deux mois, avec une tendance pour la période la plus courte ».
  - « Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet ».
  - « Les gens de l'art et les développeurs doivent collaborer quotidiennement au projet ».
  - « Une attention continue à l'excellence technique et à la qualité de la conception améliore l'agilité ».
  - « Le changement est bienvenu, même tardivement dans le développement. Les processus agiles exploitent le changement comme avantage compétitif pour le client ».

La manifeste agile 2/2

- « Bâtissez le projet autour de personnes motivées. Donnez leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail ».
- « La méthode la plus efficace pour transmettre l'information est une conversation en face à face ».
- « Les processus agiles promeuvent un rythme de développement soutenable. Commanditaires, développeurs et utilisateurs devraient pouvoir maintenir le rythme indéfiniment ».
- « La simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle ».
- « Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto-organisent ».
- « À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens ».

Les cycles de développement





Sources : wikipedia

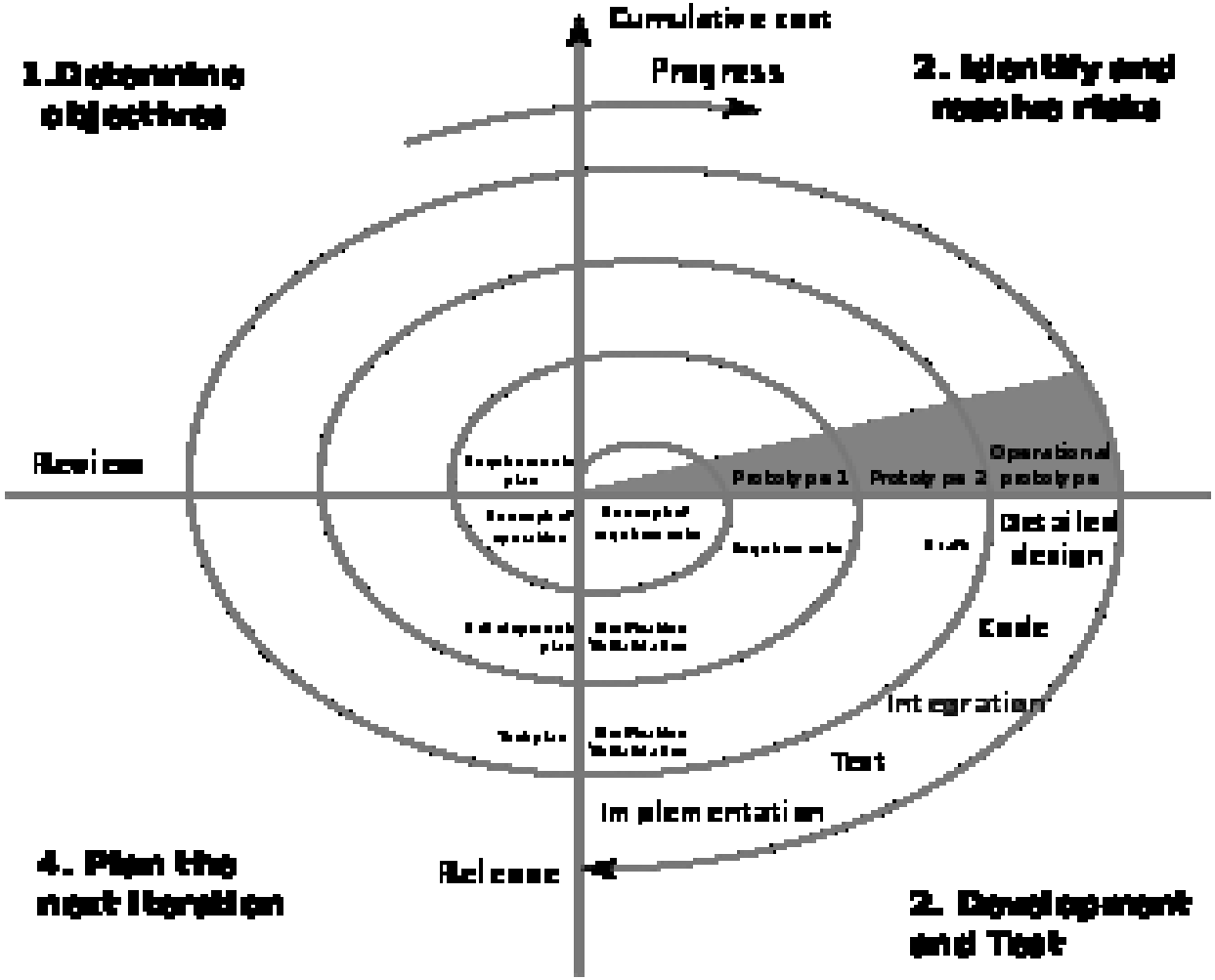
Cycle en V : le logiciel est entièrement spécifié avant d'être réalisé.

Inconvénient : l'expérience montre qu'il est très difficile de tout spécifier avant de coder.

Itératif : on ne spécifie que ce qui va être développé dans la prochaine itération.

Inconvénient : il faut parfois retoucher le code existant => refactoring du code.

Les cycles de développement



Itératif et incrémental : on réalise une série de prototypes.

Présentation de Scrum

Scrum

- **Scrum** signifie mêlée au rugby.
- Méthode agile itérative et incrémentale apparue dans les années 1990.
- Les points forts de Scrum :
    - chaque itération aboutit à un sous-ensemble des fonctionnalités du produit potentiellement livrables.
    - Le client intervient activement dans le projet pour définir l'ordre des fonctionnalités à implanter, valider les prototypes, voire modifier ses exigences au cours du développement.

Les rôles dans Scrum

- Le **directeur de produit** (*Product Owner*) est un membre de l' équipe, il est le représentant des clients et utilisateurs. C'est lui qui définit l'**ordre** dans lequel les fonctionnalités seront développées.

- L'**équipe** ne comporte pas de rôles prédéfinis ni de hiérarchie (il n'y a pas de chef de projet), elle est **auto-gérée.**

- Le **facilitateur / animateur** (*ScrumMaster*) est un membre de l' équipe qui doit veiller à ce que les valeurs de Scrum soient appliquées.

- Les **intervenants** (*Stakeholders*) sont les personnes qui souhaitent avoir une vue sur le projet sans réellement s'investir dedans. Il peut s'agir par exemple d'experts techniques ou d'agents de direction.

