Projet et problématique de cadrage
Use Case et méthode AGILE

Présentation de
pratiques de cadrage
qui vont dans le bon sens...

Quelques faits…

- 84% des projets en développement ne tiennent pas leurs plannings
- 64%, le pourcentage de features développées mais non utilisées dans le produit final
- 66% des projets dépassent de manière significative les budgets initiaux
- 20 à 40% des requis initiaux changent durant un développement
- 40% le temps passé par l'équipe à refaire

(Source: Standish Group, 2005 chaos report - J.Johnson, Keynote speech,2002 – Empirical software Engineering by K. molokken,2003)

Des raisons ?

Un cahier des charges trop rigide

Un projet qui dure

Alors que le besoins exprimés
sont changeants par nature

Des raisons ?

Mauvaise communication

Entre développeurs

Entre les utilisateurs finaux et les développeurs

Entre les utilisateurs finaux et les commanditaires

Avec les décideurs, responsable financier...

Une solution ?

Adopter une démarche AGILE

La Philosophie AGILE (Agile Manifesto)

Le Manifeste AGILE a été écrit en 2001 par 17 spécialistes du domaine du développement logiciel. Le manifeste commence par la phrase suivante :

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

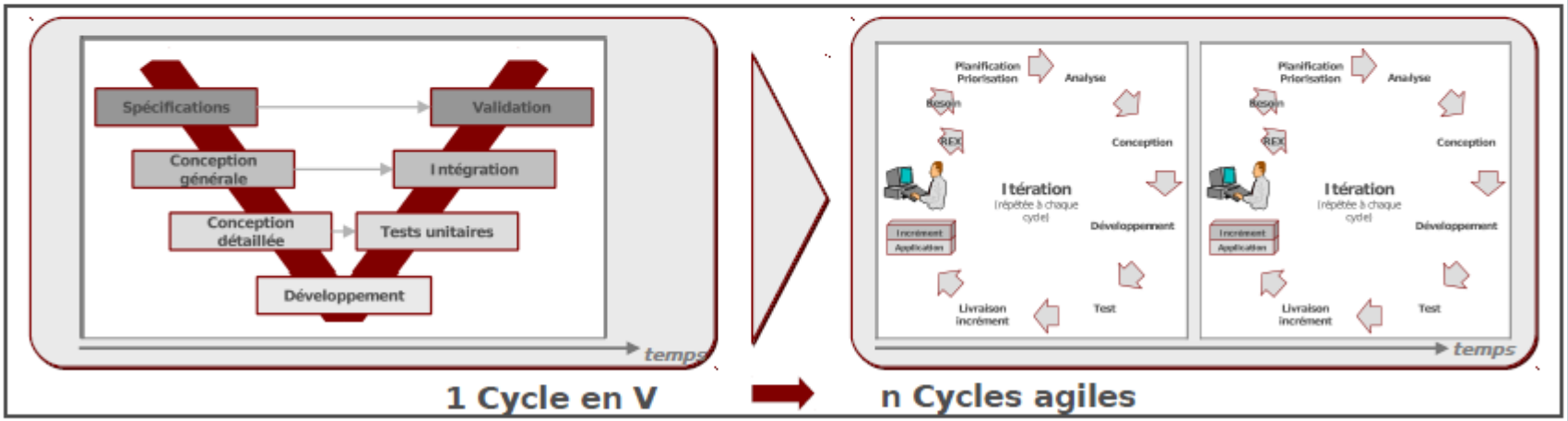
AGILE : 4 valeurs fondamentales…

Ces 4 valeurs visent à valoriser :

- **Les individus et leurs interactions** plus que le processus et les outils
- **Des logiciels opérationnels** plus qu'une documentation exhaustive
- **La collaboration avec le client** plus que la négociation contractuelle
- **L'adaptation au changement** plus que le suivi d'un plan

