UML

UML

- ***Unified Modeling Language***
- Langage de modélisation graphique
- Travaux de **Grady Booch**, **James Rumbaugh** et **Ivar Jacobson**

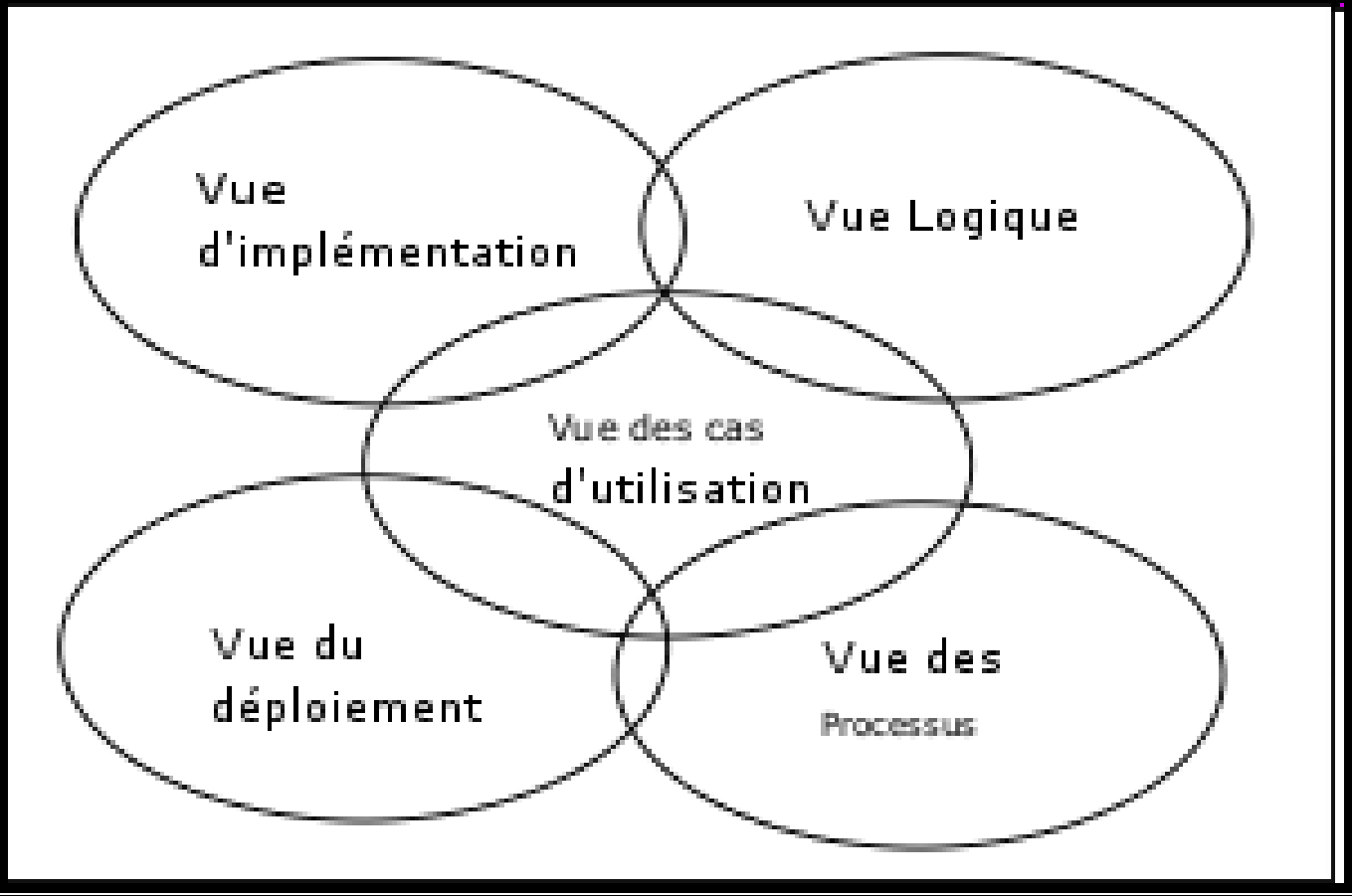
UML 2

- 13 types de diagrammes
- Le diagramme de classes est généralement considéré comme l'élément central d'UML

Mise en œuvre d'une démarche à l'aide d'UML : les vues

- Considérer différentes vues qui peuvent se superposer pour collaborer à la définition du système:
  - Vue des ***cas d'utilisation*** : description du modèle « vue » par les acteurs du système.
  - Vue ***logique*** : définition du système vu de l'intérieur.
  - Vue ***d'implémentation*** : définit les dépendances entre les modules.
  - Vue des ***processus*** : la vue temporelle et technique.

UML Vues


Vue
d'implémentation
Vue Logique
Vue des cas
d'utilisation
Vue du
déploiement
Vue des
Processus

Diagrammes UML

- Diagrammes structurels ou statiques
- Diagrammes comportementaux
- Diagrammes d'interaction ou dynamiques

Diagrammes structurels ou statiques

- **Diagramme de classes** (cf. Class diagram) : représente les classes intervenant dans le système.
- **(Diagramme d'objets** (cf. Object diagram) : sert à représenter les instances de classes (objets) utilisées dans le système.)
- **Diagramme de déploiement (cf. Deployment diagram) :** sert à représenter les éléments matériels et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux.