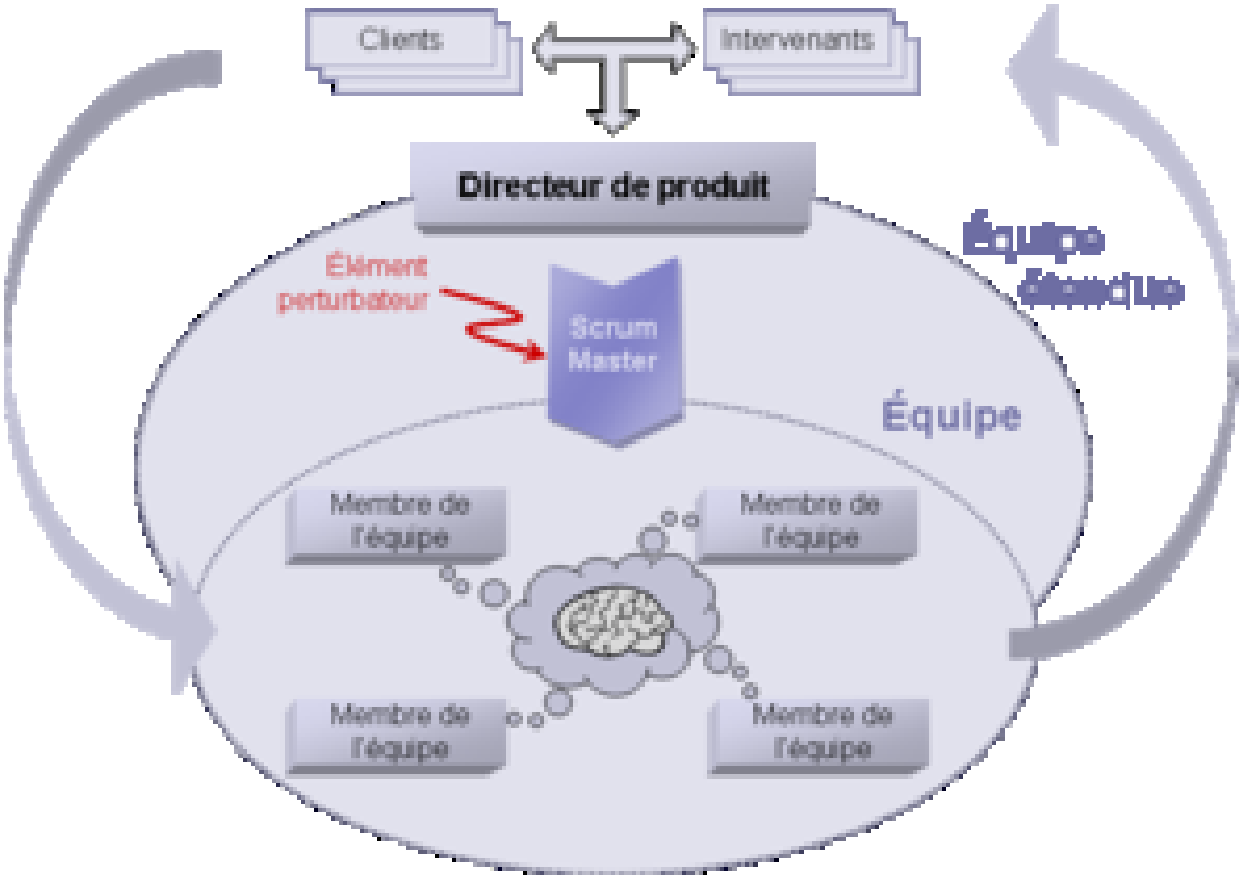
Source : wikipedia

Le product owner

- Il est responsable du backlog du produit.
- Ses rôles :
  - Définir avec le clients et les utilisateurs le contenu du produit
  - Fait partager sa vision du produit avec l'équipe
  - Définit l'ordre dans lequel les parties du produits (releases et stories) sont réalisées.
- Ces qualités :
  - Bonne connaissance du domaine métier.
  - Maitrise des techniques des user stories ou des use cases.
  - Capacité à prendre des décisions rapide.
  - Sait communiquer mais prendre des décisions.

