

# *CONDUITE DE PROJET LOGICIEL*

**LES METHODES AGILE**

# Agilité

---

AGILITE: Aptitude à s'adapter pour une entreprise ou une organisation

Méthodes agiles: Ensemble de principes et de valeurs pour le développement de logiciel basé sur des techniques de production légères pour produire de la valeur rapidement et fréquemment



# Bref historique

---

**L'agilité est à la mode, mais les méthodes de développement logiciel dites « itératives et incrémentales » existent depuis longtemps:**

- **1986 : Modèle de la spirale de B.W. Boehm**
- **1991: Méthode RAD de James Martin**
- **1995: SCRUM**
- **1996: eXtrem Programming.**
- **2001: Manifeste Agile = Texte écrit par 17 figures éminentes du développement logiciel**

***" Nous avons trouvé une voie améliorant le développement logiciel en réalisant ce travail et en aidant les autres à le faire. De ce fait nous avons déduit des valeurs communes."***

# Manifeste Agile

---

- Texte rédigé par 17 experts reconnus pour leurs apports respectifs au développement d'applications informatiques sous la forme de plusieurs méthodes dont les plus connues sont Extreme Programming et Scrum.
- Valeurs et principes du Manifeste Agile défendus par l'Agile Alliance.
- Principes:
  - Abandon du cycle de développement en cascade qui ne correspond plus aux nouveaux besoins applicatifs.
  - Style de conduite de projet itératif et incrémental,
  - Autonomie des ressources humaines impliquées dans la spécification, la production et la validation
  - Intégration et test en continu.

# Les 4 valeurs du Manifeste Agile

---

Personnes & interactions

Processus & outils

Logiciel exécutable

Documentation

Collaboration avec le client

Respect d'un contrat

Adaptation au changement

Prévue au début du projet

# Les 12 principes du Manifeste

---

1. Chercher à satisfaire le client avec un logiciel utile dès que possible .
2. Accepter le changement demandé même tardivement
3. Livrer fréquemment (2 sem./2 mois) une application fonctionnelle.
4. Faire collaborer les experts métier et les développeurs en permanence.
5. Bâtir le projet autour de personnes motivées.
6. Privilégier la communication en face à face.
7. Mesurer l'avancement au travers du logiciel fonctionnel
8. Adopter un rythme de développement durable.
9. Encourager l'excellence technique et la qualité de la conception
10. Faire simple et ne faire que ce qui est nécessaire (pas de travail inutile)
11. Laisser les développeurs s'auto-organiser et les y aider
12. Réfléchir, de temps en temps, aux moyens de devenir plus efficace.

# Processus agile

---

En plus d'être itératif et incrémental, un processus agile est caractérisé par

- des itérations courtes et régulières
- un client qui fait partie de l'équipe
- une capacité au changement
- une forte implication des membres de l'équipe dans la conduite du projet
- des pratiques d'ingénierie permettant de garantir la qualité du code

# Quelques méthodes agiles

---

- RAD (Rapid Application Development)
  - Première méthode Agile, ancêtre de PUMA
- DSDM (Dynamic Systems Development Method)
  - Méthode UK, Schéma directeur et réversibilité
- FDD (Feature Driven Development)
  - Définition d'itération guidée par la « Business value »
- SCRUM
  - Focalisation de l'équipe sur des itérations courtes (Sprints),
- XP (eXtrem Programming)
  - La plus connue
- PUMA ([Proposition pour Unification des Méthodes Agiles](#))