Diagramme de classes

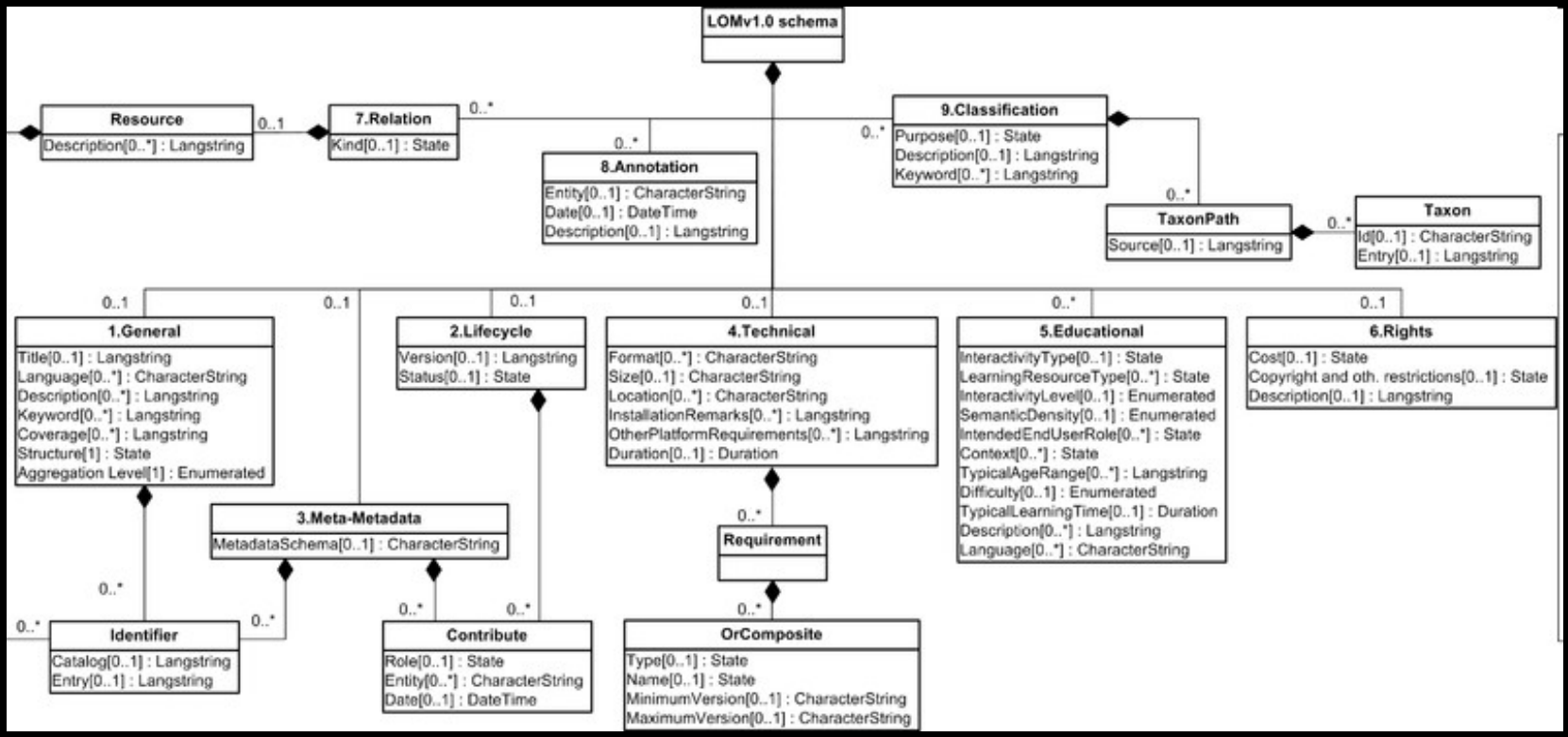
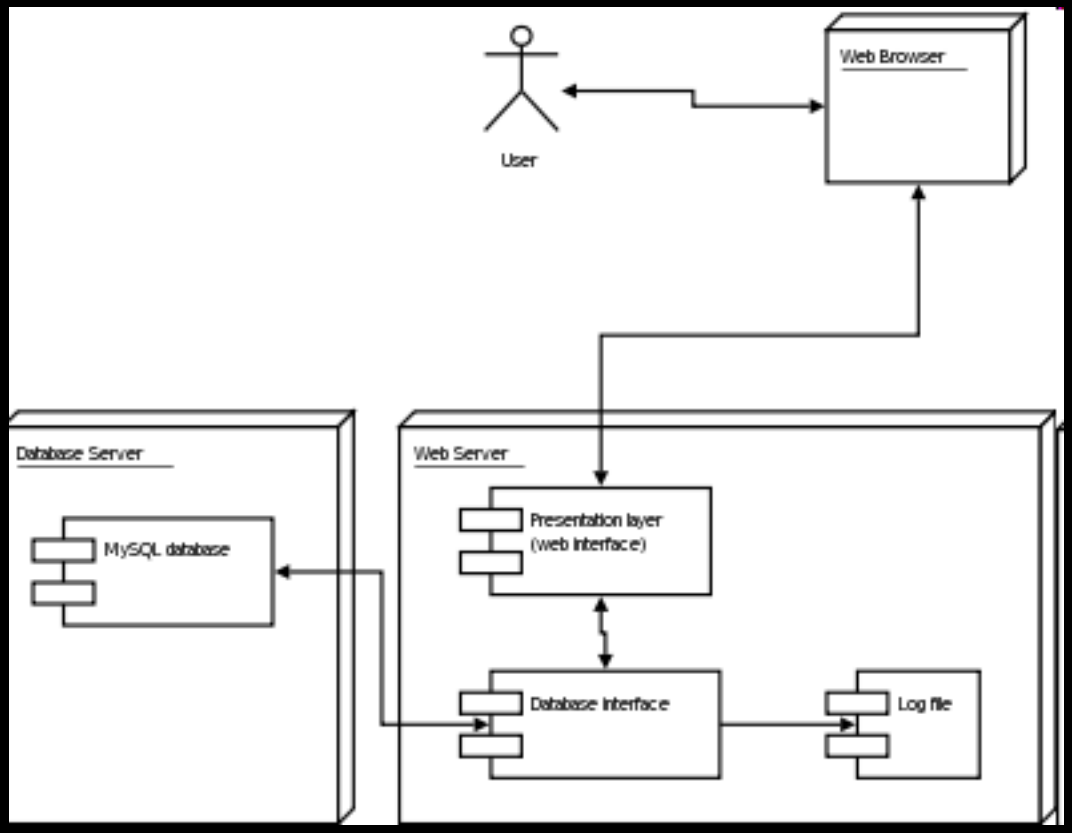


Diagramme de déploiement



Diagrammes comportementaux

- **Diagramme des cas d'utilisation** (*use-cases*) (cf. Use Case Diagram) : il permet d'identifier les possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système), c'est-à-dire toutes les fonctionnalités que doit fournir le système.
- **Diagramme états-transitions** (cf. State Machine Diagram) : permet de décrire sous forme de machine à états finis le comportement du système ou de ses composants.
- **Diagramme d'activité** (cf. Activity Diagram) : permet de décrire sous forme de flux ou d'enchaînement d'activités le comportement du système ou de ses composants.