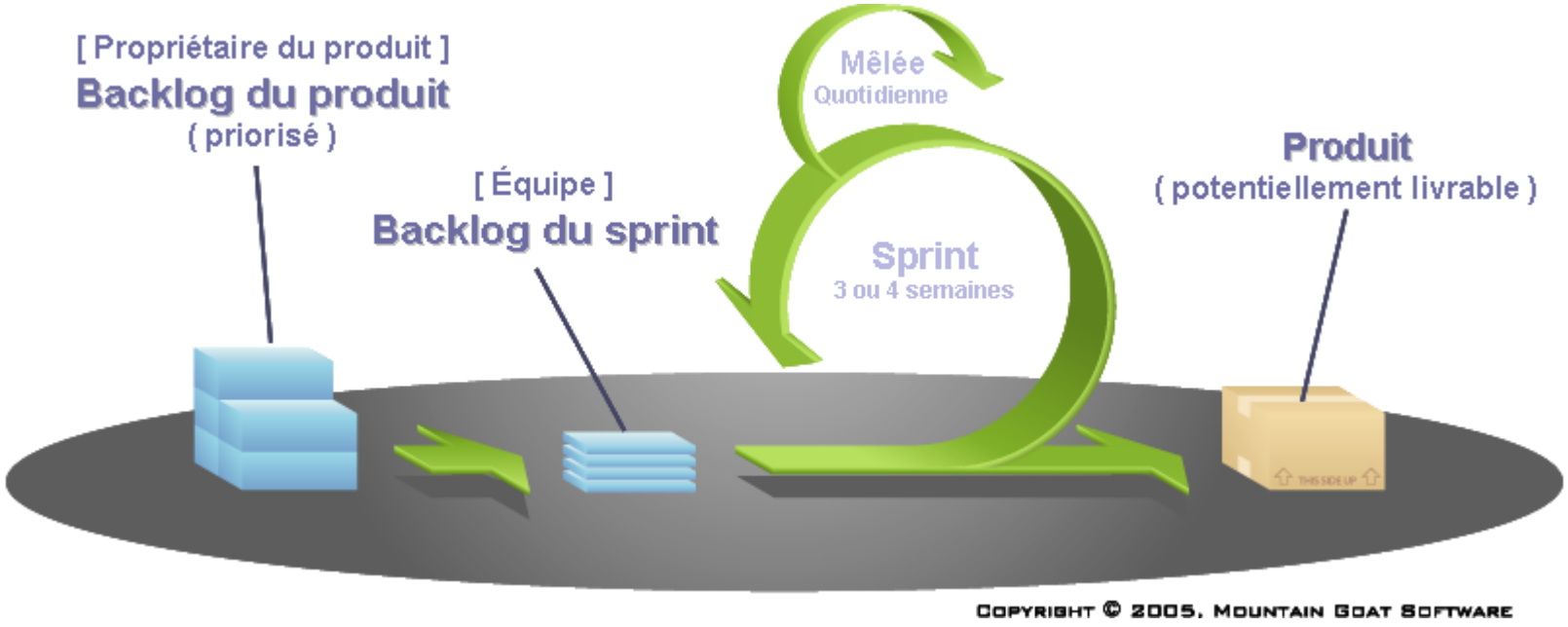
Source : wikipedia

L'interaction entre les acteurs

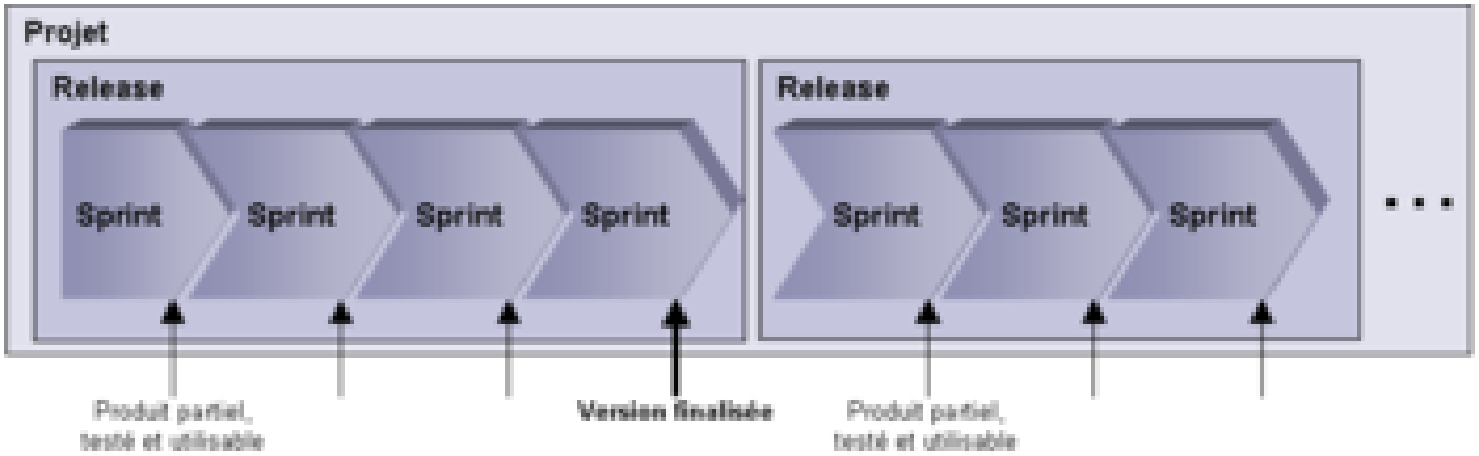


Les Sprints

- 1 itération = 1 Sprint



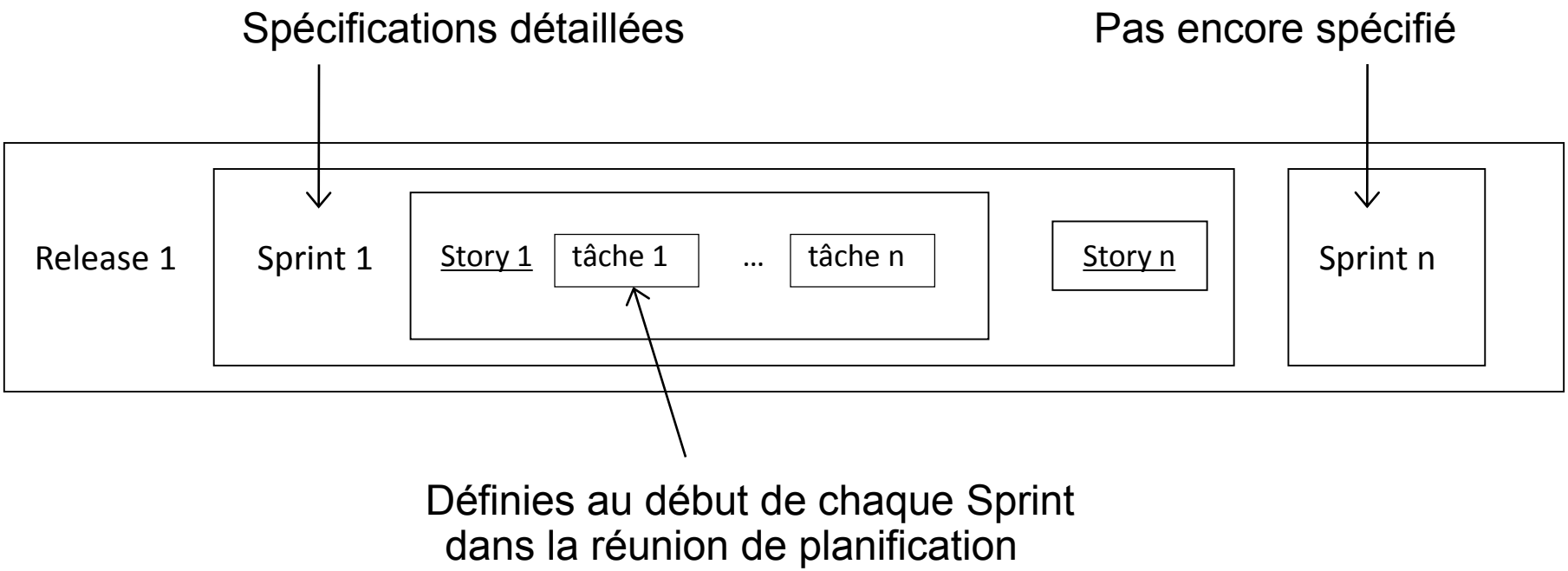
Les releases



- Attention ! Une nouvelle release n'est pas une nouvelle version de fonctionnalités déjà implantées.

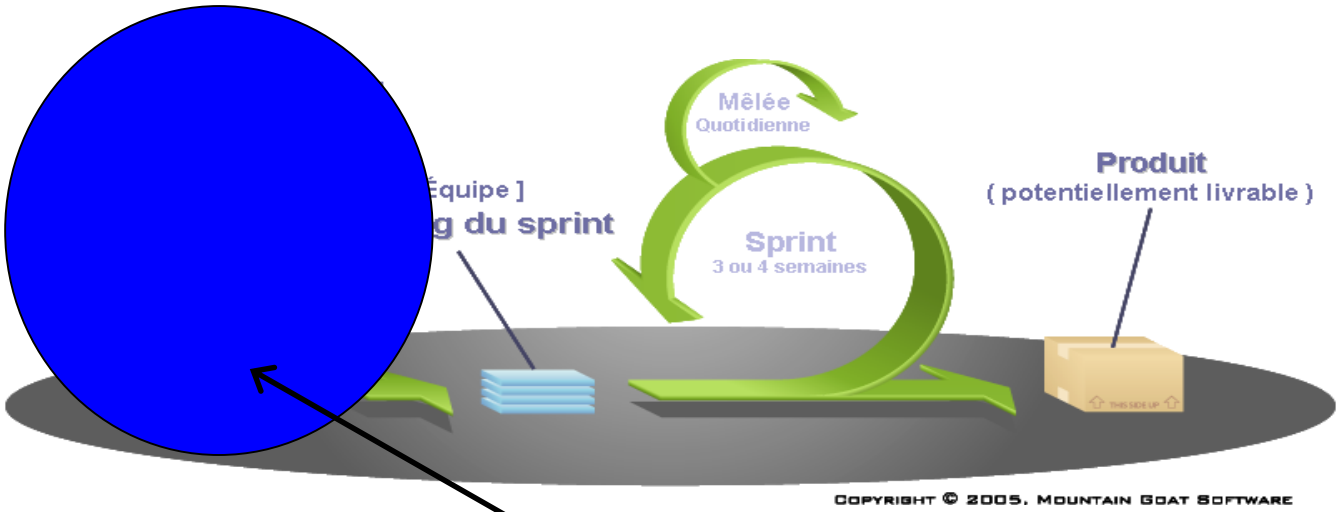
La grande question !

- Qu'est-ce qui différencie Scrum (et les méthodes agiles en générale) des méthodes traditionnelles ?



Constitution du backlog du produit

Phase de préparation des itérations



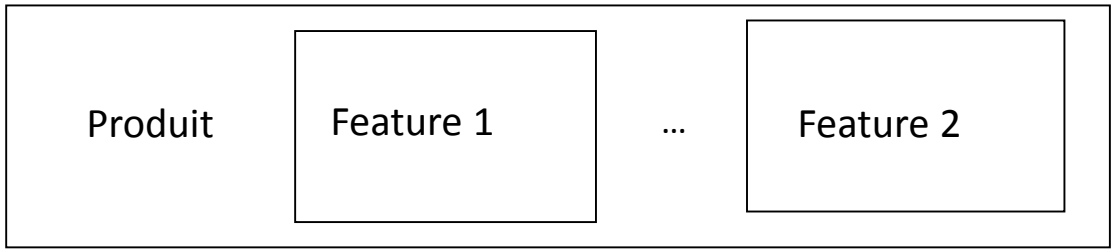
Produit	Release1	Story 11 … Story 1m	Release2	Story 21 … Story 2m

Exemple d'un backlog de produit (avec IceScrum)

Première version du backlog

- Une **user story** est une représentation d'un besoin formulée en phrases courtes dans le language de l'utilisateur.
- Le backlog du produit est composé de **stories** :
    - **User stories**.
    - **Stories techniques** (voir plus loin).

- Il est parfois difficile de trouver d'emblée les stories.
- On peut alors créer une première version du backlog composée de **features**.
- Un **feature** est un service fournit à un utilisateur et qui répond à un besoin (semblable à un **use case** d'UML).



User Story

- Une **user story** est composée d'une ou de plusieurs phrases écrite dans le language de tous les jours (ou dans le language du métier de l'utilisatuer) qui capture ce qu'un utilisateur fait ou ce dont il a besoin pour faire une partie de son travail.

- Exemple d'une **user story** appliquée à une station service :
  - « En tant que pompiste je veux armer une pompe afin de permettre à un client de se servir de l'essence ».
  - Afin de est important car cela permet de définir les conditions de satisfaction qui vont devenir des tests de validation de la **user storie**.

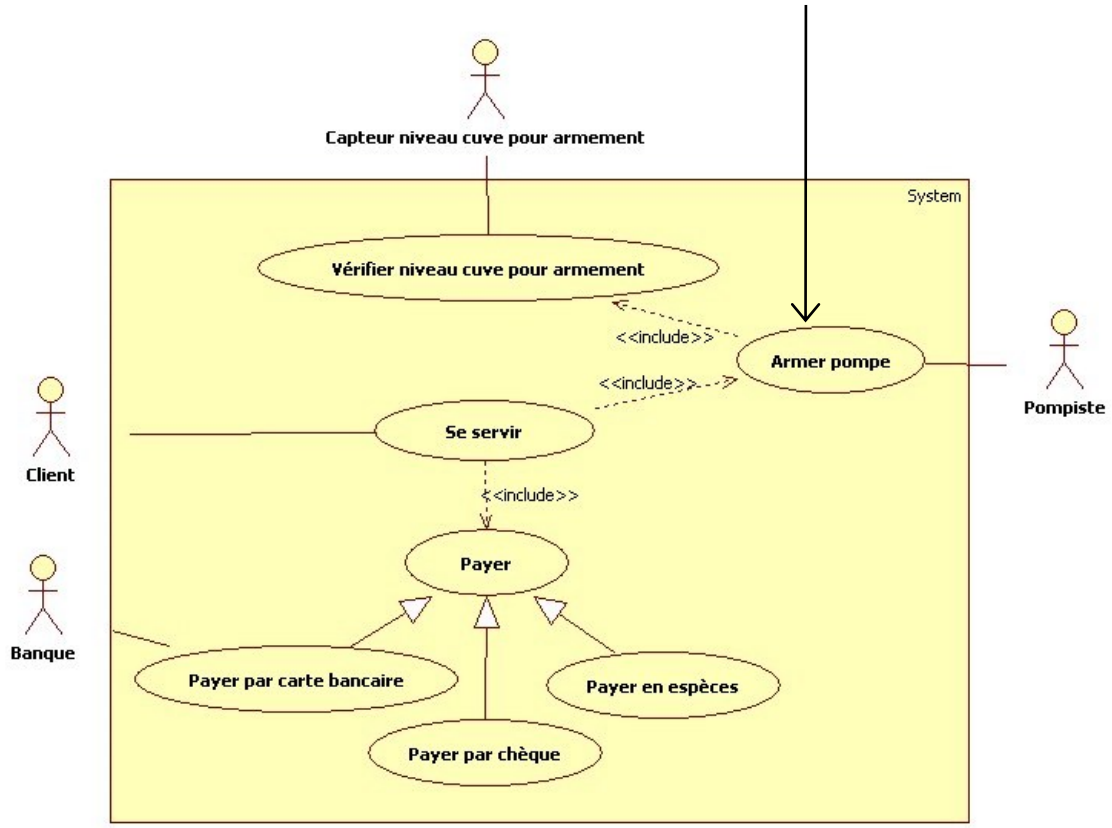
Les conditions de satisfaction d'une User Story

- Soit la **user story** :
  - « En tant que pompiste je veux armer une pompe afin de permettre à un client de se servir de l'essence ».