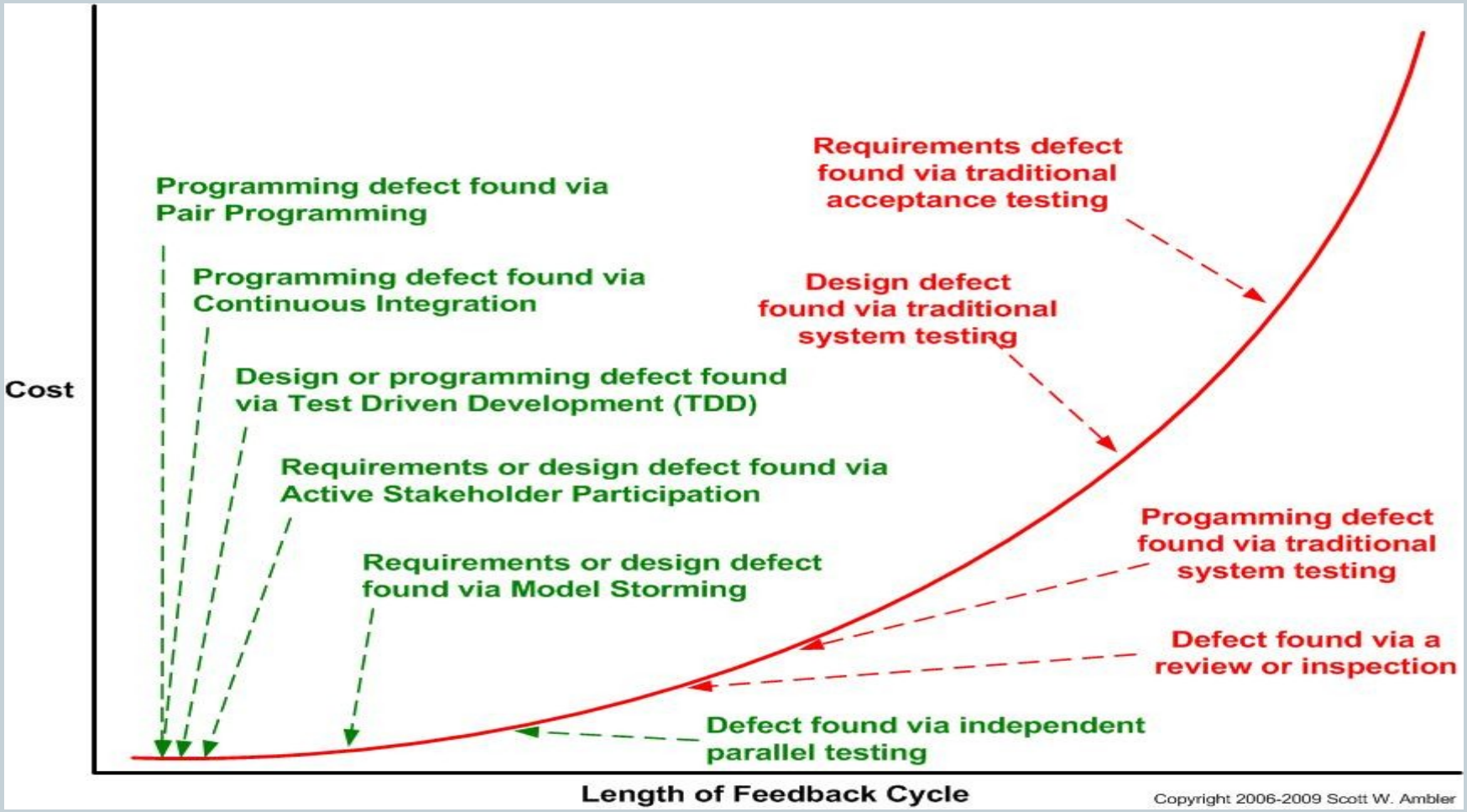
AGILE : Schéma méthodo (source www.cereza.fr/)

- Les méthodologies agiles visent à réaliser des développements informatiques dans un système où les acteurs tendent vers une **collaboration mutuellement profitable**, par opposition à un système concurrentiel.
- Les différentes étapes du développement traditionnel s'établissent alors sur du court terme pour **plus de souplesse et de réactivité**, c'est le principe de « l'agilité ».
- Ces **méthodes itératives** sont simplement la contraction des méthodes en cascade traditionnelles, on y retrouve les spécifications - sous une forme plus proche du besoin - puis les phases de développement, de test et finalement la livraison.
- Dans ces projets, le client et le fournisseur font équipe dans l'accomplissement des objectifs.
- Scrum est la méthode agile la plus utilisée, elle propose une structure simple et adaptée pour implémenter une solution agile complète.



Les méthodes agiles offrent une **alternative au cycle en V standard** en proposant un fonctionnement itératif et collaboratif

AGILE VS TRADITION



Et 12 principes

- Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
- Accueillez positivement les changements de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client.
- Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.