# Rapid Application Development

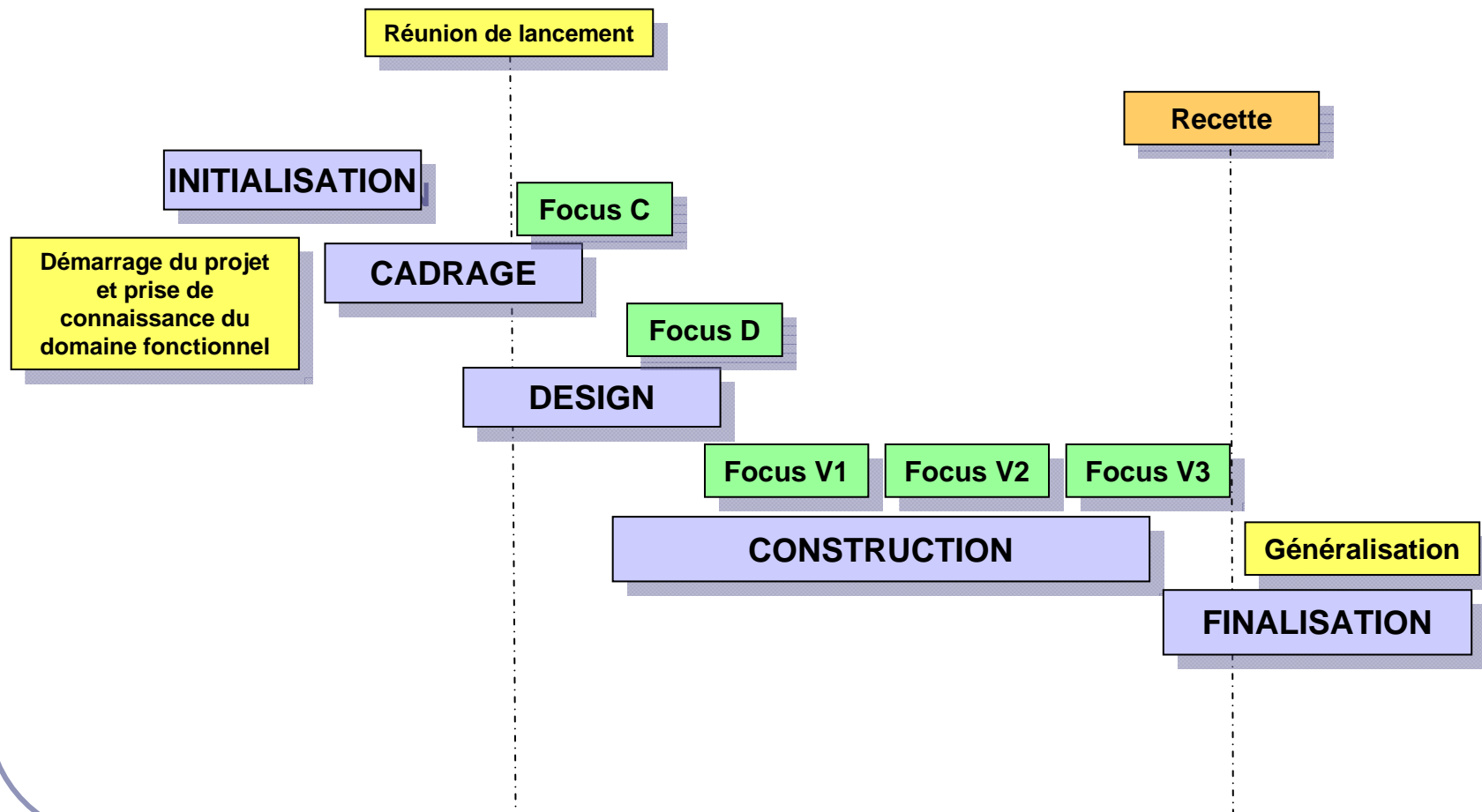
---

## 5 phases :

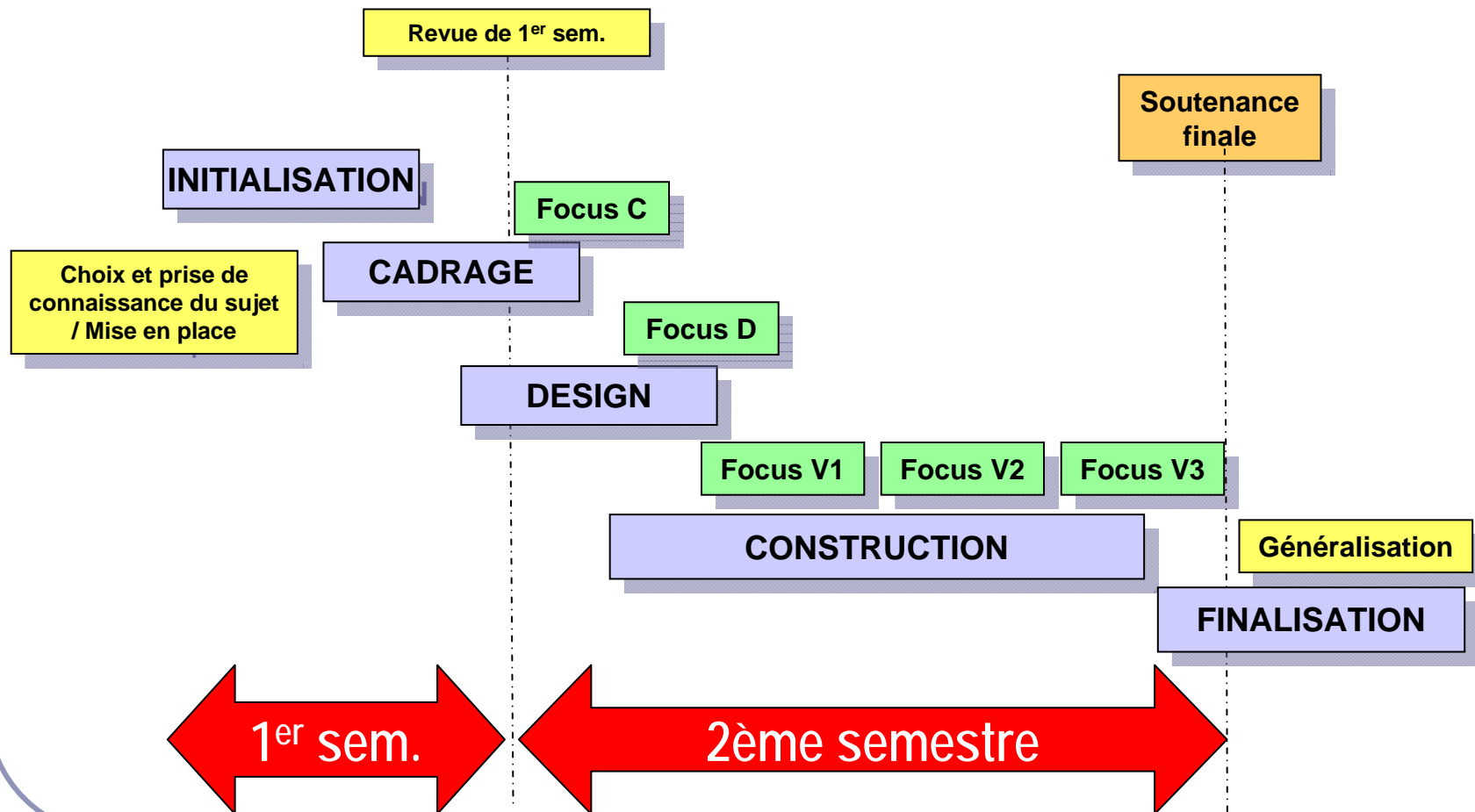
- **Initialisation:** organisation, définition du périmètre et du plan de communication.
- **Cadrage:** définition des objectifs, solutions et moyens.
- **Design:** modélisation de la solution.
- **Construction:** Prototypage actif (intégration/validation continues)
- **Finalisation:** contrôle final de qualité en site pilote.



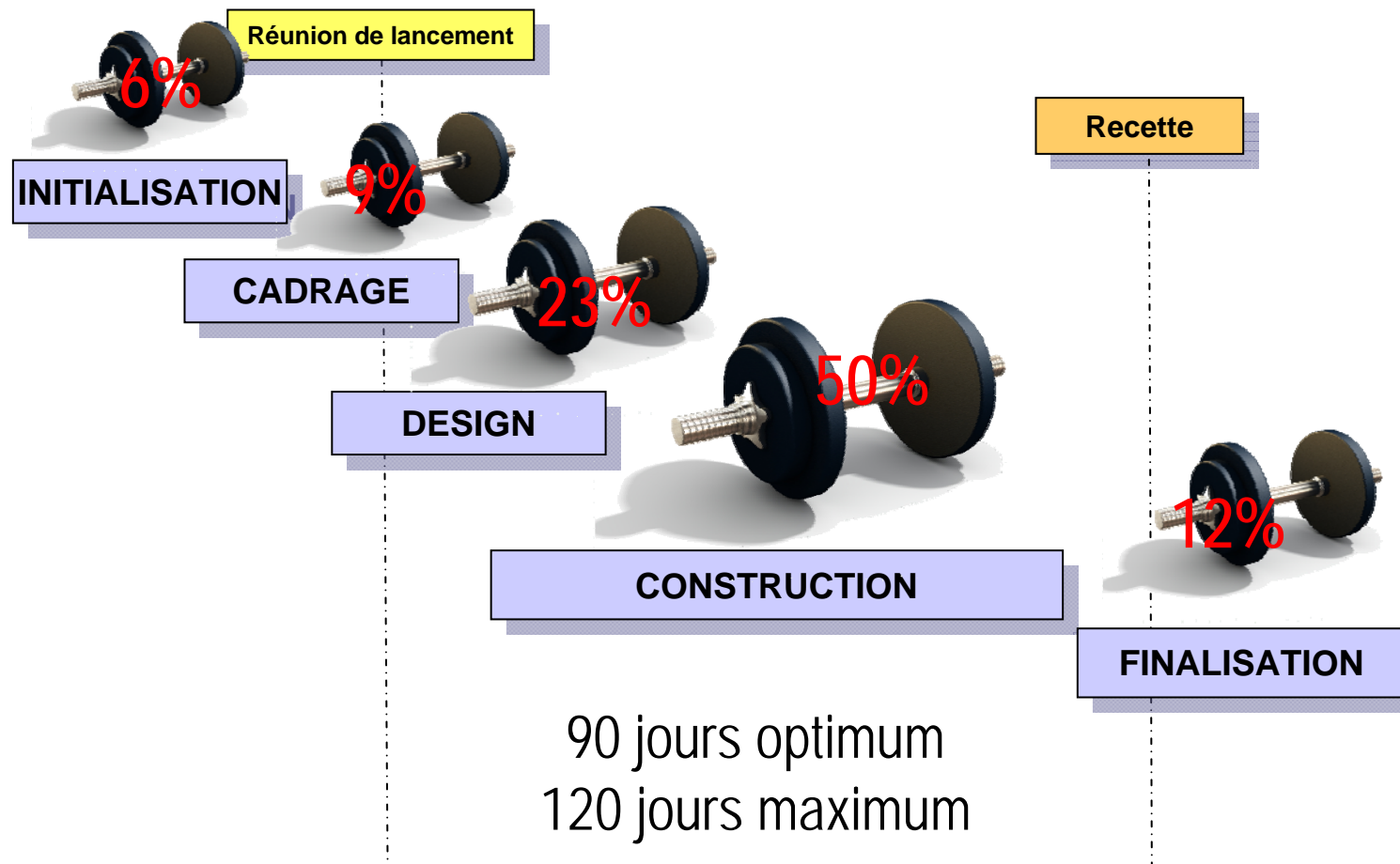
# Processus RAD



# Le RAD de votre projet



# Répartition de l'effort



# eXtreme Programming (XP)

- “Nouvelle” *méthode* de développement dédiée à de petites équipes confrontées à un environnement changeant ou des besoins mal connus
- Repose sur une réduction du coût de modification du logiciel
- 2 objectifs majeurs:
  - ➡ **Développer vite** : «Le plus tôt et le plus rapidement possible mais en prévoyant les changements »
  - ➡ **Développer juste** : « En se focalisant sur les besoins du client.
- Ouvrage de référence: « **Extreme Programming Explained : Embrace Change** »  
par Kent Beck,
- Livre en français: « **L'eXtreme Programming** »  
J.L.Bénard - L.Bossavit - R.Médina - D.Williams (Eyrolles)
- Sites WEB: [www.extremeprogramming.com](http://www.extremeprogramming.com)  
[www.xp-France.org](http://www.xp-France.org)

# XP : Principes

---

## ● 5 principes majeurs

- Livraisons fréquentes
- Développement itératif
- Suivi et mesures d'avancement + ajustement du planning à chaque itération.
- Qualité et simplicité du design et du code
- Travail en équipe

## ● 4 valeurs fondamentales

- Communication
- *Feedback*
- Simplicité
- Courage

# XP et la communication

---

- Un projet est un travail d'équipe
- Rien ne remplace le contact humain et la communication orale
- Les éléments de communication écrits figurent directement dans le code
- Les pratiques XP sont plus faciles à mettre en œuvre dans une équipe de petite taille

# XP et la simplicité

---

- La solution la plus simple est, a priori, la meilleure
- La solution ne doit rien faire de plus que répondre au problème posé
- La solution doit répondre aux besoins exprimés et non pas aux besoins potentiels
- Les éléments de solution doivent être codés une fois et une seule fois (Simple  $\neq$  simpliste)



## XP et le « feedback »

---

- L'avancement doit être évalué sur des éléments concrets ou factuels
- L'avis du client est toujours pertinent (à tout instant du projet)
- L'expérience est source de progrès
- La livraison de versions préliminaires permet à la fois de préciser le besoin et de rassurer les développeurs