Diagramme des cas d'utilisation

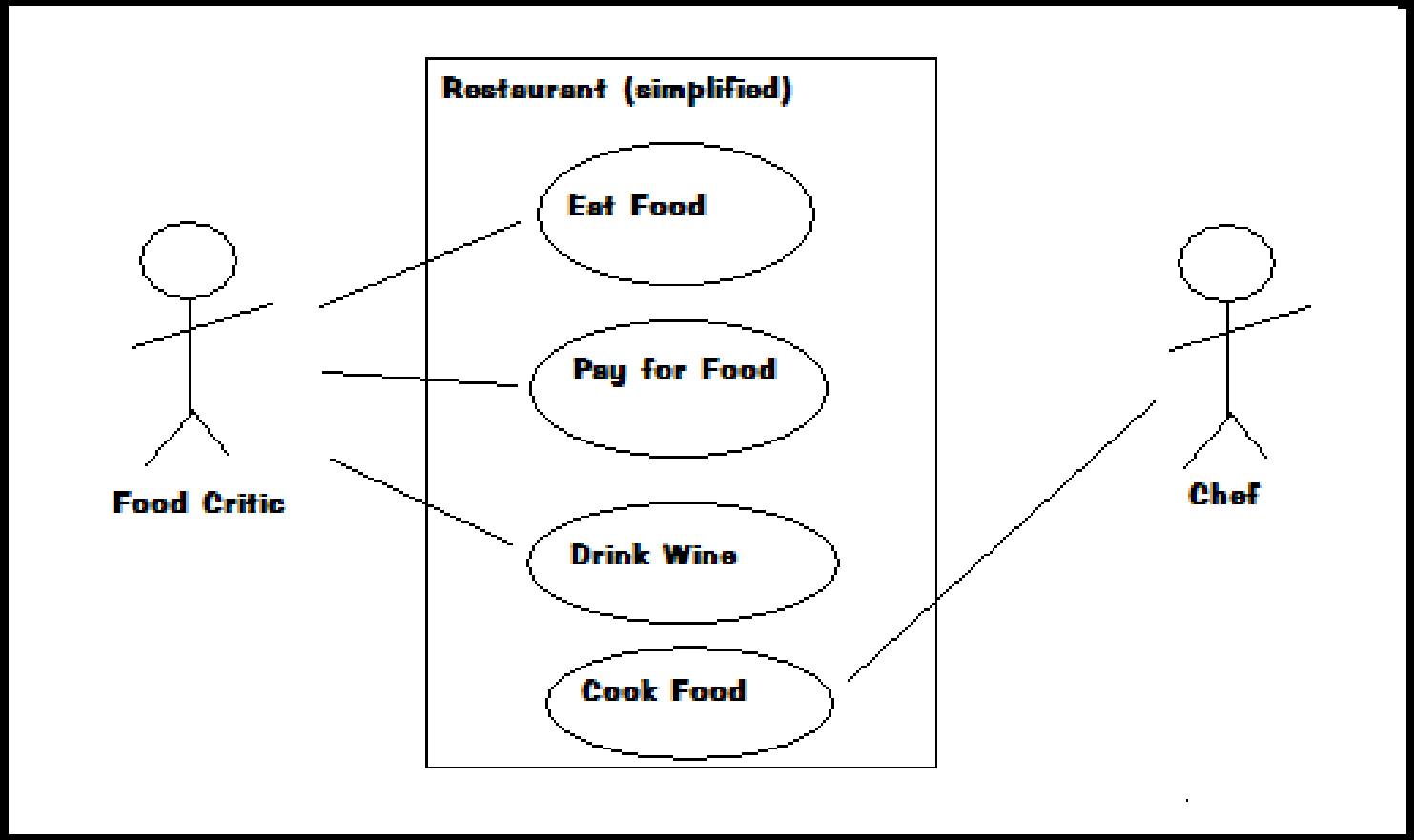


Diagramme états-transitions

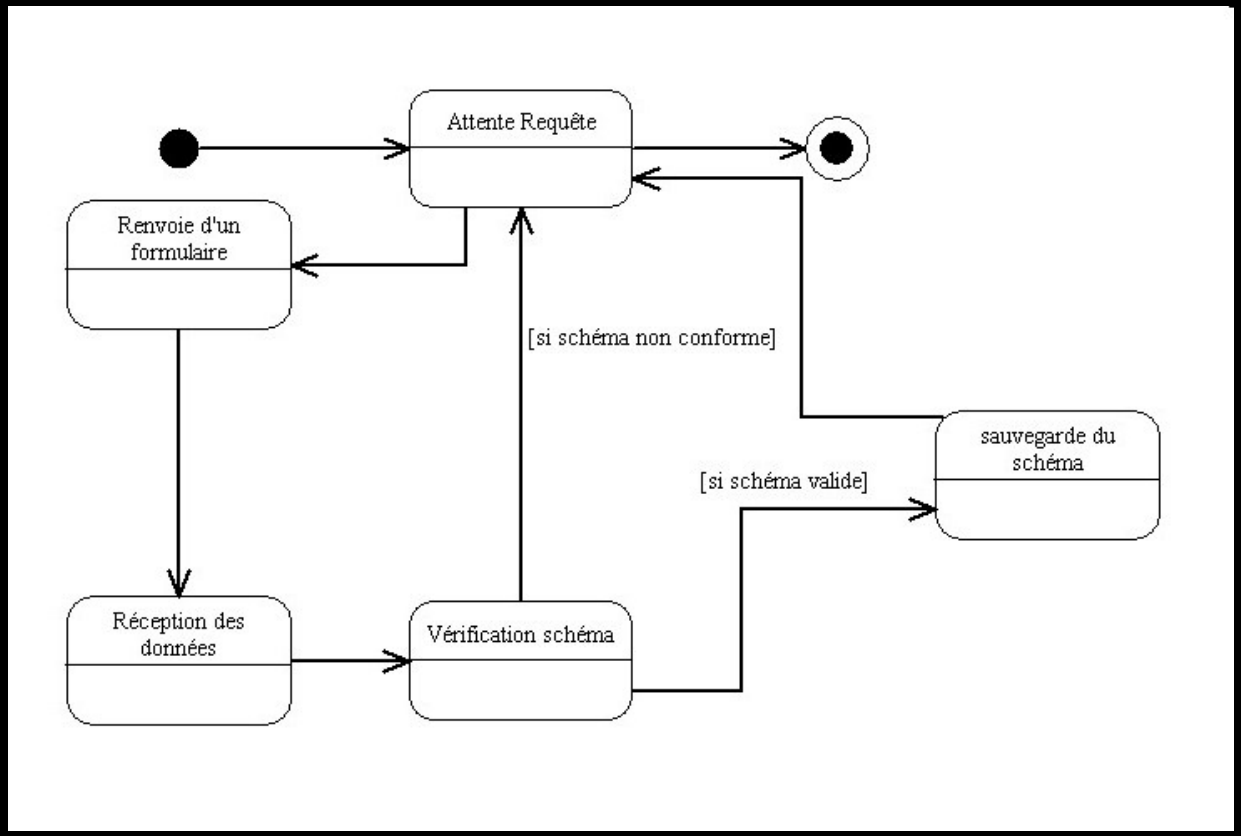
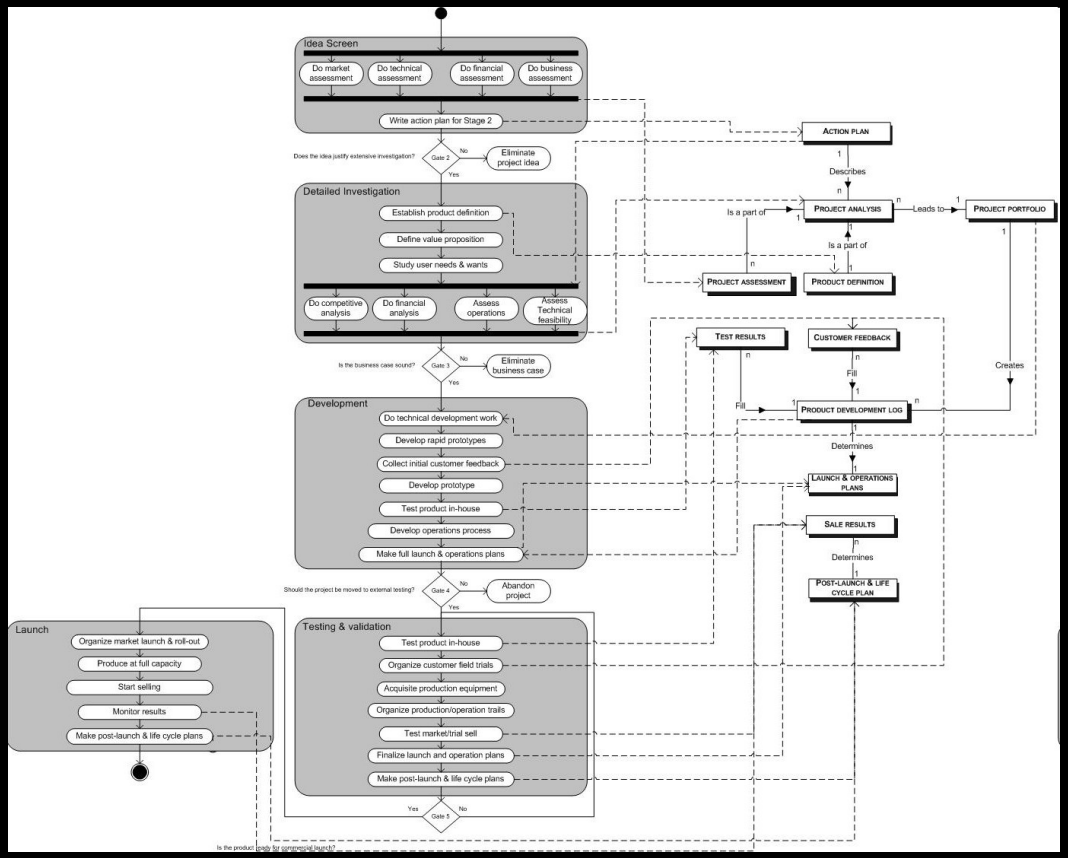