- Les **user stories** peuvent être complétées avec des conditions de satisfaction. Pour l'armement de la pompe, de telles conditions pourraient être :
  - Vérifier que la pompe est armée et est prête à l'emploi
  - Vérifier que si le pompiste n'arme pas la pompe, celle-ci est inutilisable
  - Vérifier que seule la pompe correspondant au type d'essence sélectionnée par le client est armée

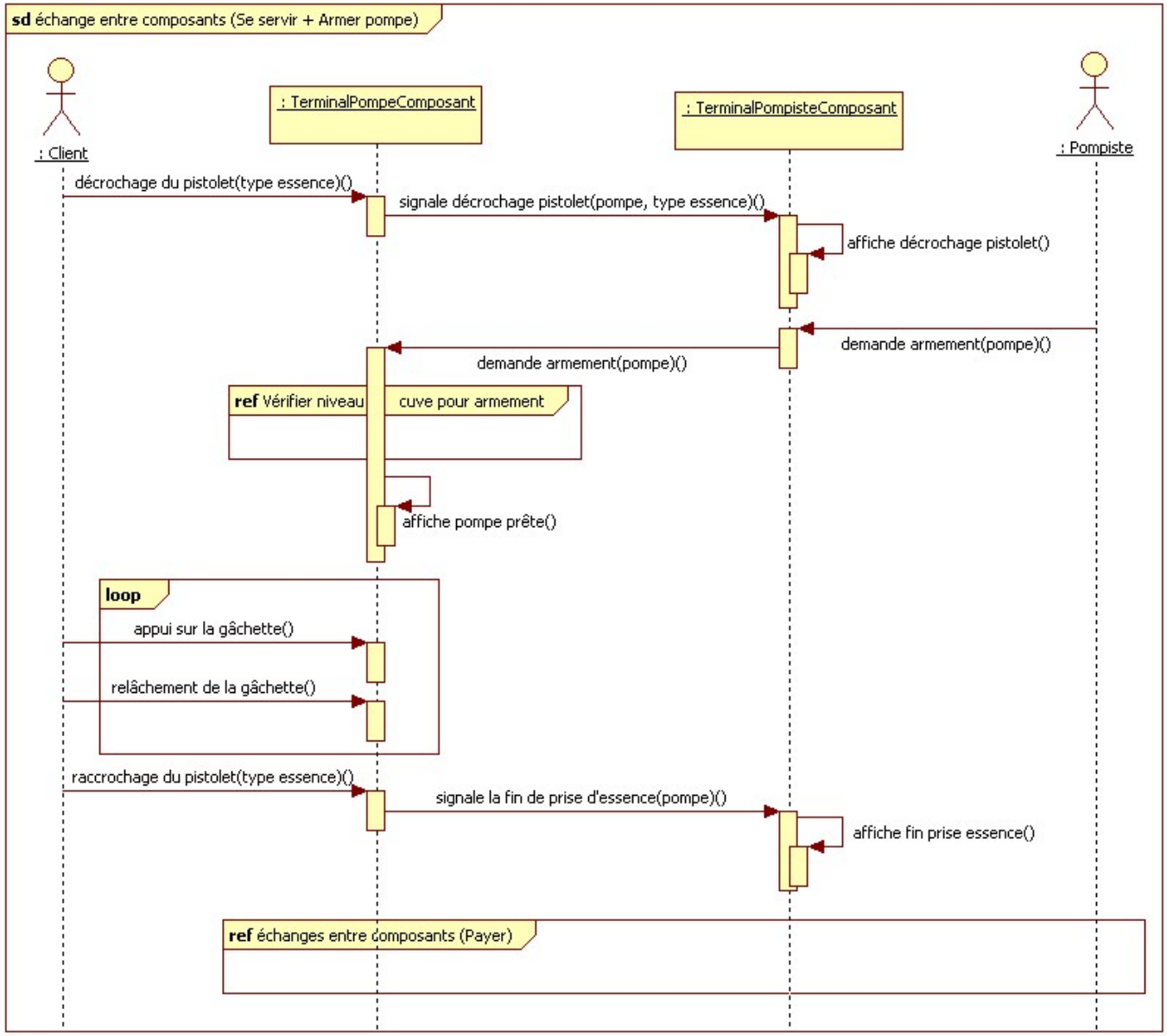
User Story versus Use Case d'UML

- Avec **UML** l'armement de la pompe est représenté par un **use case** :



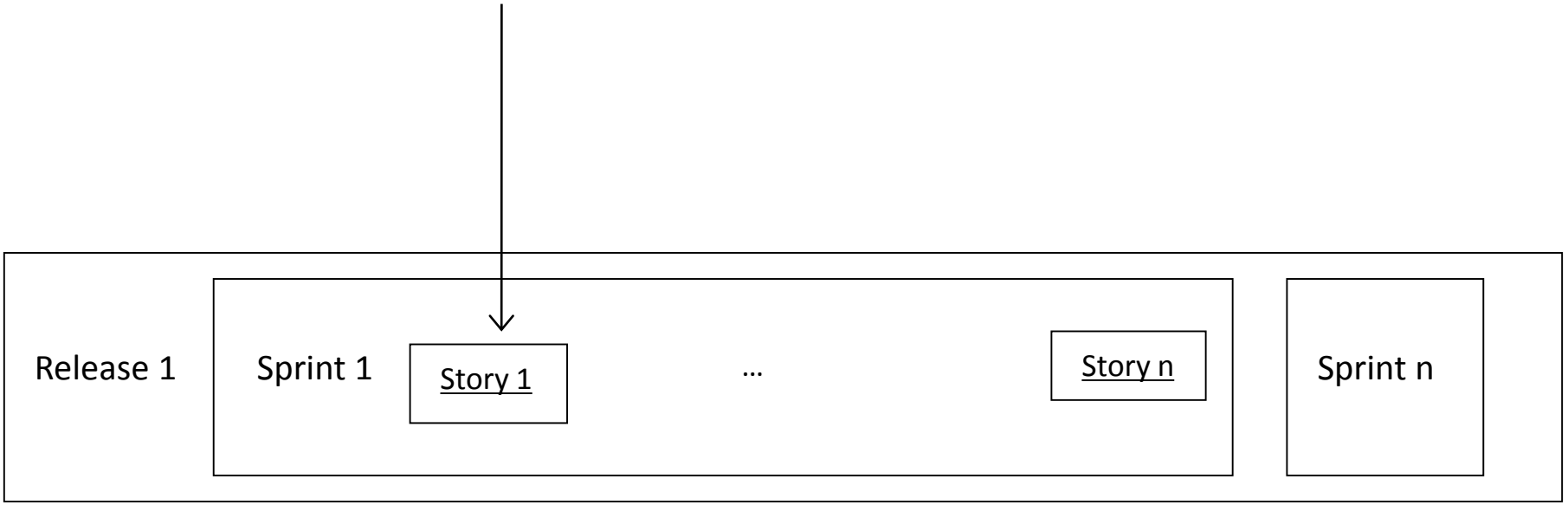
User Story versus Use Case d'UML

- Avec **UML** les cas de tests sont représentés avec des diagrammes de séquences :