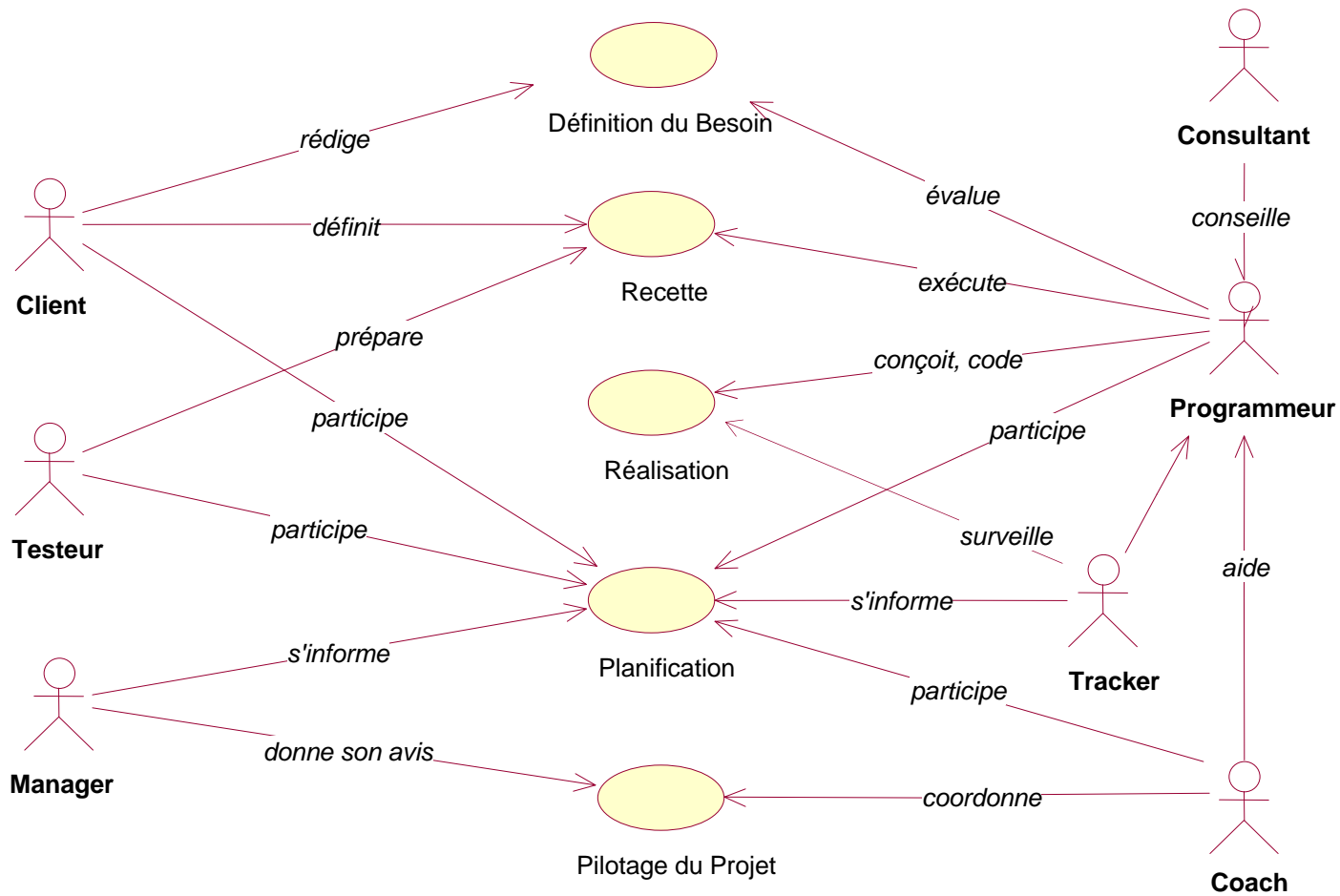
# XP et le courage

---

Ne pas avoir peur

- de débiter le projet sans en connaître précisément les objectifs (prise de risque)
- de prendre des décisions allant dans le sens de la simplification (prise de responsabilité)
- de dire la vérité et d'offrir au client une bonne visibilité du projet (transparence)
- de jeter le code qui n'est pas satisfaisant (remise en question)

# Organisation XP



## XP: Le rôle du programmeur

- Est responsable de la production du code
- Conçoit pour assurer la pérennité du code
- Teste pour assurer la qualité du code
- Dialogue en permanence avec le client
- Procède au remaniement du code
- Émet et révisé des estimations de charge

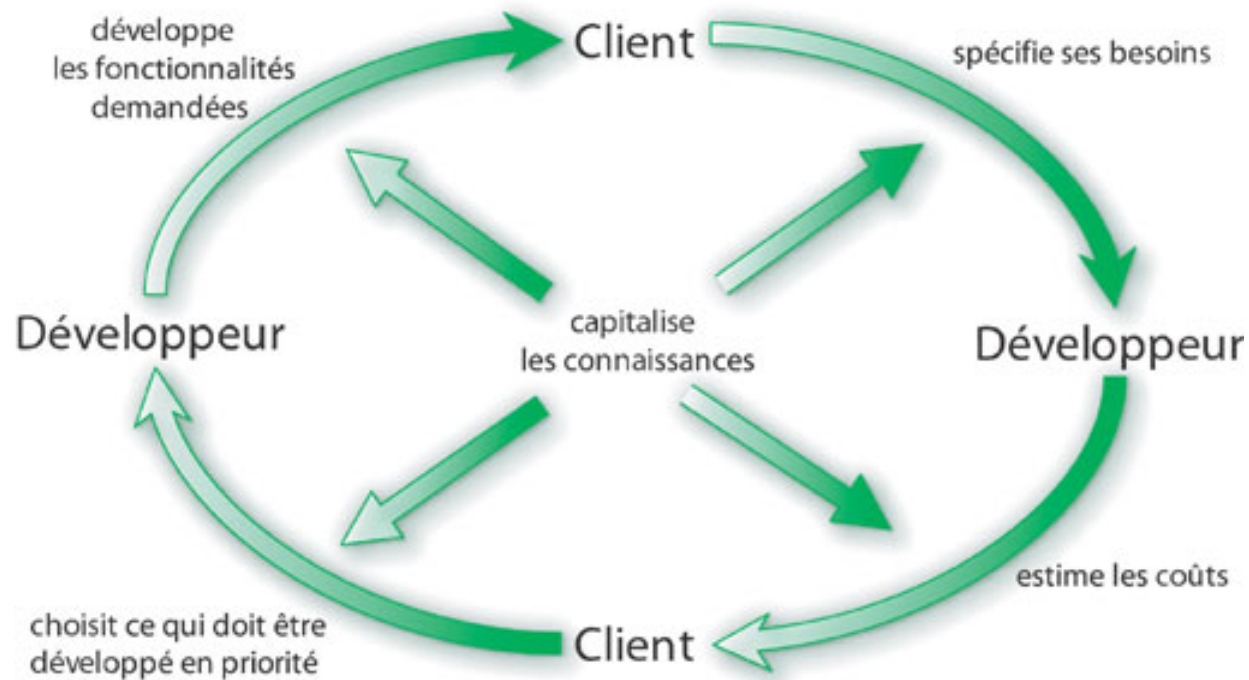


## XP: Le rôle du client

- Fait partie intégrante de l'équipe
- Est responsable de la définition de « ce que doit faire le logiciel »
- Communique les informations utiles aux développeurs
- Dispose d'un « feedback » de la part des programmeurs
- Spécifie les fonctionnalités sur la base de scénarios
- Spécifie les tests de recette
- Participe à la planification et définit les priorités



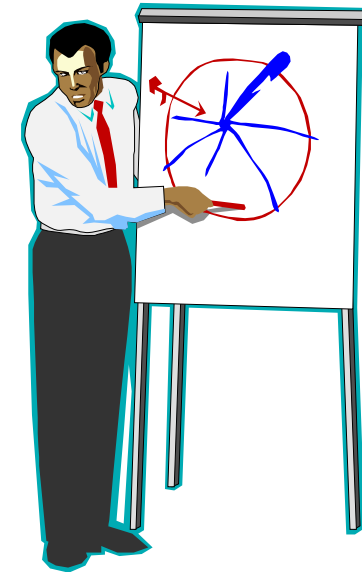
# XP: Relation Client/Développeur



# XP: Le rôle du Coach

---

- Garant du processus et de la méthodologie
- Vérifie que chacun joue son rôle
- Organise et anime les réunions et les séances de planification
- Valide les orientations techniques
- Rend compte de l'avancement au manager



## XP: Le rôle du testeur

---

- Conçoit et réalise les tests de recette définis par le client
- Recherche l'automatisation systématique des tests
- Développe les outils de tests nécessaires et les scripts à exécuter
- Témoigne de l'avancement du projet





## XP: Le rôle du Manager

---

- Ne fait pas partie de l'équipe
- Suit l'avancement du projet et peut exiger des comptes-rendus réguliers
- Fournit les moyens matériels et humains nécessaires pour le projet
- Définit les règles de fonctionnement et les fait respecter



## XP: Le rôle du Tracker

---

- Assure le suivi des tâches et du planning
- Cherche à détecter les difficultés le plus tôt possible
- Ne prend pas de décision mais informe le coach en cas de problème