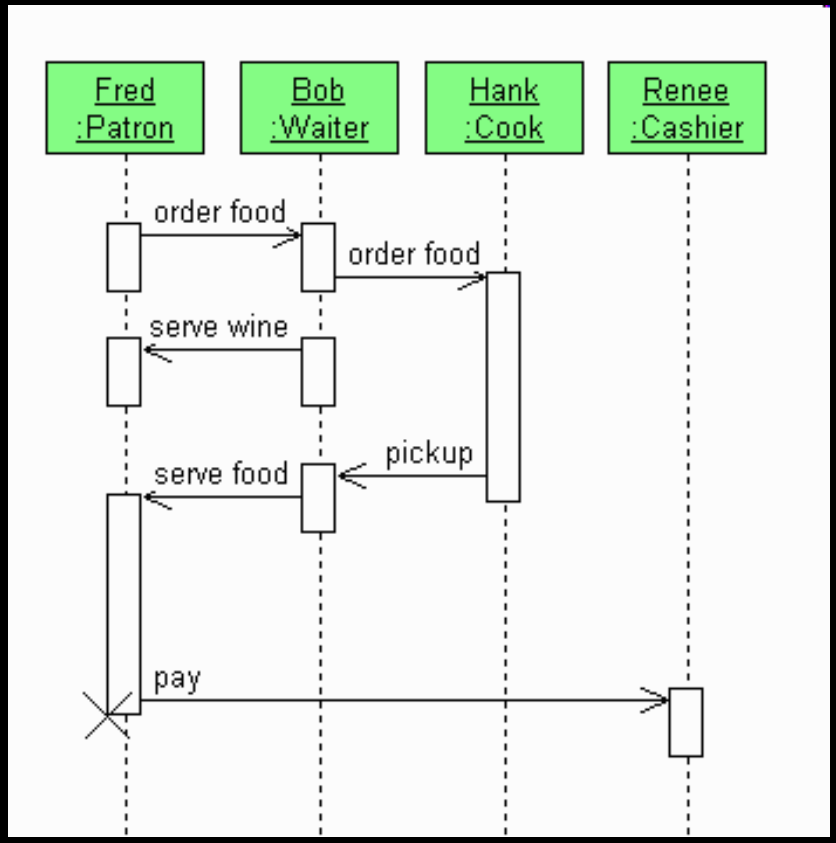
Diagramme d'activité



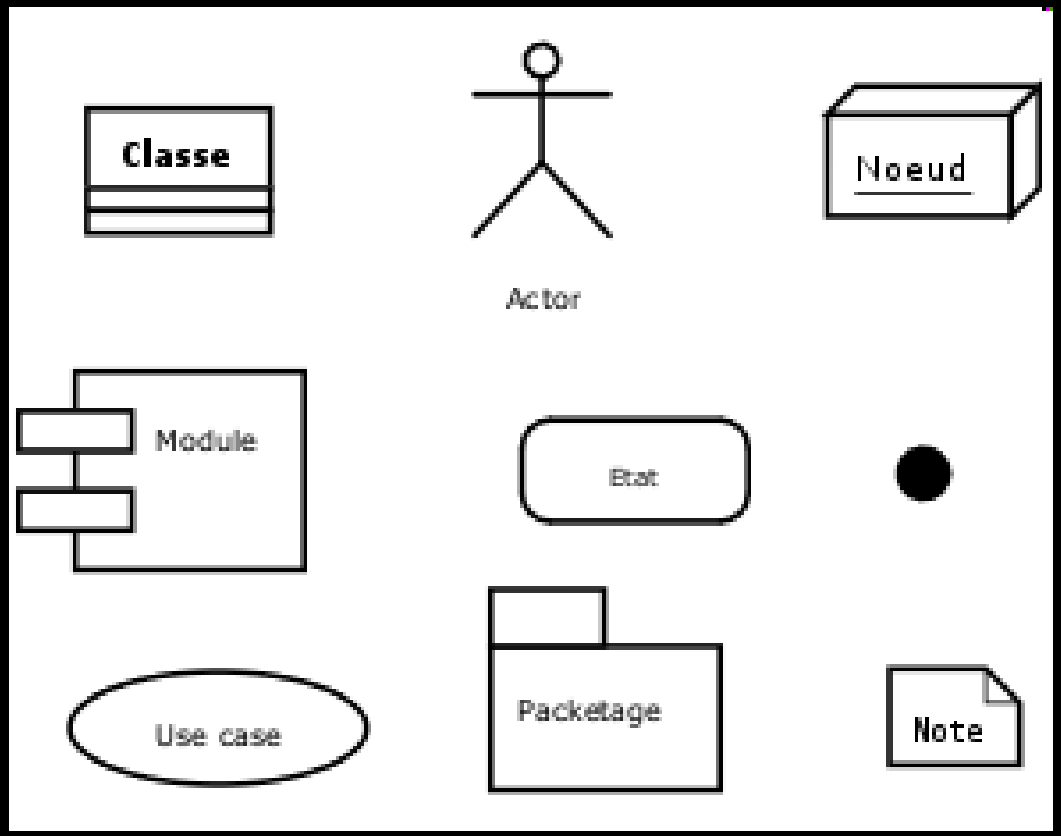
Diagrammes d'interaction ou dynamiques

- **Diagramme de séquence** (cf. Sequence Diagram) : représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.
- **Diagramme de communication** (cf. Communication Diagram) : représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.

Diagramme de séquence



Les éléments de modélisation



Méthodes Agiles

Cycles de développement

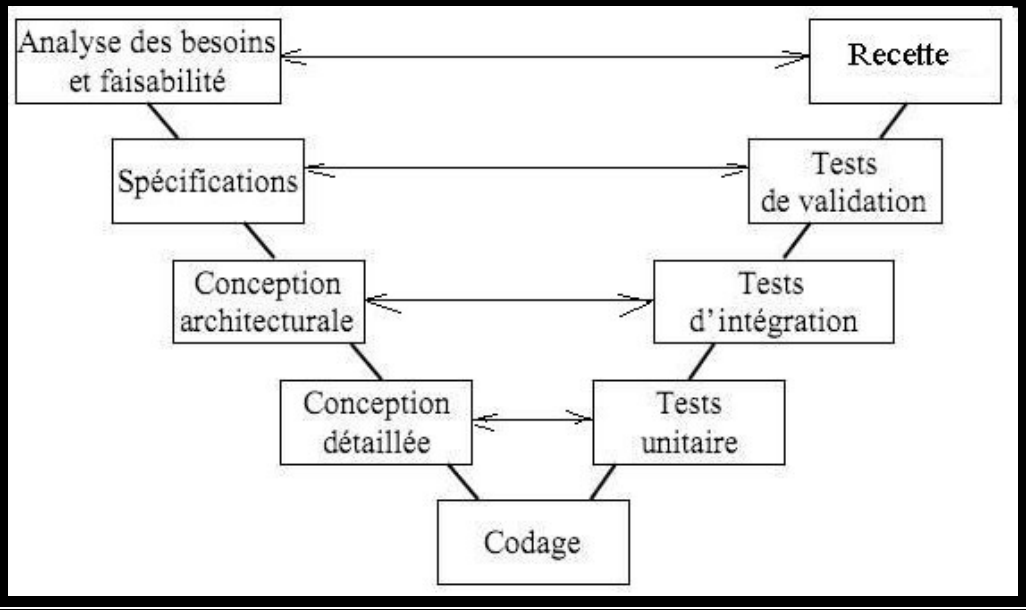
- Toutes les étapes de la conception d'un logiciel
- **Les grandes familles:**
  - **Modèle en cascade**
  - **Cycle en V**
  - **Cycle en spirale**
  - **Cycle semi itératif**
  - **Cycle itératif**

Modèle en cascade

- Les phases traditionnelles de développement sont effectuées les unes après les autres avec un retour sur les précédentes.
- Le processus de développement exécute des phases qui ont pour caractéristiques :
  - produire des livrables définis au préalable ;
  - se terminer à une date précise ;
  - ne se terminer que lorsque les livrables sont jugés satisfaisants lors d'une étape de validation-vérification.