User Story versus Use Case

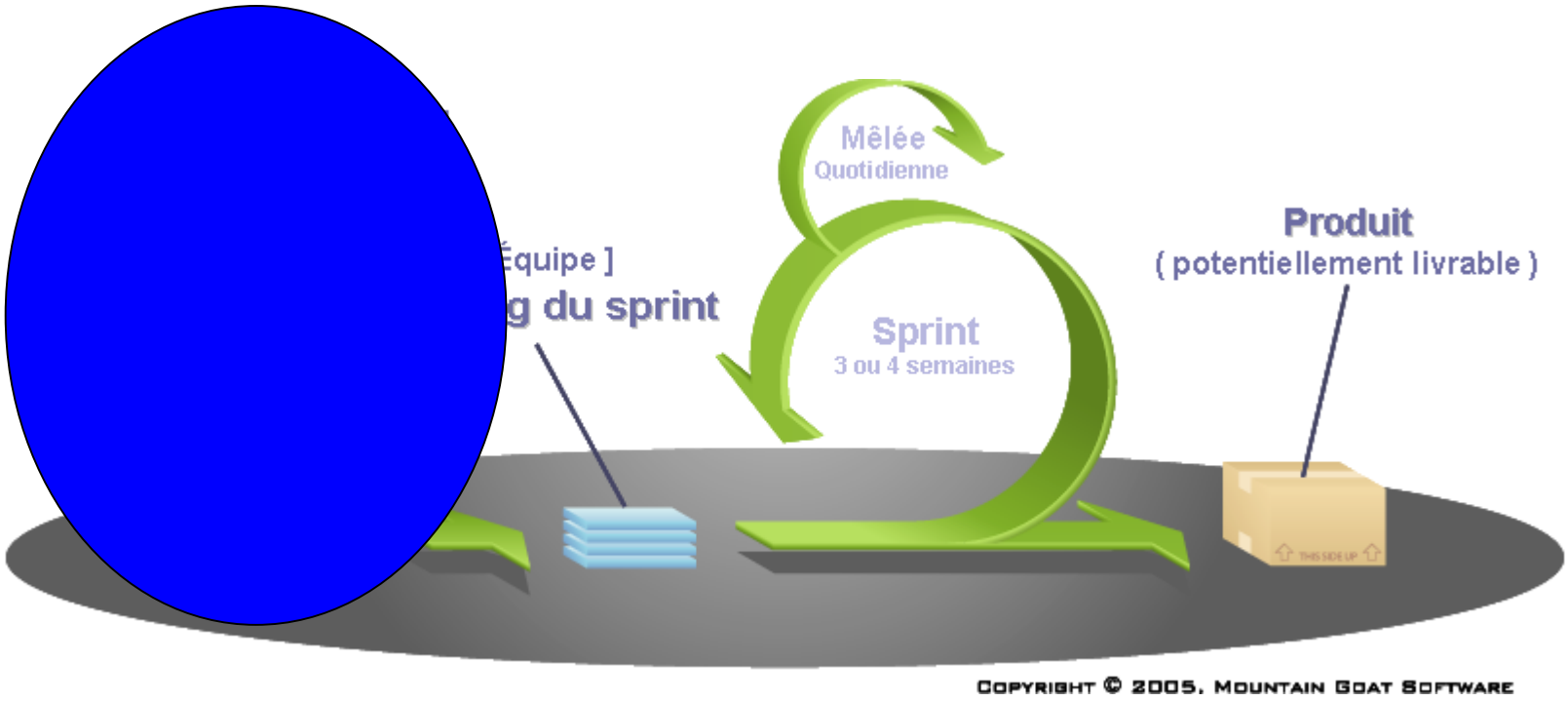
- Conclusion : **user stories** ou **use cases** peuvent être utilisés pour représenter le contenu des sprints.



Comment classer les user stories dans l'ordre de leur développement ?

La grande question ?

- Comment mettre des priorités sur le backlog du produit ?

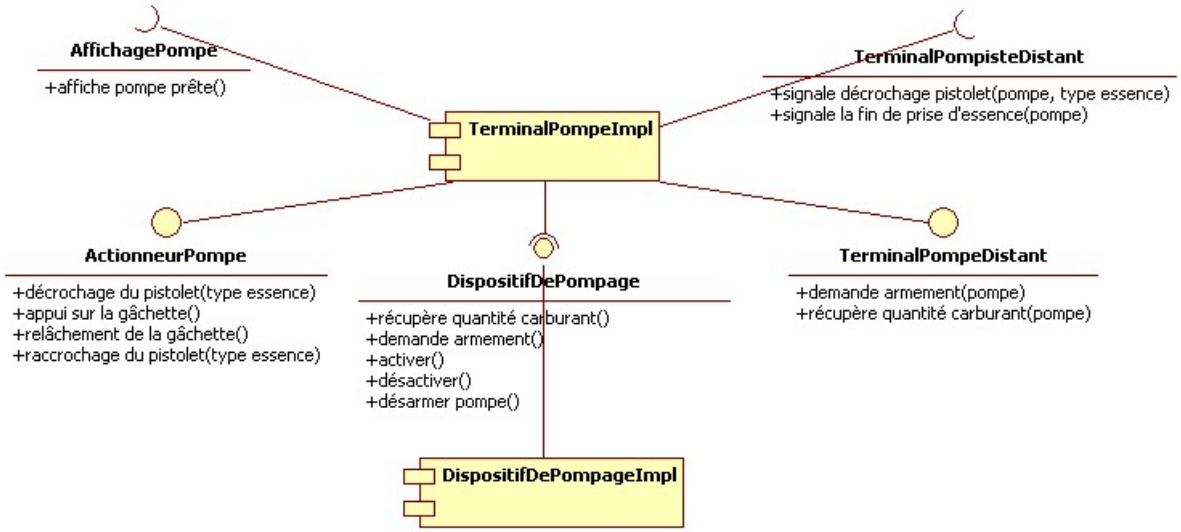


Comment constituer le backlog du produit ?

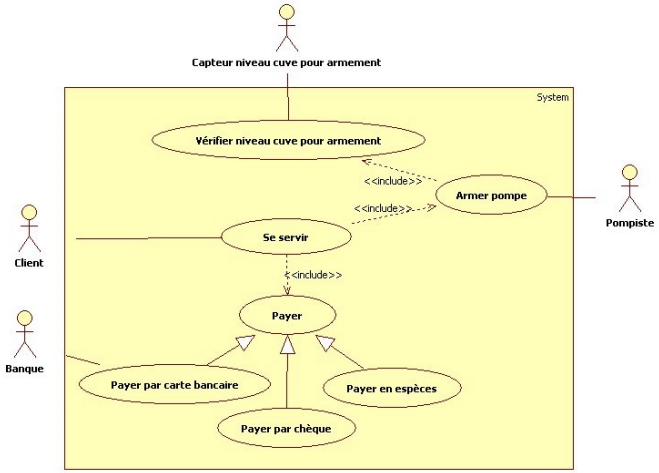
- Classer les user stories uniquement d'après la demande des utilisateurs peut conduire à un refactoring important du code car il peut y avoir des dépendances entre le code d'une fonctionalités à une autre =>

  Il est nécessaire de définir l'architecture de l'application pour repérer les dépendances :

  - Les diagrammes de composants d'UML peuvent illustrer les dépendances :
    - Par exemple TerminalPompeImpl dépend de DispositifDePompage :

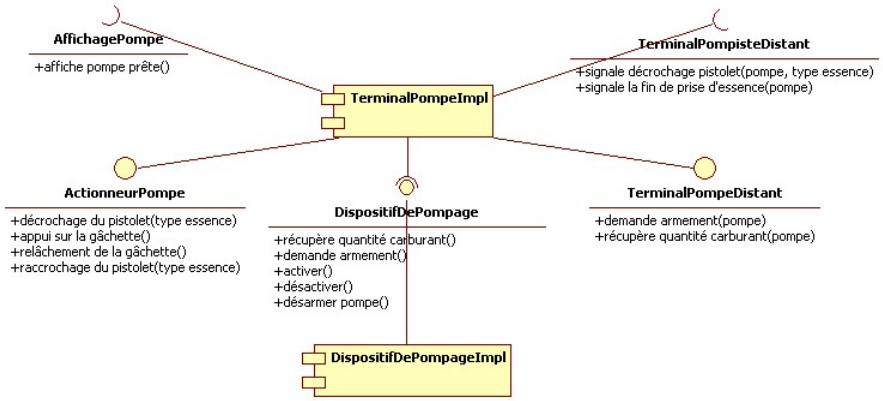


Comment constituer le backlog du produit ?