Les Principes

- Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet.
- Réalisez les projets avec des personnes motivées. Fournissez-leur l’environnement et le soutien dont ils ont besoin et faites-leur confiance pour atteindre les objectifs fixés.
- La méthode la plus simple et la plus efficace pour transmettre de l’information à l'équipe de développement et à l’intérieur de celle-ci est le dialogue en face à face.

Les Principes

- Un logiciel opérationnel est la principale mesure d’avancement.
- Les processus Agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.
- Une attention continue à l'excellence technique et à une bonne conception renforce l’Agilité.

Les Principes

- La simplicité – c’est-à-dire l’art de minimiser la quantité de travail inutile – est essentielle.
- Les meilleures architectures, spécifications et conceptions émergent d'équipes auto organisées.
- À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

Quiz

- Un petit quiz…
- Remplir les mots manquants dans les phrases
- Le manifeste Agile établie les valeurs suivantes :

- **Les individus et leurs _____** plus que les processus et les _____.
- **Des _____ opérationnels** plus qu'une _____ exhaustive.
- **La _____ avec le client** plus que la négociation _____.
- **_____ au** changement plus que le suivi d'un _____.

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

Résumons….

Le développement Agile, appelé aussi développement adaptatif, se caractérise donc par un style de conduite de projet itératif incrémental, centré sur l'autonomie des ressources humaines impliquées dans la spécification, la production et la validation d'une application intégrée et testée en continu.

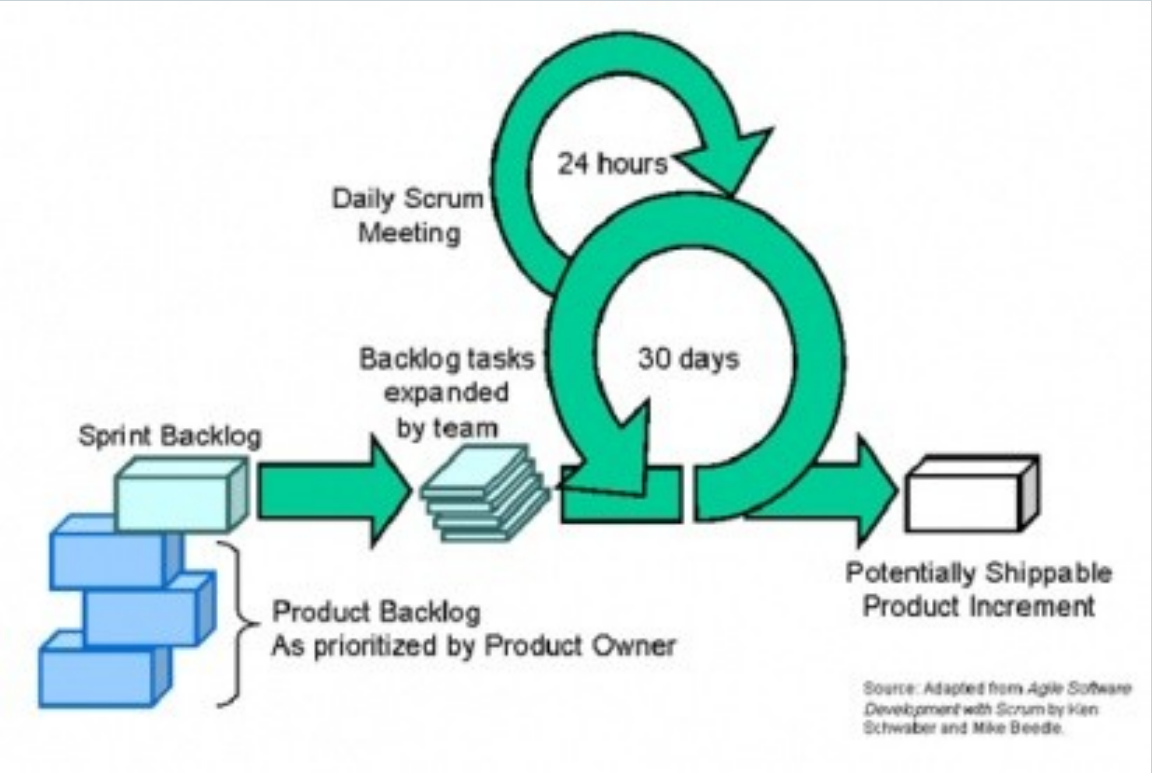
C'est à partir de ces réalités pratiques, et non pas sur la base d'une théorie globale (ou structurante), que l'Agilité progresse à travers toutes les couches de l'entreprise.