Cycle en V

- Pallier le problème de réactivité du modèle en cascade
- Permet en cas d'anomalie, de limiter un retour aux étapes précédentes.



Cycle en spirale

- Reprend les différentes étapes du cycle en V.
- Implémentation de versions successives
- Le début de chaque itération comprend une phase d'analyse des risques

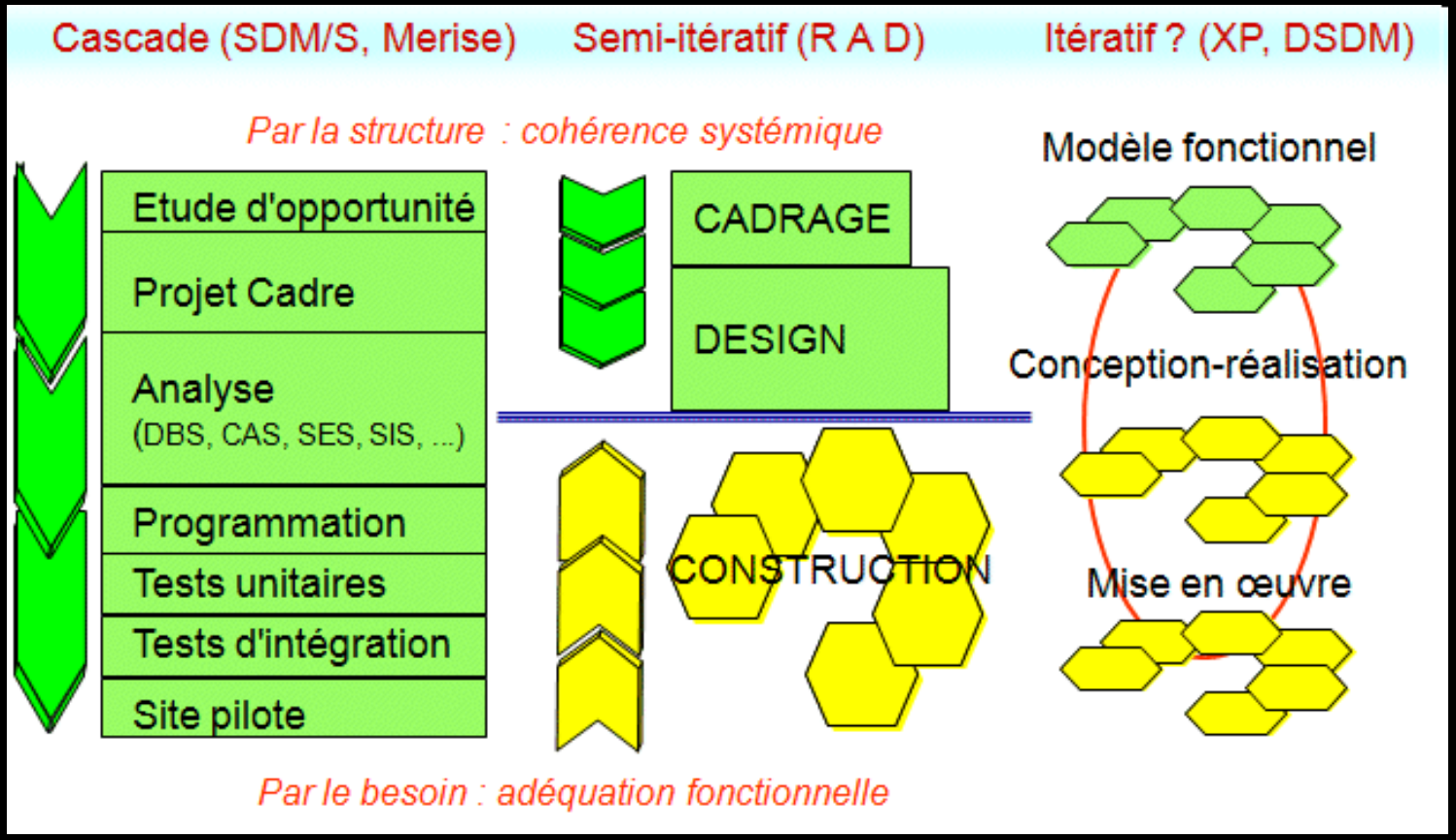
Cycle semi itératif

- Travaux de James Martin publiés à partir de 1989
- Totalement formalisé en 1991 dans le livre RAD (Développement rapide d'applications)
- C'est vers 2001 avec l'apparition de plusieurs méthodes que la vision s’uniformise dans le cadre du Manifeste Agile (Agile Manifesto) et de l'Agile Alliance

Cycle semi itératif

- Les deux premières phases classiques (top down, par la structure) consistent en l'expression des besoins et la conception de la solution.
- Lors de la troisième et dernière grande phase, la construction du produit (bottom up, par le besoin)) la notion d'itérations courtes intervient