# XP: Comptabilité des rôles

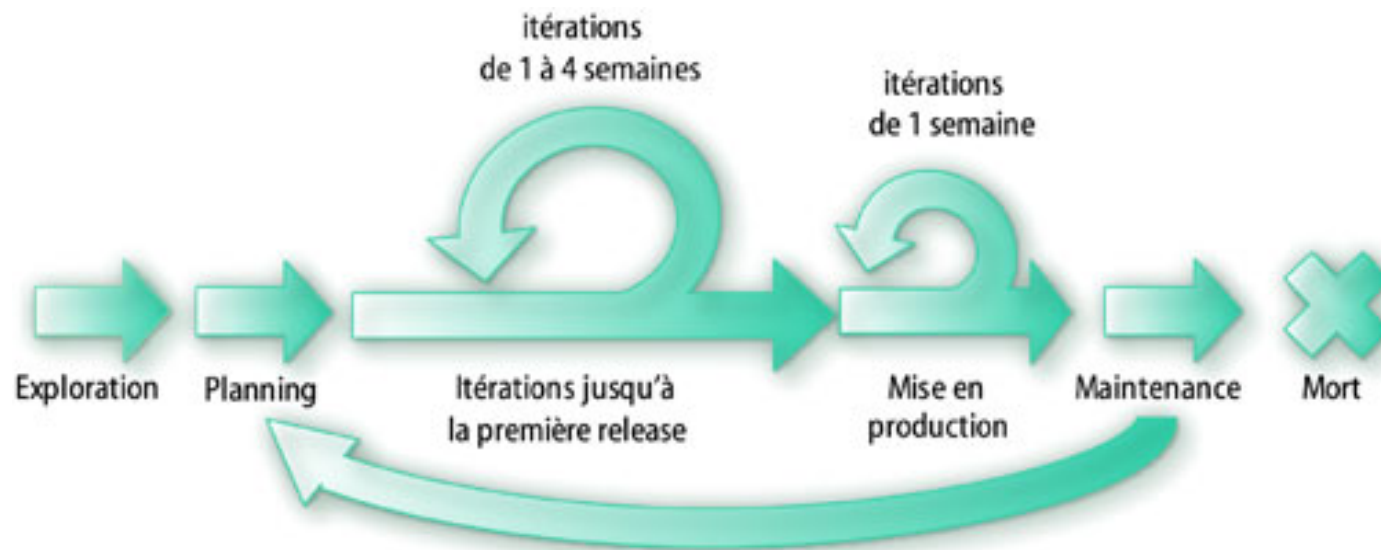
	Programmeur	Client	Testeur	Tracker	Manager	Coach
Programmeur		inconciliable	envisageable	envisageable	inconciliable	envisageable
Client	inconciliable		conciliable	inconciliable	inconciliable	inconciliable
Testeur	envisageable	conciliable		inconciliable	inconciliable	inconciliable
Tracker	envisageable	inconciliable	inconciliable		envisageable	envisageable
Manager	inconciliable	inconciliable	inconciliable	envisageable		inconciliable
Coach	envisageable	inconciliable	inconciliable	envisageable	inconciliable	

# XP: Pratiques de gestion de projet

---

- Livraisons fréquentes
  - *Production de versions préliminaires*
- Planification itérative
  - *Révision périodique du planning avec la participation du client*
- Client sur site
  - *Le client fait partie de l'équipe*
- Rythme de travail durable
  - *Horaires réguliers sans dépassement excessif*

# Le cycle XP



maintenance = ajouter des fonctionnalités → nouvelles releases  
utilise le même processus que pour la release 1

# Gestion des livraisons

4 variables associées à chaque livraison

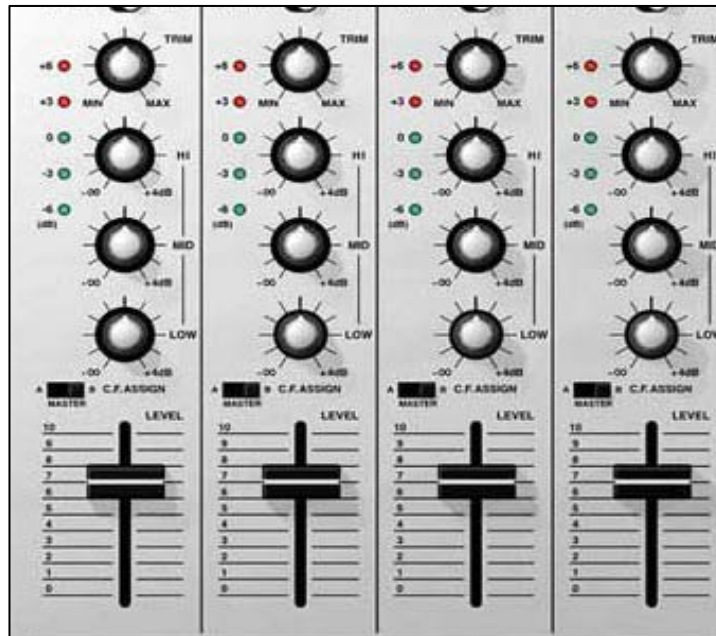
Dates  
Ressources  
Qualité  
Contenu

Généralement dictées  
par des contraintes  
externes

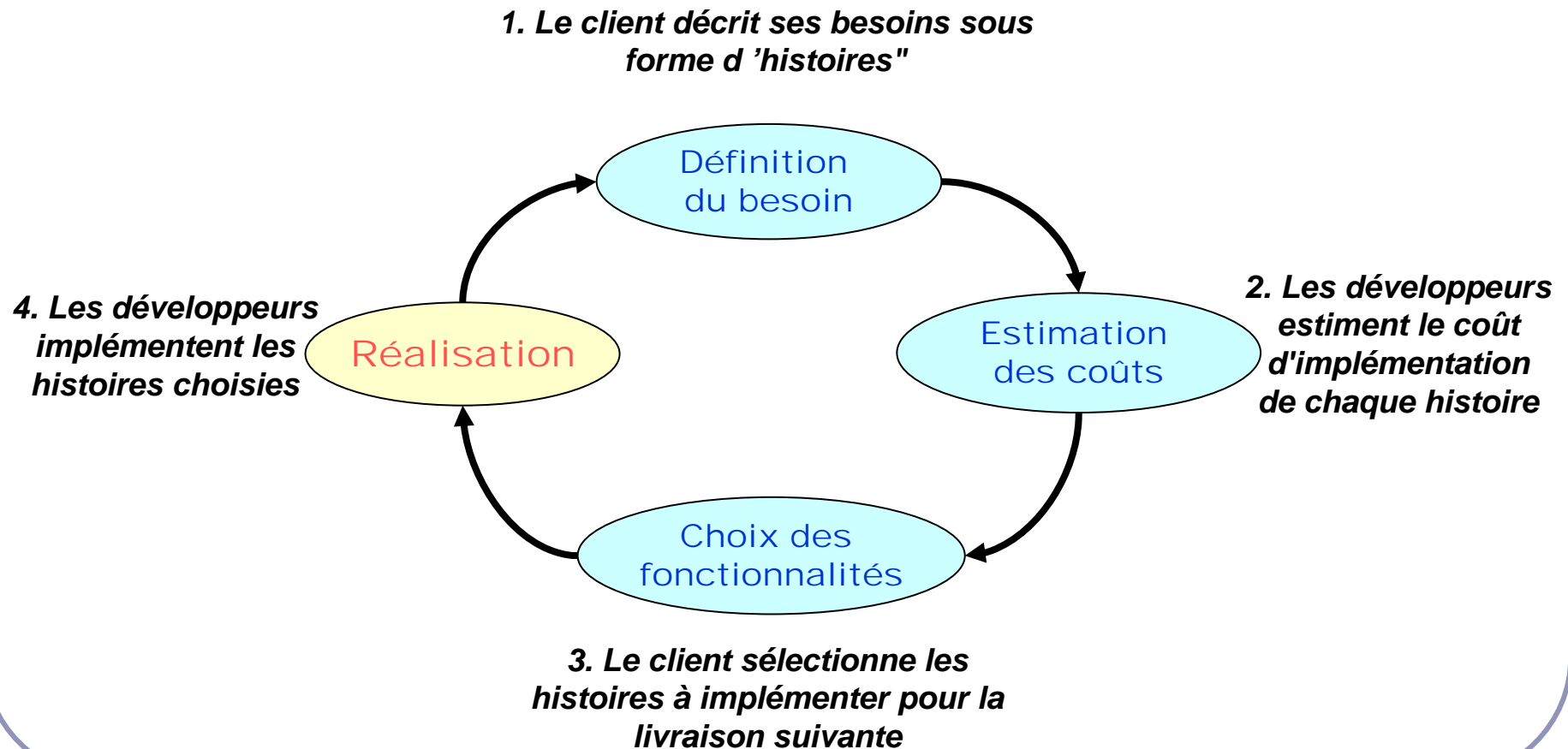
Allouées par la  
direction et difficiles  
à modifier

La meilleure possible  
dans tous les cas

Défini conjointement par le  
client et par les  
développeurs avec  
arbitrage du chef de projet



# Le cycle de livraison XP



# Comparaison XP et cycle en v

## Cycle en V

- Traitement du problème dans son ensemble
- Finalisation de la spécification avant entrée en conception
- Finalisation de la conception avant entrée en réalisation
- Résultat de conception = Document d'architecture
- Fabrication du logiciel = codage
- Chaque phase correspond à une activité et conduit à la production d'un document
- Chaque activité est confiée à des « spécialistes »