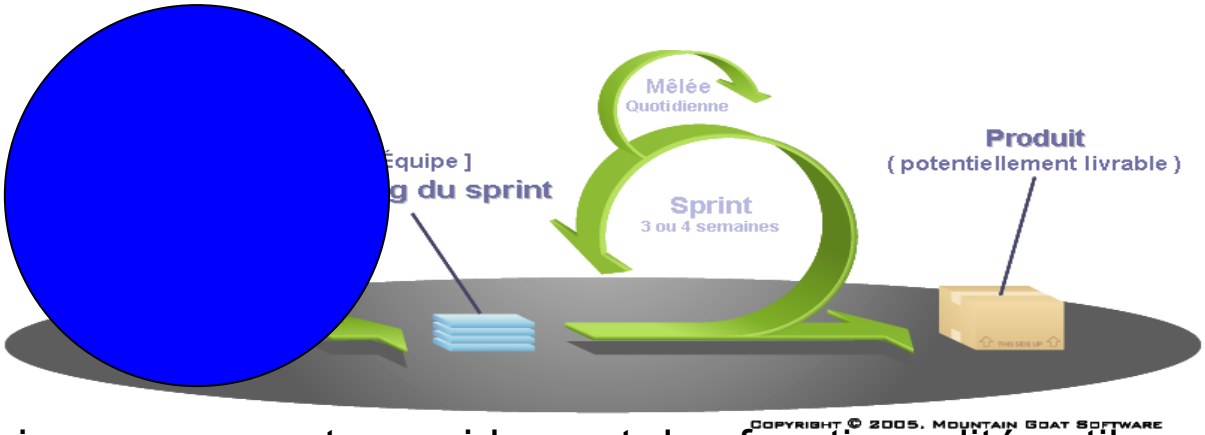
Fonctionnalités

+



Composants

=

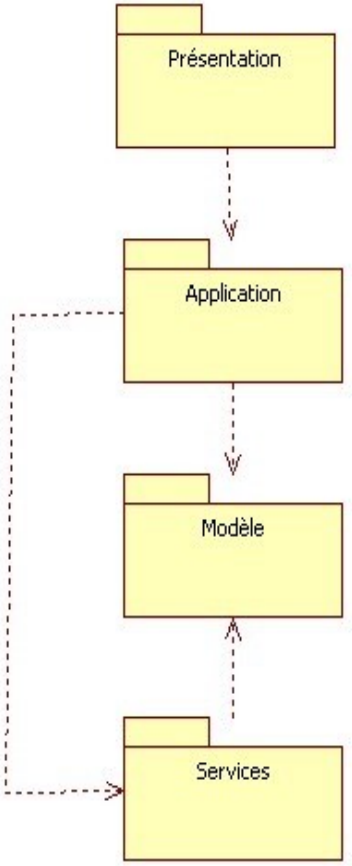


- Classer les user stories pour apporter rapidement des fonctionnalités utiles aux utilisateurs, tout en commençant par les stories qui ont des composants logiciels dont dépendent les autres composants.

Améliorer l’architecture de l’application

Application de “patterns” de conception

- Il est toujours souhaitable, si possible, de conformer l’architecture à des “patterns” de conception, découpage en couches, modèles MVC... Car cela permet d’avoir une architecture modulaire. Ce qui limitera le refactoring du code.
    - Exemple d’une architecture en couches :



Utilisation de framework

- Il est rare de nos jours que l'équipe de projet n'utilise pas de framework de développement tels que zend pour PHP, Spring en Java...

- Les avantages principaux à utiliser un framework :
  - réutilisation de composants logiciels fournis par le framework qui font gagner beaucoup de temps de développement.
  - communauté active qui aide à résoudre rapidement les problèmes.

- Les risques principaux à utiliser un framework :
  - Un framework mal adapté peut conduire à écrire soi même des composants qu'un autre framework aurait fourni.

Il n’y a pas que des user stories !

Les stories “techniques”

- Les membres de l’équipe de projet ont à réaliser des tâches qui ne sont pas uniquement du développement :
  - Installation et paramétrage des logiciels de développement
  - Mise en place de l’environnement de test
  - …
- Toutes ces taches qui prennent du temps doivent figurer dans le backlog du produit, pas en tant que **user stories**, mais en tant que **technical stories**.
- Exemple :
  - « En tant que développeur, je veux disposer d’un environnement de développement intégrer afin de pouvoir participer au développement collaboratif de l’application. »

Résumé sur la constitution du backlog du produit

Conclusion sur comment constituer le backlog du produit ?

1. Définir les releases.
2. Définir les user stories.
3. Placer les user stories dans les releases.
4. Définir l'architecture de l'application :
    1. Diagramme de déploiement.
    2. Diagramme de composants :
        1. Déduire les interfaces des user stories.
        2. Placer les composants qui implantent les user stories et leurs interfaces dans le diagramme de déploiement.
    3. Diagramme de classes métier.
    4. Utiliser des patterns de conception, évaluer l'intérêt d'utiliser des framework
5. **Ajouter les story techniques.**
6. Classer les user stories par ordre de développement en respectant les contraintes suivantes :
    1. Satisfaire le client.
    2. Commencer par développer les composants dont dépendent les autres composants.

Comment fixer la date d'achèvement du développement des user stories ?

Définir et planifier les releases

1. Estimer grossièrement, en **points**, toutes les **stories** du backlog
2. Définir le critère de fin de chaque **release**
3. Définir la durée des **sprints** (durée fixe)
4. Estimer la capacité de l'équipe
5. Planifier le **release** => **plan de release**

- 1 **point** (valeur relative sans unité).
- Les points sont attribués par rapport à une **story** connue qu'on estime valoir X points.

- Revoir le plan de release à la fin de chaque sprint

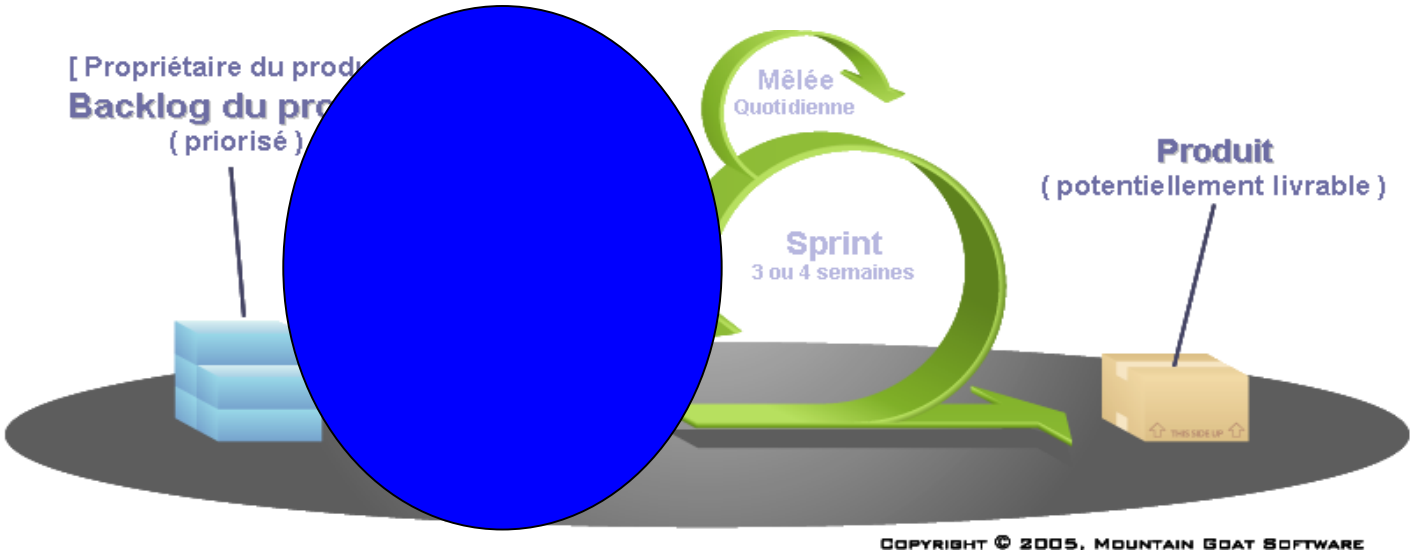
- Ordre de grandeur couramment pratiqué :
  - 2 semaines / sprint
  - 5 sprints / release

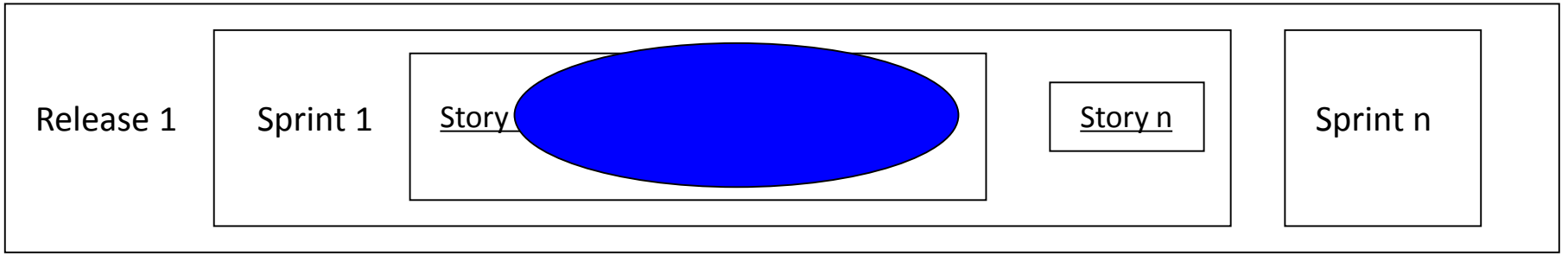
La fin de la phase de préparation des sprints

- A ce stade la phase de préparation des sprints est terminée et le développement peut commencer.