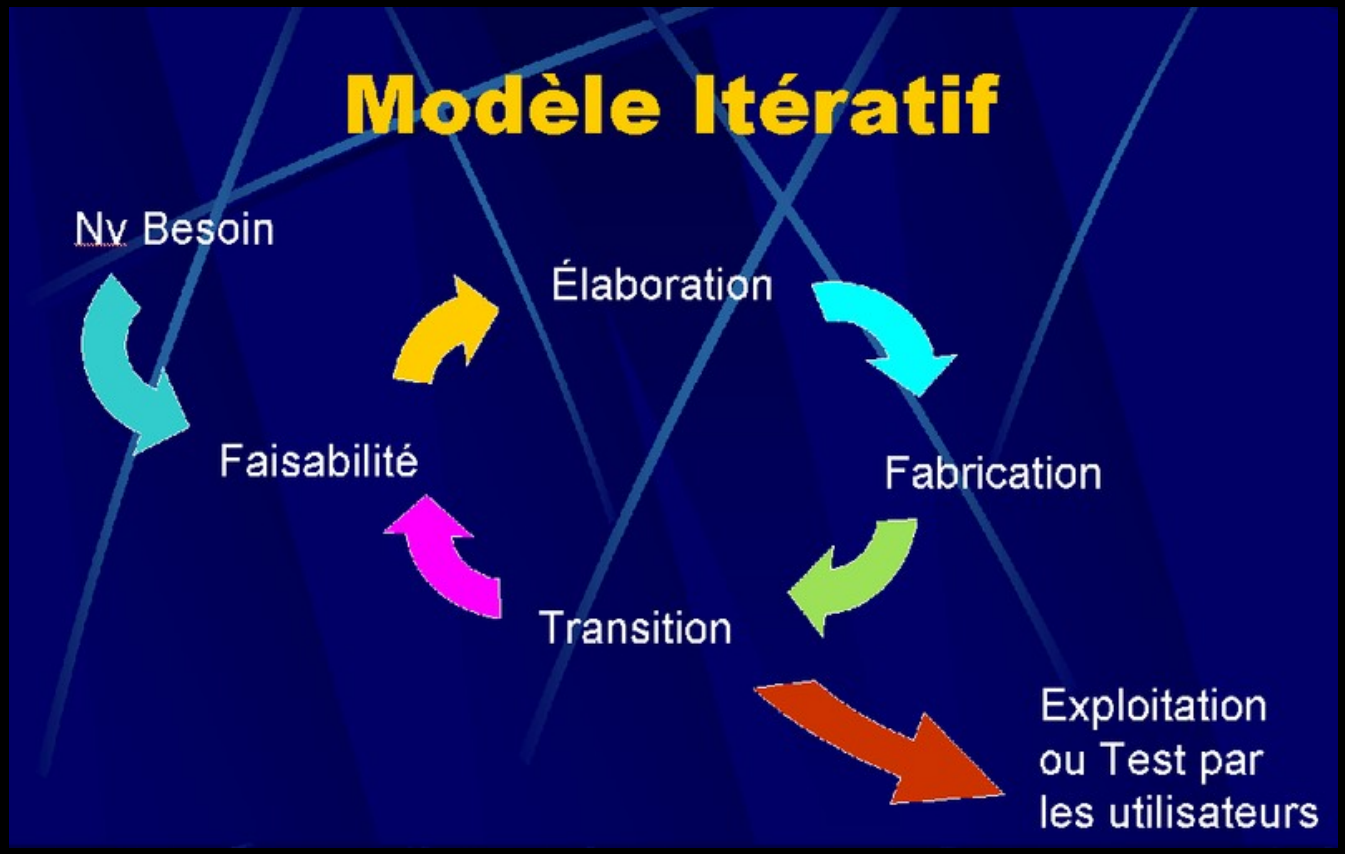
Cycles de développement



Cycle itératif

- Artéfact: le produit issu d'une activité
- Phase:
  - la faisabilité : l'acceptation d'un nouveau besoin
  - l'élaboration : on imagine comment on va le réaliser
  - la fabrication : construction
  - la transition : tout est mis en œuvre pour livrer au client
- Chaque itération ne dépasse **jamais** huit semaines

Cycle itératif



Cycles de développement

- Toutes les étapes de la conception d'un logiciel
- **Les grandes familles:**
  - **Modèle en cascade**
  - **Cycle en V**
  - **Cycle en spirale**
  - **Cycle semi itératif**
  - **Cycle itératif**

Méthodes agile

- Procédures de conception de logiciel impliquant au maximum le demandeur
- Grande réactivité aux demandes
- Visent la satisfaction réelle du besoin du client, et non des termes du contrat de développement.
- Officialisée en 2001 par un document: Manifeste Agile

Méthodes agile: valeurs

- **L'équipe** (« Personnes et interaction plutôt que processus et outils »)
- L'équipe est bien plus importante que les outils (structurants ou de contrôle) ou les procédures de fonctionnement.
- Préférable d'avoir une équipe soudée et qui communique composée de développeurs (éventuellement à niveaux variables) plutôt qu'une équipe composée d'experts fonctionnant chacun de manière isolée.
- La communication est une notion fondamentale.

Méthodes agile: valeurs

- **L'application** (« Logiciel fonctionnel plutôt que documentation complète »)
- Il est vital que l'application fonctionne.
- Une documentation précise est utile comme moyen de communication.
- La documentation représente une charge de travail importante, mais peut pourtant être néfaste si elle n'est pas à jour.
- Il est préférable de commenter abondamment le code lui-même, et surtout de transférer les compétences au sein de l'équipe (on en revient à l'importance de la communication).

Méthodes agile: valeurs

- **La collaboration** (« Collaboration avec le client plutôt que négociation de contrat »)
- Le client doit être impliqué dans le développement.
- Le client doit collaborer avec l'équipe et fournir un feed-back continu sur l'adaptation du logiciel à ses attentes.

Méthodes agile: valeurs

- **L'acceptation du changement** (« Réagir au changement plutôt que suivre un plan »)
- La planification initiale et la structure du logiciel doivent être flexibles afin de permettre l'évolution de la demande du client tout au long du projet.
- Les premières *releases* du logiciel vont souvent provoquer des demandes d'évolution.

Méthodes Agiles

- Scrum (1996)
- Extreme programming (XP, 1999)
- Crystal clear (2004)

Scrum

- Racines de Scrum se retrouvent dans la publication de Takeuchi et Nonaka dans "The New New Product Development Game“ 1986
- L'aspect métaphore du rugby
- Ne couvre aucune technique d'ingénierie du logiciel
- Nécessite de lui adjoindre une méthode complémentaire