## XP

- Décomposition du problème en itérations
- Spécification de la première itération puis développement avec feedback
- Évolution de la conception à chaque itération
- Résultat de la conception = Code
- Fabrication du logiciel = compilation
- La documentation est contenue dans le code
- Polyvalence des développeurs



# XP: Pratiques collaboratives

---

- Stand-up meeting
- Utilisation de métaphores
- Programmation en binômes
- Responsabilité collective du code
- Règles de codage
- Intégration continue

# Le « Stand-up meeting »

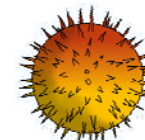
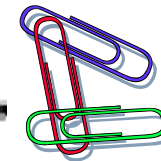
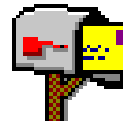
Réunion quotidienne de toute l'équipe

- Animée par le coach
- De préférence le matin
- Durée = une dizaine de minutes
- Point rapide de chaque développeur sur son avancement et les difficultés rencontrées
- Débats techniques interdits



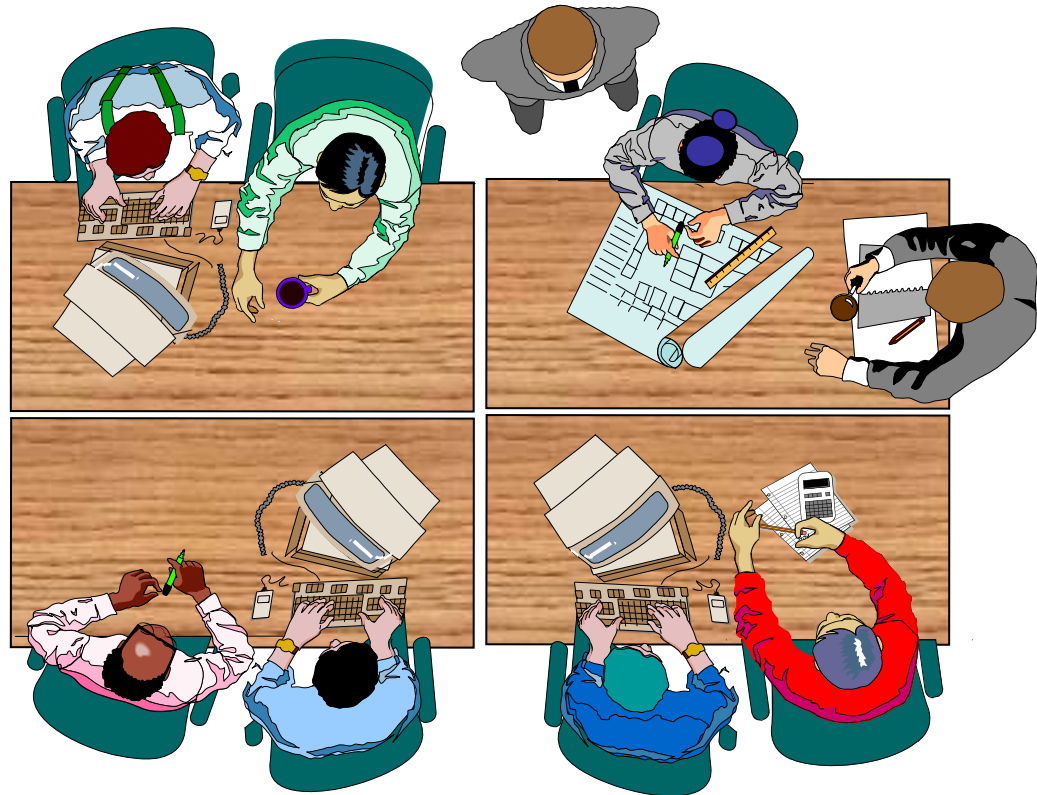
# Utilisation de métaphores

- Faciliter la communication
- Minimiser l'effort de documentation
- Décrire le résultat attendu en termes imagés
- Éviter le jargon technique
- Fédérer les idées autour d'une de l'équipe



# Plate-forme de développement

- Espace central dédié au développement
- Salle de réunion à proximité de la plate-forme
- Bureaux isolés
- Panneaux d'affichage et tableaux noirs



# Programmation en binômes

- Développement systématiquement à deux devant une même machine
- Rôles de « pilote » et de « co-pilote »
- Relecture du code au fur et à mesure de l'avancement
- Recherche de complémentarité dans la constitution des binômes
- Amélioration de la cohésion de l'équipe
- Connaissance de l'application mieux répartie



# Responsabilité collective du code

---

- Chaque binôme peut intervenir sur n'importe quelle partie du logiciel.
- On doit pouvoir s'adresser à n'importe lequel des développeurs en cas de problème
- Les experts ne se cantonnent pas à leur domaine de prédilection mais ils jouent le rôle de consultants internes
- Si une compétence particulière est requise pour une tâche, elle est confiée à un binôme réunissant un expert et un novice



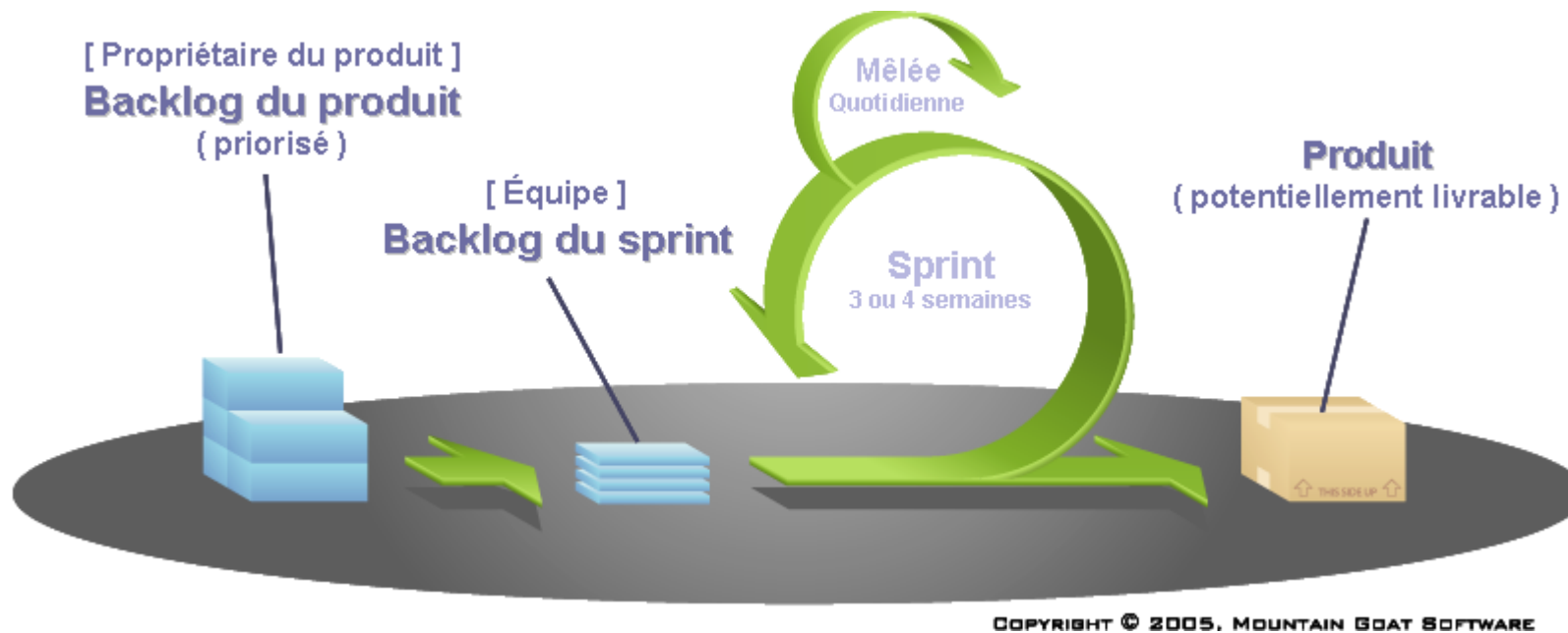
# Scrum = mêlée

---

**Avancer collectivement, à la manière d'une équipe de rugby (vs. course de relais)**



# Processus Scrum





# Principes généraux

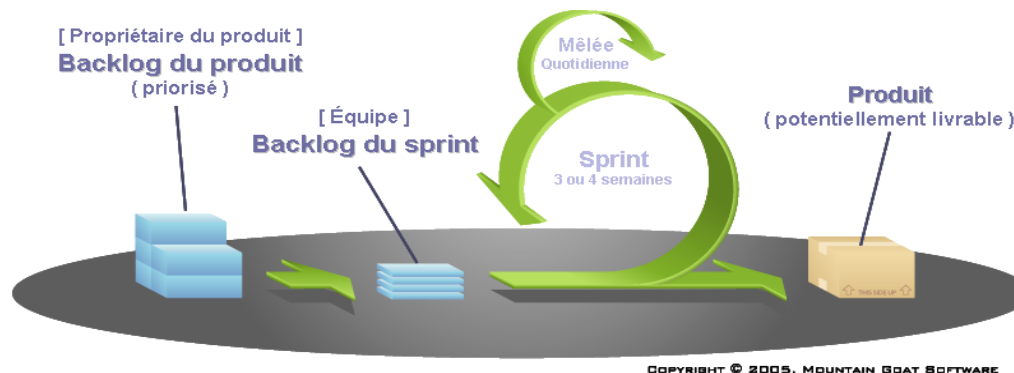
---

- **Du logiciel fonctionnel est produit à chaque sprint (toutes les 2 à 4 semaines).**
- **Le métier définit les priorités. L'équipe s'organise elle-même pour déterminer la meilleure façon de produire les exigences les plus prioritaires.**
- **A chaque fin de sprint, tout le monde peut voir fonctionner le produit courant et décider soit de le livrer dans l'état, soit de continuer à l'améliorer pendant un sprint supplémentaire.**