Comment travailler en équipe durant les sprints ?

La réunion de planification du prochain sprint



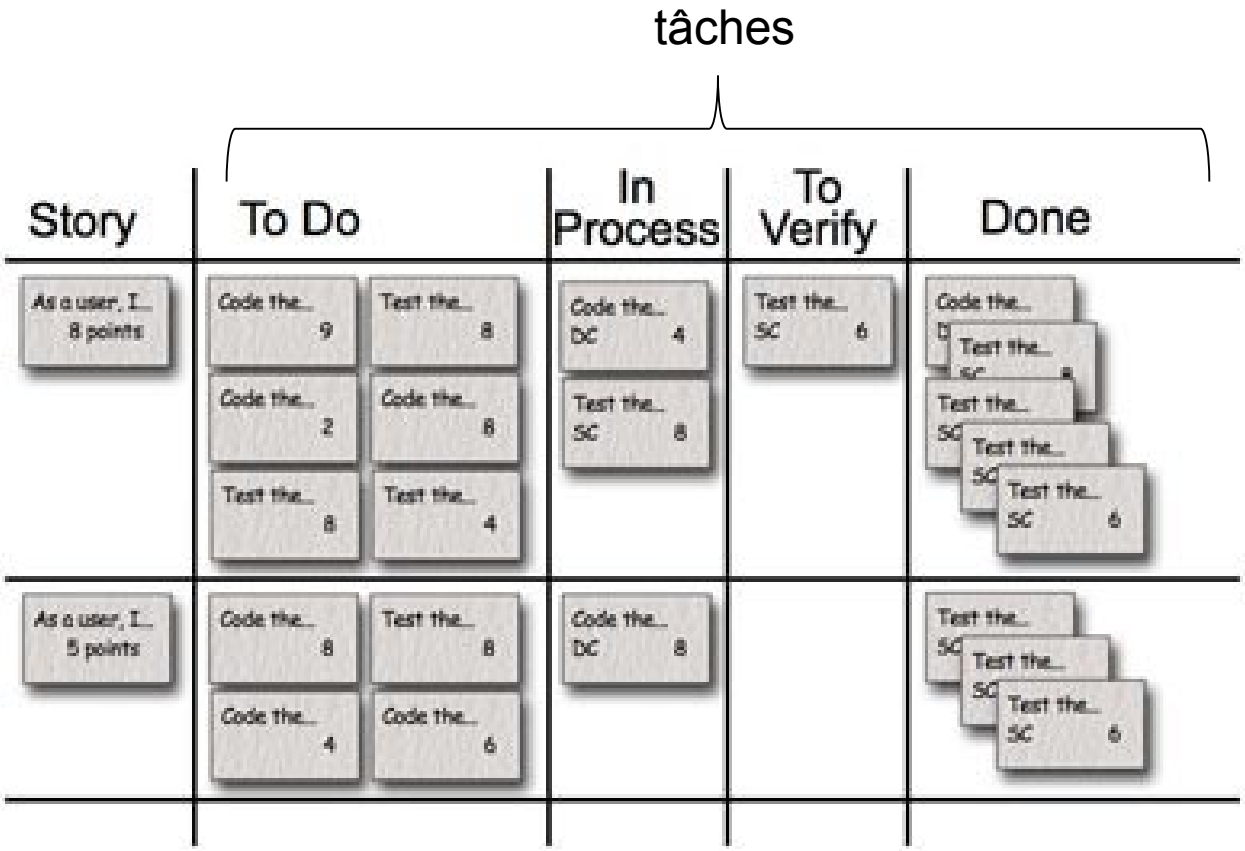


Rappel des principes des méthodes agiles

- « Les gens de l'art et les développeurs doivent collaborer quotidiennement au projet ».
- « Bâtissez le projet autour de personnes motivées. Donnez leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail ».
- « La méthode la plus efficace pour transmettre l'information est une conversation en face à face ».
- « Les processus agiles promeuvent un rythme de développement soutenable. Commanditaires, développeurs et utilisateurs devraient pouvoir maintenir le rythme indéfiniment ».
- « Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto-organisent ».
- « À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens ».

Le découpage en tâches des stories

- Lors de la réunion de préparation du sprint, chaque story (user ou technical) est décomposée en tâches :
    - Les tâches sont attribuées aux membres de l'équipe :



Les storytests

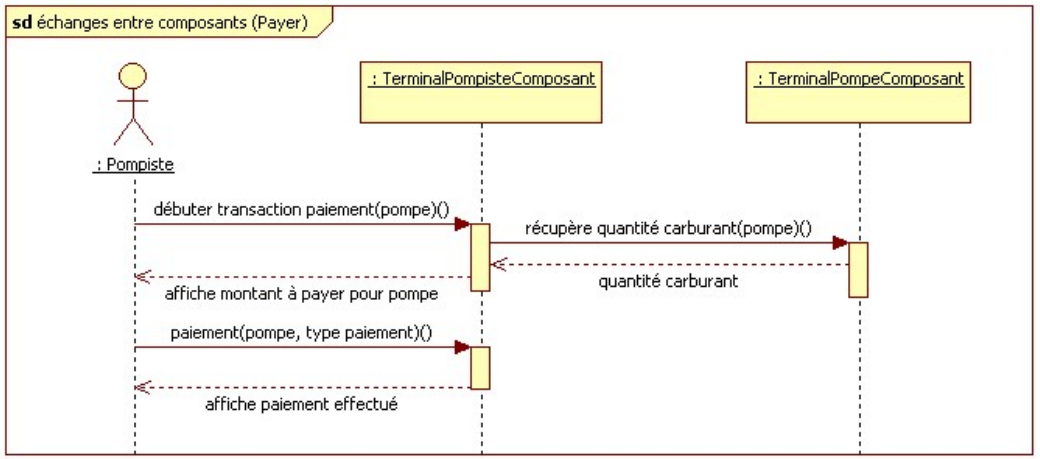
- Avec les user stories :
  - les **user stories** sont déclinées en cas de test appelés **storytests**.
    - Exemple du test associé à la condition de satisfaction « Vérifier que la pompe est armée et est prête à l'emploi » :

Pré-condition : un client a décroché le pistolet du diesel

l'écran du pompiste indique que le pistolet du diesel a été décroché
Le pompiste arme la pompe correspondante

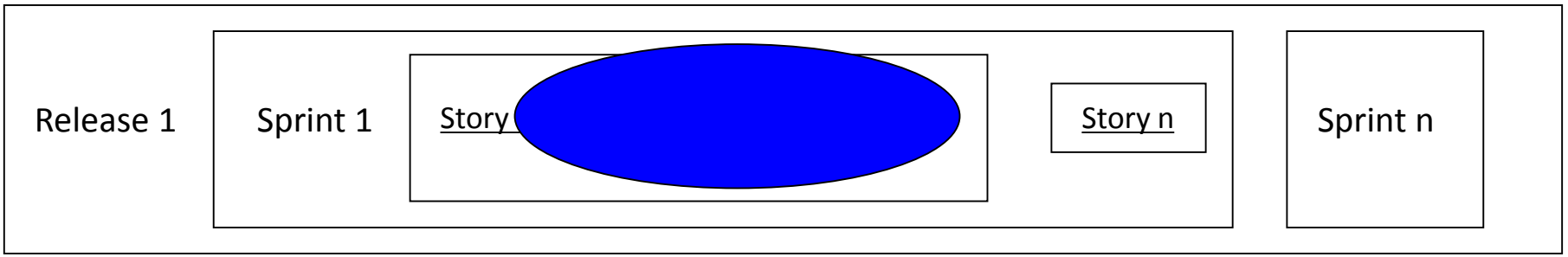
Post-condition : la pompe a été armée (la confirmation de l'armement apparaît sur le terminal du pompiste) et le client peut se servir de l'essence.

- Avec UML :
  - Les diagrammes de séquences servent de **storytests**



Résumé sur la constition du backlog du sprint

- Le backlog du sprint contient :
  - La liste des tâches à effectuer pour développer chaque story.
  - La liste des story tests associés à chaque story.