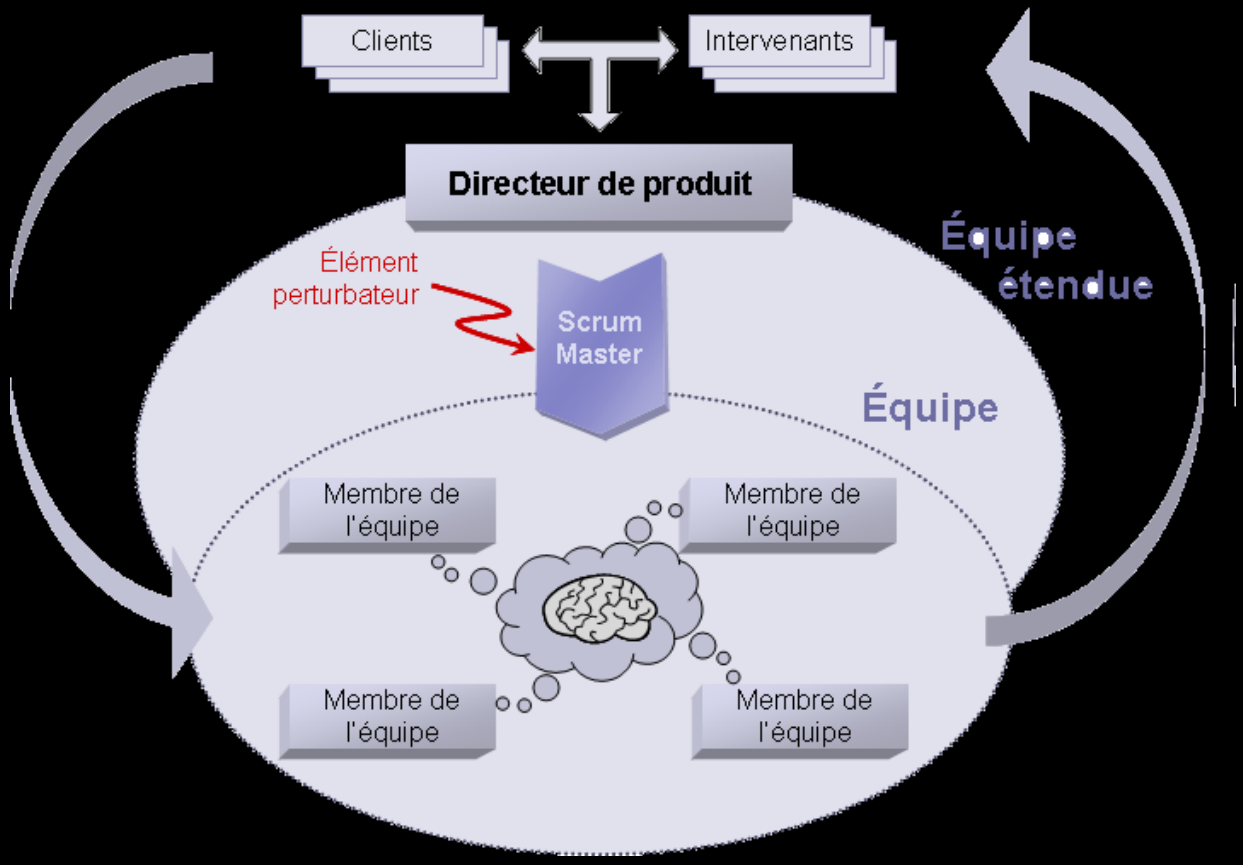
Scrum

- *Mêlée*
- Processus s'articule en effet autour d'une équipe soudée, qui cherche à atteindre un but
- Focaliser l'équipe de façon itérative sur un ensemble de fonctionnalités à réaliser, dans des itérations de durée fixe de une à quatre semaines, appelées **sprints**.

Scrum

- Chaque sprint possède un **but** à atteindre, défini par le *directeur de produit*, à partir duquel sont choisies les fonctionnalités à implémenter dans ce sprint.
- Un sprint aboutit toujours sur la livraison d'un produit partiel fonctionnel.
- Le *ScrumMaster* a la charge de réduire au maximum les perturbations extérieures et de résoudre les problèmes non techniques de l'équipe.

Scrum


Clients
Intervenants
Directeur de produit
Équipe étendue
Élément perturbateur
Scrum Master
Équipe
Membre de l'équipe
Membre de l'équipe
Membre de l'équipe
Membre de l'équipe

Directeur de produit

- Le **directeur de produit** (*Product Owner*) est le représentant des clients et utilisateurs.
- Définit l'**ordre** dans lequel les fonctionnalités seront développées et qui prend les décisions importantes concernant l'orientation du projet.
- Dans l'idéal, le directeur de produit travaille dans la même pièce que l'équipe. Il est important qu'il reste très disponible pour répondre aux questions de l'équipe et pour lui donner son avis sur divers aspects du logiciel (interface par exemple).

Équipe

- L'**équipe** est **auto-gérée**.
- Il n'y a pas non plus de notion de hiérarchie interne : toutes les décisions sont prises ensemble et personne ne donne d'ordre à l'équipe sur sa façon de procéder. Contrairement à ce que l'on pourrait croire, les équipes auto-gérées sont celles qui sont les plus efficaces et qui produisent le meilleur niveau de qualité de façon spontanée.
- L'équipe s'adresse directement au directeur de produit et lui montre le plus souvent possible le logiciel développé.

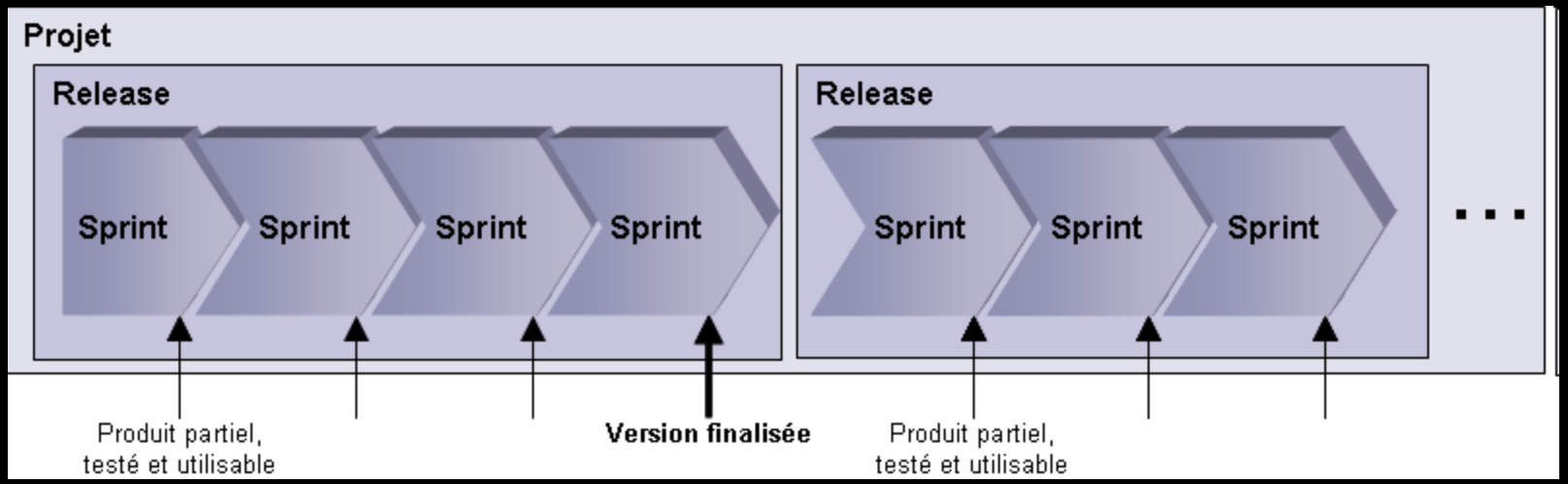
ScrumMaster

- Chargé de protéger l'équipe de tous les éléments perturbateurs extérieurs à l'équipe et de résoudre ses problèmes non techniques (administratifs par exemple).
- **Equipe étendue** intègre en plus le ScrumMaster et le directeur de produit.

Planification

- Planification à trois niveaux : release/projet, sprint et quotidien
- Les itérations (sprints) durent généralement entre 2 et 4 semaines.
- Chaque sprint possède un **but** et on lui associe une liste d'*items de backlog de produit* (fonctionnalités) à réaliser.
- Ces items sont décomposés par l'équipe en tâches élémentaires de quelques heures, les *items de backlog de sprint*.