# Le Backlog du produit

Le product backlog est

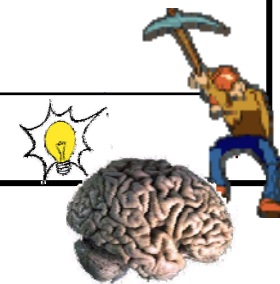
- est une liste de toutes les fonctionnalités, métier ou techniques, qu'on retrouvera dans le produit.
- représente la charge de travail de l'équipe.
- évolue et doit être priorisée en fonction des attentes des clients ou utilisateurs.



Les priorités sont revues à chaque sprint

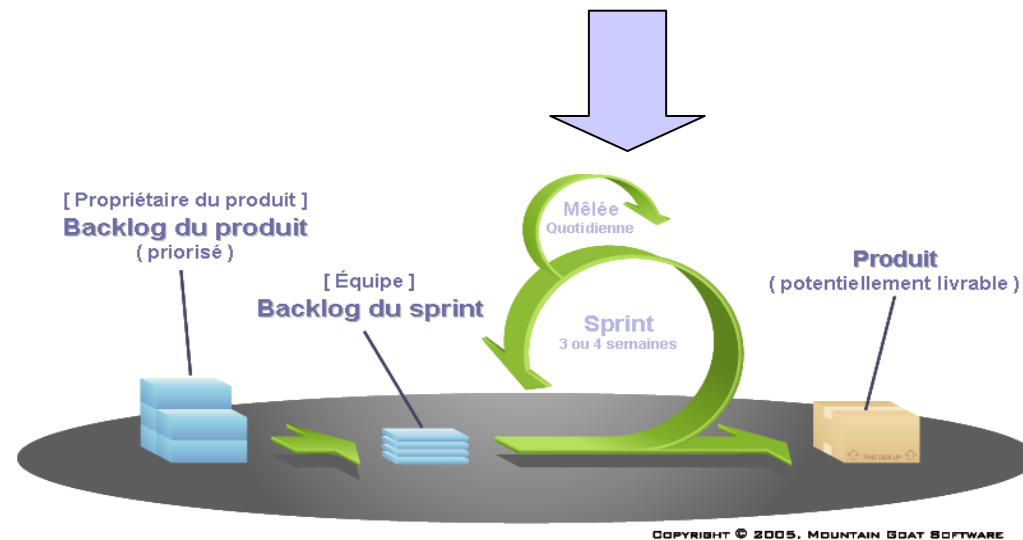
# Exemple de backlog de produit

Élément de backlog	Estimation
En tant que responsable, je peux organiser une réunion	3
En tant qu'invité, je peux m'inscrire à une réunion	5
En tant que participant, je peux exprimer une idée	8
Améliorer l'ergonomie	3
En tant que responsable je peux recenser les présents	3



# Le sprint

- **Déterminé par un objectif sur lequel va se concentrer le travail pendant la durée du sprint**
- **Défini et approuvé par l'ensemble des intervenants dont les utilisateurs, clients ou managers concernés.**
- **A partir de ce but va être défini le backlog du sprint.**



# Planification du sprint

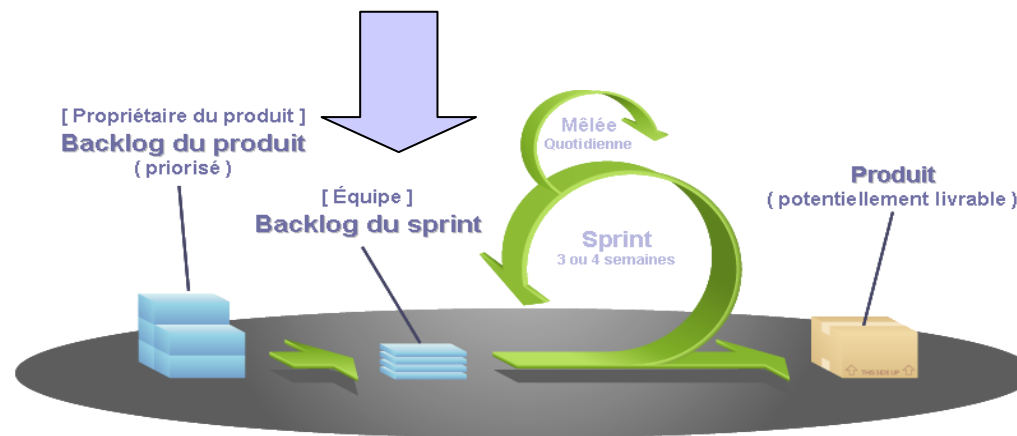
- L'équipe choisit, à partir du backlog de produit, les éléments qu'elle s'engage à finir.
- La liste des tâches est créée (sprint backlog)
  - Les tâches sont identifiées et estimées (1-16 heures)
  - Collectivement, pas seulement par le ScrumMaster
- La conception de haut niveau est abordée

En tant que participant,  
je peux exprimer une  
idée




**Coder l'IHM (2)**  
**Intégrer le service XMPP(3)**  
**Maj la base de données (1)**  
**Ecrire les tests (2)**  
**Tester les perfos (1)**  
...

# Backlog de sprints

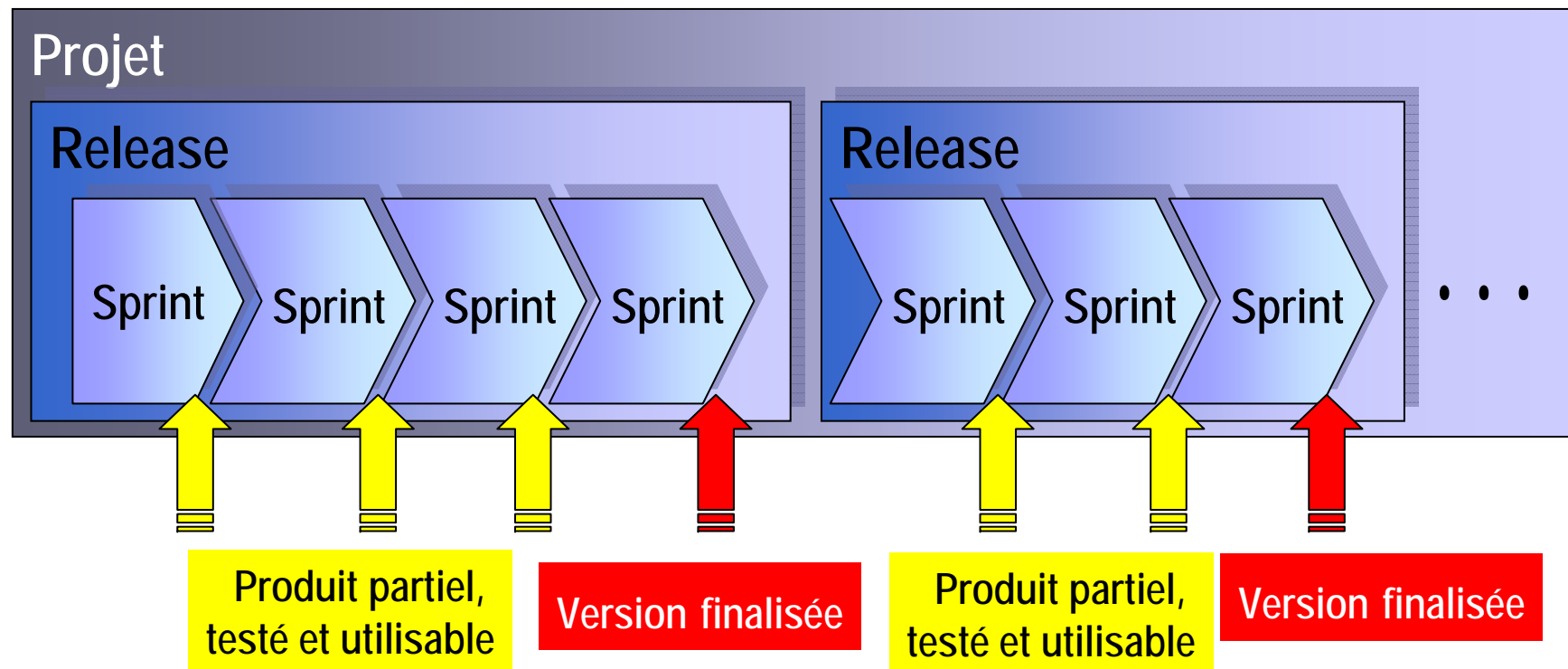
- Une liste de tout ce qui va engendrer du travail
- Chacun s'engage sur le travail qu'il a lui-même choisi (pas de travail imposé)
- Si un travail n'est pas clair, on définit une tâche avec plus de temps et on la décomposera quand on l'appréhendera mieux.