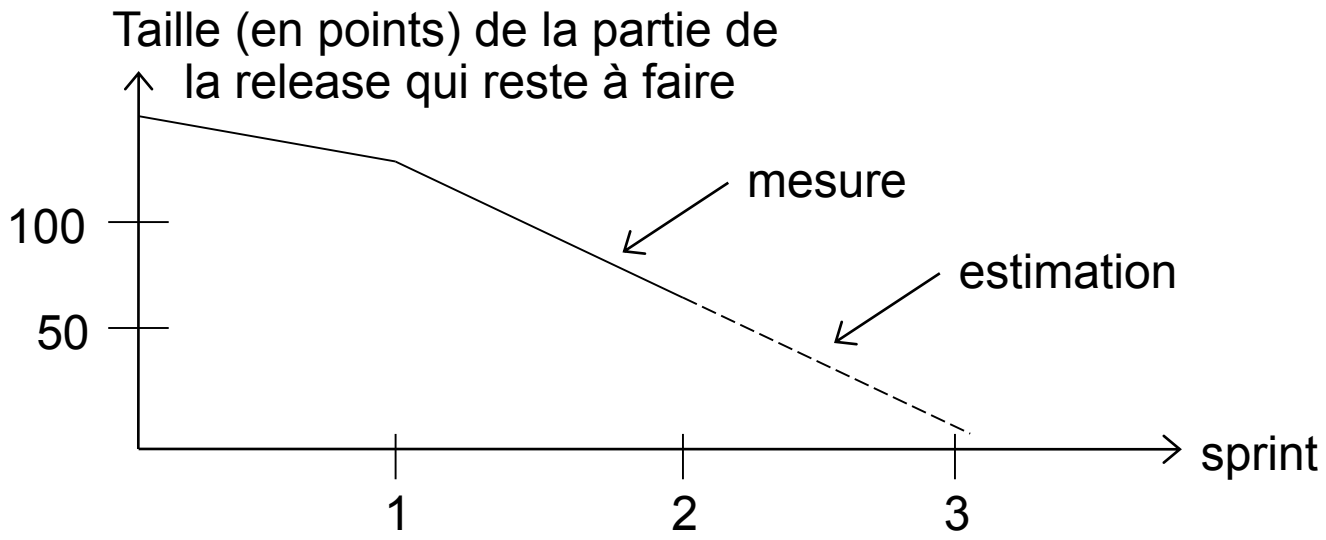
Indicateurs de suivi du projet et divers outils

Quelques indicateurs de suivi

- La **vélocité** est la mesure de la partie du backlog réalisée par l'équipe pendant un sprint. Elle se mesure à la fin d'un sprint.
- La **capacité** de l'équipe, basée sur la vélocité, est une prévision est une prévision de ce que l'équipe est capable de faire pendant un sprint.

- Un **burndown chart** est une représentation graphique du reste à faire dans une période (une release, un sprint…).

Indicateurs de suivi



Exemple d'estimation pour un backlog de 100 points avec une équipe ayant une capacité de 25 points :

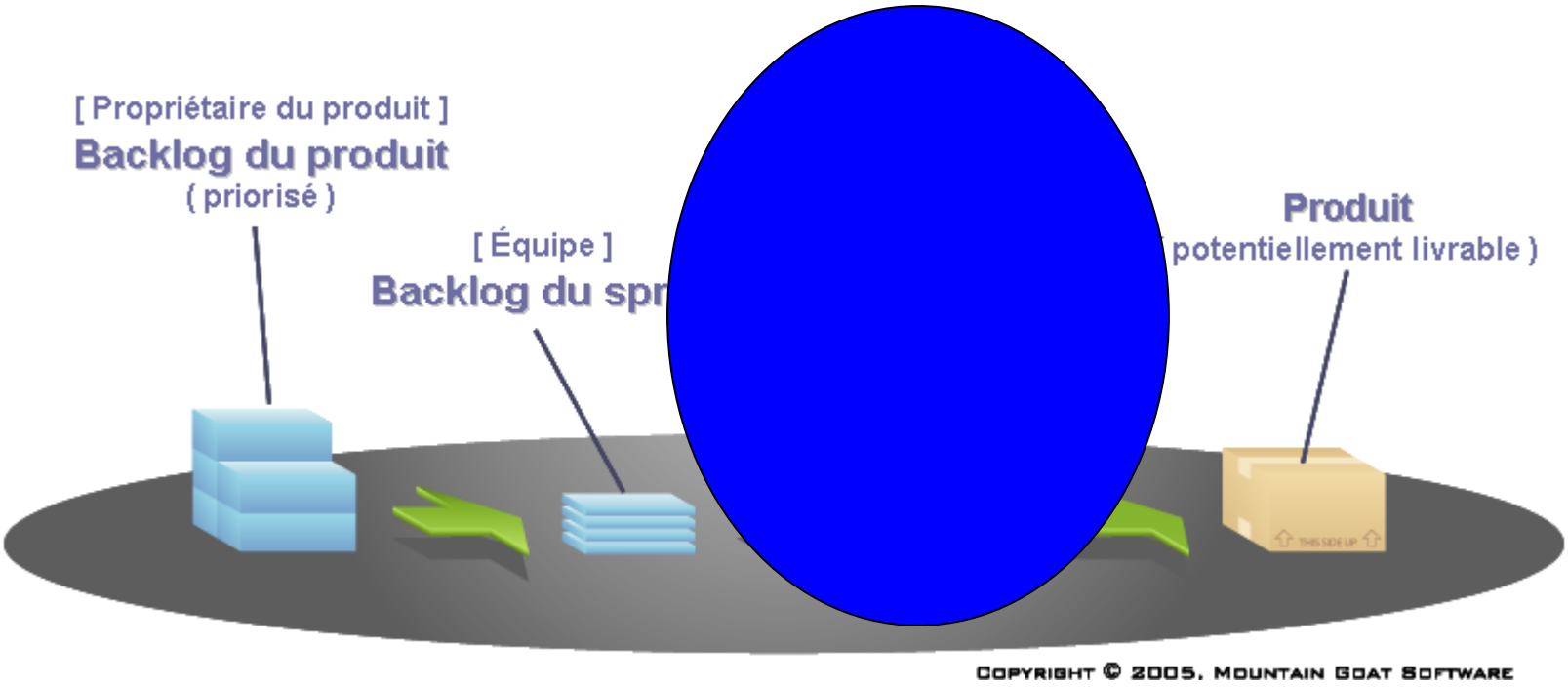
- 4 sprints sont nécessaire
- 16 semaines sont nécessaire si un sprint dure 4 semaines.

Quelques outils qui aident au développement

- Logiciel de suivi de problèmes (bug tracker) : bugzilla, mantis…
- Automatisation des tests : Junit, Marathon…
- Logiciels de développement collaboratif et gestion de version : SVN, Github.
- Logiciel Scrum :
  - Scrumwise (payant)
  - Ice Scrum (pour un premier test)

Comment travailler en équipe durant les sprints ?

Le travail durant les sprints



Développement guidé pas les tests

1. Ecrire les tests d'acceptation pour chaque story (les storytests)
2. Développer
3. Tester

- L'utilisation de tests d'acceptation ne dispense pas :
  - De faire des tests unitaires.
  - D'avoir des vecteur de tests avec un bonne "couverture".

La réunion quotidienne

- But :
  - Éliminer les obstacles de l'équipe, communiquer, évaluer l'avancement du travail.
- Qui : tous les membres de l'équipe (d'autres personnes peuvent y assister sans intervenir).
- Fréquence : quotidienne.
- Durée : ¼ d'heure maximum.
- Le cérémonial :
  - Répondre aux 3 questions :
    - qu'as-tu fait depuis hier ?
    - que prévoies-tu de faire aujourd'hui ?
    - quels sont les obstacles qui te freinent dans ton travail ?
- Les résultats :
  - Actualiser le plan du sprint (modifier la liste des tâches, les ré-affecter…).
  - Actualiser le burndown chart de sprint.
  - Actualiser une liste des obstacles (qui peuvent donnés lieu à des tâches).

A la fin d'un sprint

- La revue de sprint :
  - But : montrer le produit aux personnes impliquées dans le projet.
  - Etapes :
    - Préparer la démonstration.
    - Rappeler les objectifs du sprint.
    - Effectuer la démonstration.
    - Calculer la vélocité.
    - Ajuster le plan de release.
- La retrospective de sprint :
  - L'équipe « refait le match » et cherche quelles améliorations peuvent être mises en place lors du prochain sprint.