Planification



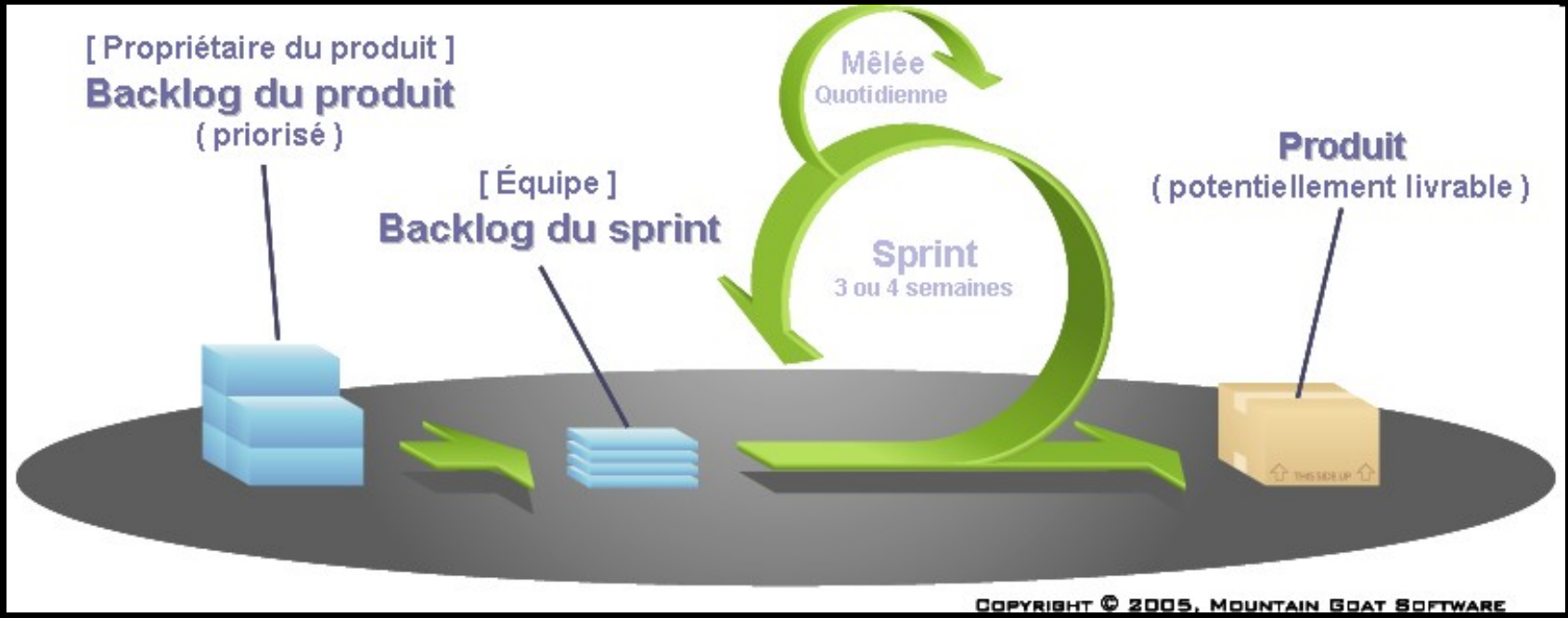
Gestion des besoins

- Backlog de produit: liste de fonctionnalités à réaliser
- Backlog de sprint: quels items du backlog de produit seront réalisés dans ce sprint.

Déroulement d'un sprint

- **Réunion de planification** (*Sprint Planning*) (<4h) consiste à définir d'abord un but pour le sprint, puis à choisir les items de backlog du produit qui seront réalisés dans ce sprint
- Chaque journée de travail commence par une réunion de 15 minutes maximum appelée **mêlée quotidienne** (*Daily Scrum*):
  - *Qu'est-ce que j'ai fait hier ?*
  - *Qu'est-ce que je compte faire aujourd'hui ?*
  - *Quelles difficultés est-ce que je rencontre ?*
- **Revue de sprint** (<4h)
- **Rétrospective du sprint**

Scrum


[Propriétaire du produit]
Backlog du produit
(priorisé)
[Équipe]
Backlog du sprint
Mêlée
Quotidienne
Sprint
3 ou 4 semaines
Produit
(potentiellement livrable)
Copyright © 2005, Mountain Goat Software

Extreme programming (XP)

- Inventée par Kent Beck, Ward Cunningham et Ron Jeffries pendant leur travail sur un projet « C3 » de calcul des rémunérations chez Chrysler.
- Née officiellement en octobre 1999 avec le livre *Extreme Programming Explained* de Kent Beck.

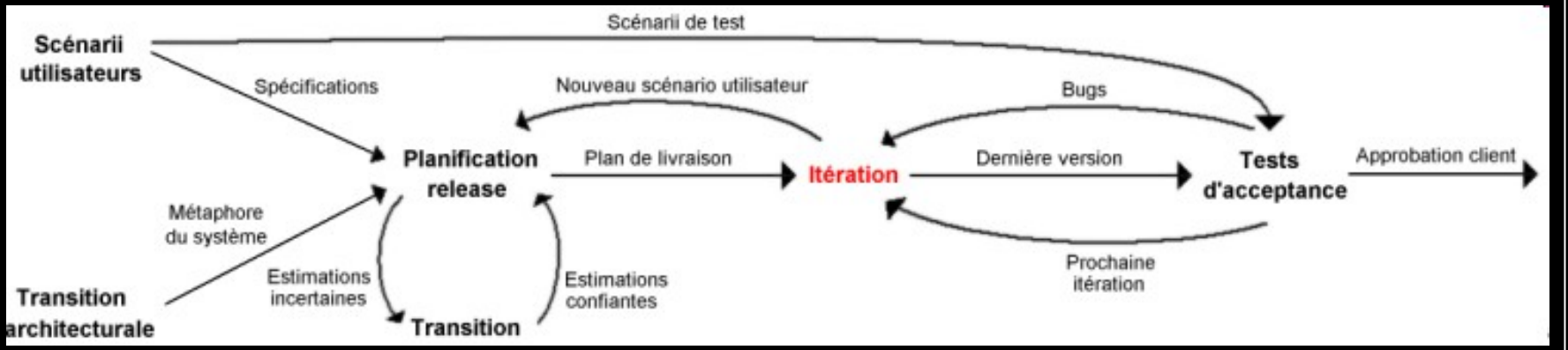
XP est (Kent Beck)…

- une tentative de réconcilier l´humain avec la productivité;
- un mécanisme pour faciliter le changement social;
- une voie d'amélioration;
- un style de développement;
- une discipline de développement d ´applications informatiques.

XP

- But principal est de réduire les coûts du changement
- L'Extreme Programming repose sur des cycles rapides de développement:
  - une phase d'exploration détermine les scénarios clients qui seront fournis pendant cette itération ;
  - l'équipe transforme les scénarios en tâches à réaliser et en tests fonctionnels ;
  - chaque développeur s'attribue des tâches et les réalise avec un binôme ;
  - lorsque tous les tests fonctionnels passent, le produit est livré.

XP


Scénarii de test
Scénarii utilisateurs
Spécifications
Nouveau scénario utilisateur
Bugs
Planification release
Plan de livraison
Itération
Dernière version
Tests d'acceptance
Approbation client
Métaphore du système
Estimations incertaines
Estimations confiantes
Prochaine itération
Transition architecturale
Transition

Crystal Clear

- Alistair Cockburn
- *Crystal Clear, A Human-Powered Methodology for Small Teams*, Alistair Cockburn, October 2004, pages 336, paperback, Addison-Wesley Professional, ISBN 0-201-69947-8

Crystal Clear

- La communication est omniprésente pour réussir le « jeu coopératif »
- Le nombre de membres d'une équipe est limité à six personnes afin que l'équipe soit solidaire.
- Tous les membres de l'équipe doivent travailler dans une même pièce afin de faciliter la communication par proximité.
- Les schémas de modélisation doivent être réalisés en groupe et sur tableau blanc car cela améliore la communication et la collaboration.
- La collaboration avec le client est elle aussi très importante, notamment grâce à de nombreuses conversations entre utilisateurs et développeurs.
- Livrer des parties exécutables de l'application le plus fréquemment possible afin que le client se rende compte du travail en cours et propose des changements.

Crystal Clear

- Différentes étapes :
  - La spécialisation consiste à observer les utilisateurs dans leur travail pour mieux connaître leurs besoins et leur environnement. Ensuite, les différents cas d'utilisation sont classés par ordre de priorité en collaboration avec les utilisateurs, ce qui permet de savoir quelles fonctionnalités ont le plus de valeur et doivent être développées en premier.
  - Une ébauche de conception est réalisée au tout début du projet, cela inclut les choix des technologies à utiliser et implique une ébauche d'architecture.
  - Le planning consiste à prévoir vers quelles dates les itérations vont se suivre, il est recommandé de définir des itérations d'une longueur de 2 à 3 mois, chacune produisant un produit à livrer fonctionnel.