## Exemple de backlog de sprints

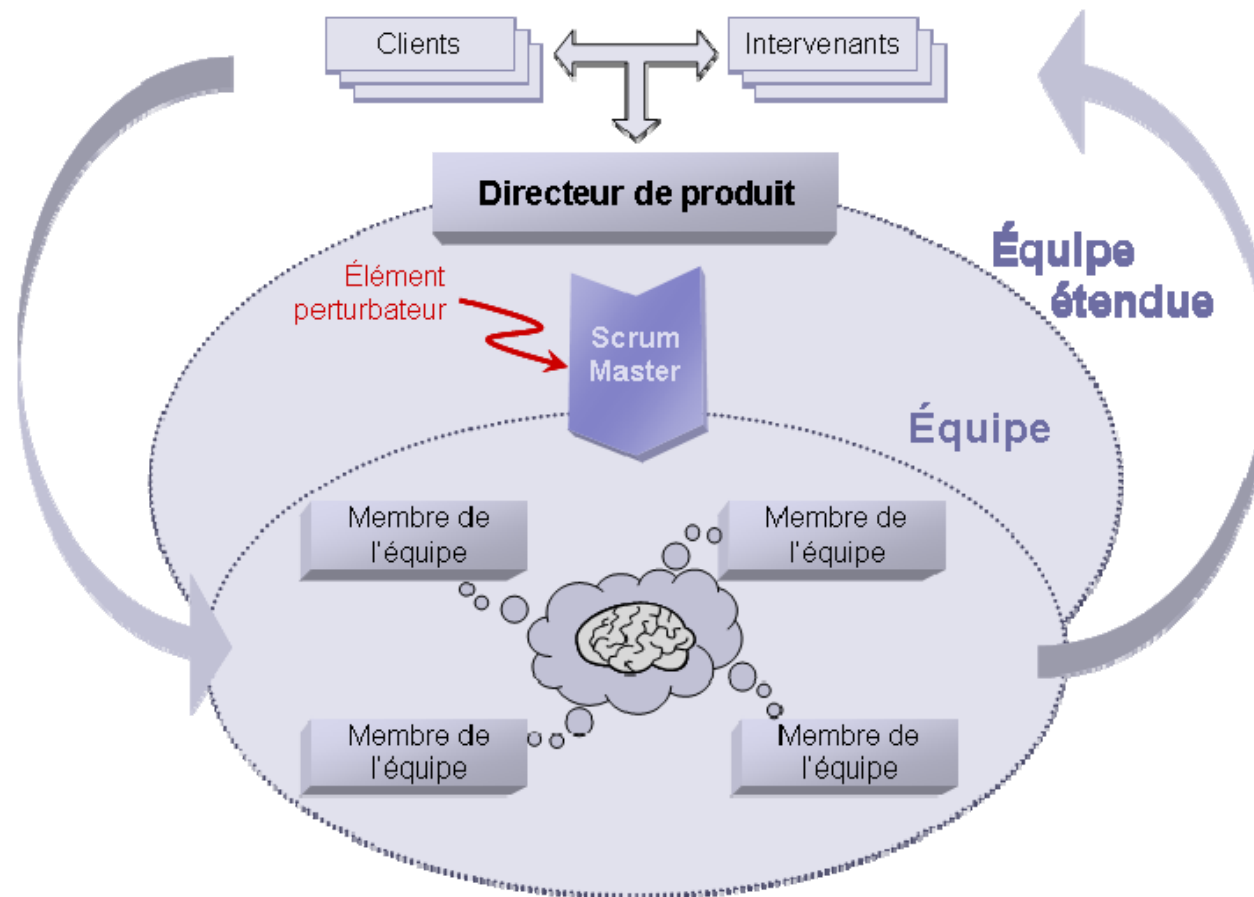
Tâches	Lun	Mar	Mer	Jeu	Ven
Coder l'IHM	8	4	4		
Intégrer serveur XMPP		12	12		
Mettre à jour la BD			8		
Ecrire les tests	8				
Tester les perfos				8	
Analyser les idées				4	4

# Planification à 3 niveaux





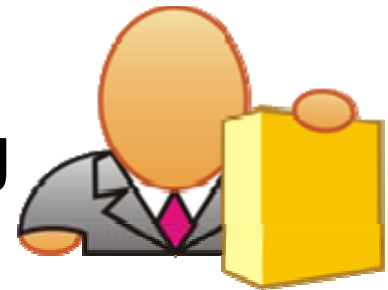
# Organisation



# Product Owner

---

- **Choisit la date et le contenu de la release**
- **Est responsable du retour sur investissement**
- **Définit les priorités dans le backlog en fonction de la valeur « métier »**
- **Ajuste les fonctionnalités et les priorités à chaque sprint si nécessaire**



# Scrum master

---

- **Est responsable de faire appliquer par l'équipe les valeurs et les pratiques de Scrum**
- **Résout des problèmes**
- **S'assure que l'équipe est complètement fonctionnelle et productive**
- **Facilite une coopération poussée entre tous les rôles et fonctions**
- **Protège l'équipe des interférences extérieures**



# L'équipe

---

- Regroupe tous les rôles :  
Architecte, concepteur,  
développeur, spécialiste IHM,  
testeur, etc.
- A plein temps sur le projet, de  
préférence - Exceptions  
possibles (administrateur, ...)
- Organisation autonome



# Daily Scrum

- Réunion organisée tous les jours
- 5 minutes debout
- Pas fait pour résoudre mais plutôt pour identifier les problèmes.



3 questions essentielles :

- qu'est ce que j'ai fais hier?
- qu'est ce que je fais aujourd'hui?
- quels sont les problèmes?

# Sprint Review

---

- Réunion de 4 heures max
- Participation de tous
- Pour valider le logiciel produit pendant le sprint avec une démonstration des nouvelles fonctionnalités ou de l'architecture



# Sprint rétrospective

---

- A chaque fin de Sprint
- Pour réfléchir à ce qui marche et ce qui ne marche pas.
- 15 à 30 minutes pour se réunir et faire un point.



Ce qu'on aimerait

- faire,
- arrêter de faire,
- continuer a faire.

# Burndown chart d'un sprint

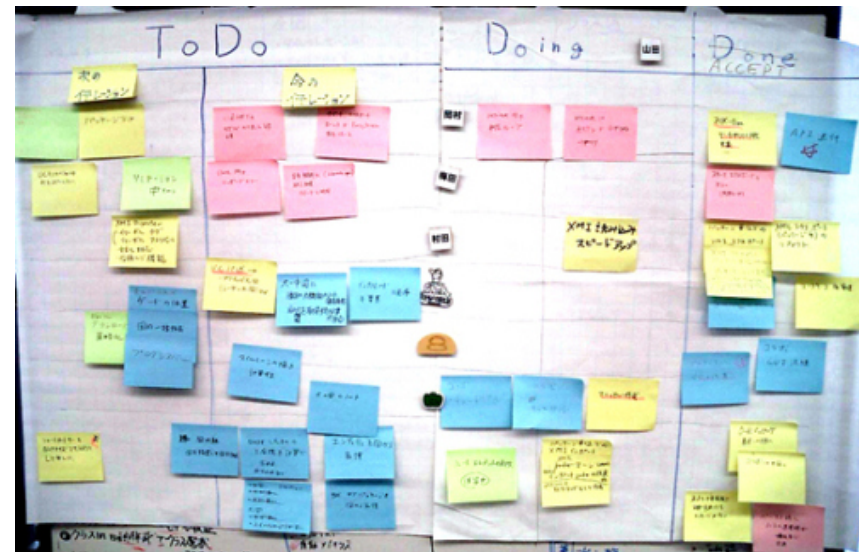




# Scrum board

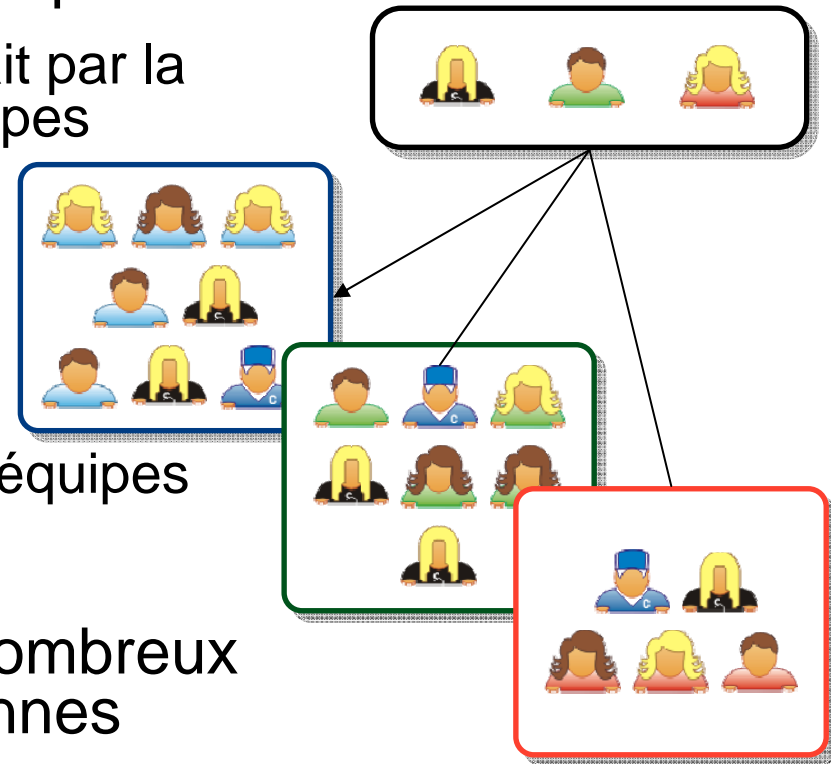
Sous forme de post-it sur un tableau :

- Les tâches à faire
- Les tâches en cours
- Les tâches terminées

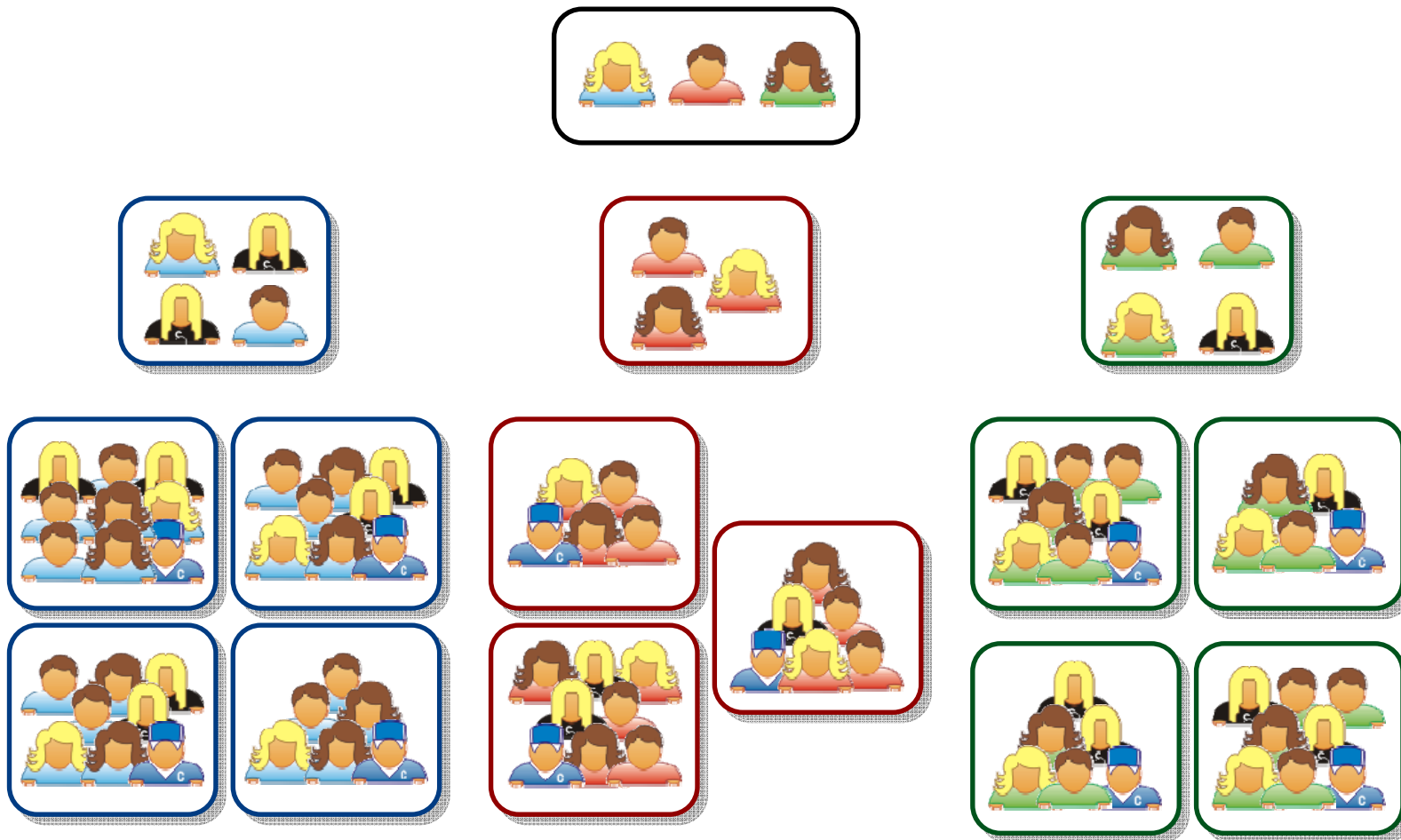


# Scrum à grande échelle

- Une équipe typique c'est  $7 \pm 2$  personnes
  - Le changement d'échelle se fait par la collaboration de plusieurs équipes
- Facteurs dans la scalabilité
  - Type d'application
  - Taille de l'équipe
  - Répartition géographique des équipes
  - Durée du projet
- Scrum a été utilisé pour de nombreux projets de plus de 500 personnes



# Scrum de scrums de scrums



# Pour aller plus loin ...

---

- [www.mountaingoatsoftware.com/scrum](http://www.mountaingoatsoftware.com/scrum)
- [www.scrumalliance.org](http://www.scrumalliance.org)
- [www.controlchaos.com](http://www.controlchaos.com)
- [scrumdevelopment@yahoogroups.com](mailto:scrumdevelopment@yahoogroups.com)
- En français
  - le groupe des utilisateurs de Scrum : [www.frenchsug.org](http://www.frenchsug.org)
  - <http://fr.groups.yahoo.com/group/frenchsug>



# Conclusion

---

**Les méthodes agiles sont séduisantes en théorie, mais il n'y a rien de magique.**

**Pour qu'elles soient bénéfiques, il est nécessaire de :**

- **les maîtriser,**
- **savoir adapter son organisation habituelle,**
- **savoir se remettre en cause,**
- **être créatif,**
- **s'impliquer, avoir le sens des responsabilités.**
- **ne pas vouloir à tout prix suivre des règles et des principes s'ils se révèlent inadaptés,**
- **faire son marché dans différentes méthodes pour satisfaire les spécificités du projet.**
- **repenser la méthode et le processus à chaque projet.**

# Attention: Humour !!!

[www.la\\_rache.com](http://www.la_rache.com)

## La-rache.com

Bienvenue sur le site of the International Institute of La RACHE



- Accueil / News
- Présentation
- La F.A.Q
- Lexique
- Publications
- Consulting
- Formations/Certifications
- En image
- Témoignages
- Liens

LA RACHE

Identifiez-vous

### La méthode R.A.C.H.E :

Rapid  
Application  
Conception and  
Heuristic  
Extreme-programming

La Rache s'appuie sur deux concepts aussi important l'un que l'autre.

D'une part la «Rapid Application Conception » correspond conceptuellement à une accélération importante dans la phase de conception de l'application par rapport aux méthodes classiques. Pour bien débiter avec La RACHE il faut soigner la phase d'étude et la rédaction du cahier des charges. Il faut ici produire un travail de synthèse important en résumant le cahier de charges en un post-it de 8 mots maximum. Puis la mission est d'extrapoler de ce post-it un sujet de développement vaseux, mais pas trop. A partir de là, en règle générale, la multiplication du nombre de mot sur le post-it par un chiffre tiré au sort entre 20 et 200 donne la durée du projet en jours/homme. On prendra soin de ne rien planifier dans cette phase.

D'autre part « L'extrême programming heuristique » est un concept assez prometteur. En effet l'heuristique est une technique consistant à apprendre petit à petit, en tenant compte de ce que l'on a fait précédemment pour tendre vers la solution d'un problème. Opposé à l'algorithmique l'heuristique ne garantit pas du tout qu'on arrive à une solution quelconque en un temps fini. Ceci sous entend d'une part une démarche pédagogique globale d'apprentissage et de capitalisation des acquis, mais aussi que les échéances annoncées le sont dans une pure otique de déconnade symbolique. Et c'est précisément le plus 'produit' de la méthode RACHE.

### Cycle typique d'utilisation de La RACHE