Le cercle vertueux…

- 1. Planifier
- 2. Produire
- 3. Vérifier
- 4 . Adapter – Retour au 1

Des méthodes agiles reconnues

- Scrum



Quelques personnalités clés

Martin Fowler

Born	1963 Walsall, England
Home town	Melrose, Massachusetts

Website

martinfowler.com

Kent Beck

Born	1961
Citizenship	United States of America
Fields	Software engineering
Alma mater	University of Oregon
Known for	Extreme Programming, Software design patterns, JUnit

Les étapes d'un processus agile (Scrum)

- Création des « user story »
- Création du « Product backlog » (détermination des fonctionnalités)
- Sprint backlog (définition d'objectif(s) fonctionnel(s) aboutissant à un prototype livrable)
- Burndown chart (tableau des réalisations)
- Obstacles rencontrés
- Validation ou non des fonctionnalités produites

Les activités d'un processus agile (Scrum)

Les activités : sprint, réunions de planning, revues et rétrospectives

http://www.scrumprimer.org/primers/fr_scrumprimer20.pdf

- Sprint : période d'un mois **au maximum**, au bout de laquelle l'équipe délivre un incrément du produit, potentiellement livrable.

Les activités d'un processus agile (Scrum)

Réunion de planification de l'itération

Réunion de préparation d’un Sprint, typiquement divisée en deux parties (la première partie c’est le « quoi » et la seconde partie c’est le « comment »).

Décide des éléments du carnet du produit qu'elle traitera dans le cadre de la prochaine itération, et comment elle s'organisera pour y parvenir

Les activités d'un processus agile (Scrum)

Daily Scrum

(Mêlée quotidienne)

À tour de rôle, chaque membre aborde 3 sujets :

- ce qu'il a réalisé la veille,
- ce qu'il compte réaliser aujourd'hui pour atteindre l'objet du sprint,
- les obstacles qui empêchent l'équipe d'atteindre le but du sprint.

Les activités d'un processus agile (Scrum)

Revue de sprint

En fin de Sprint. Inspection et adaptation de l’incrément : discussion libre autour de l'itération fonctionnelle du produit. Durée : 1h par semaine de sprint.

Il s’agit d’une inspection pratique du logiciel en fonctionnement.

Les activités d'un processus agile (Scrum)

La Rétrospective de Sprint

Inspection et adaptation du processus et de l’environnement.

Durée : 45mn par semaine de Sprint

Suite à la revue de Sprint, c’est l’opportunité pour l’Equipe d’échanger sur ce qui fonctionne bien et sur ce qui ne fonctionne pas, et de s’accorder sur des changements à expérimenter.

Les « stories » et use case

Un récit utilisateur ou "user story", est une phrase simple dans le langage de tous les jours permettant de décrire avec suffisamment de précision le contenu d'une fonctionnalité à développer.

On parle aussi de cas d'utilisation (*use case*), qui définit une manière d'utiliser le système et permet d'en décrire les exigences fonctionnelles. (Ivar Jacobson)

Le concept de Use Case est plus formel, intégré à UML (*Unified Method Language*).

User Story et Use Case décrivent une plus-value apportée par le système en interaction avec l'utilisateur.

les « stories » définition

- Qu’est ce qu’une « user story » ?

C’est une courte description d’une fonctionnalité du« produit » d’un point de vue de l’utilisateur.

Elle suit le modèle suivant :

"En tant que <rôle>, je veux <goal/désir> pour <bénéfice>"

Les « stories »

Elles peuvent être sous forme de récit (assez large)

« En tant que collectionneur, je voudrais pouvoir vendre et acheter des timbres sur internet pour pouvoir compléter ma collection »

On les appelle alors « EPIC »

Ou des Use Case de niveau stratégique

Les « stories »

Elles peuvent être précise

« En tant que collectionneur utilisateur enregistré sur le site, je voudrais pouvoir retrouver mon mot de passe oublié pour pouvoir toujours accéder à mon compte »

Priorité : 2

Les « stories »

- Elles peuvent être regroupées par thème

« Enchère On line »

L’utilisateur peut vendre des cartes

L’utilisateur peut émettre une enchère sur une ou plusieurs cartes

Le vendeur peut mettre un montant minimum au montant de son enchère

Les « stories »

Dans un projet traditionnel :

1.1 Le programme doit permettre de…

1.2 Le programme doit permettre de…

… **orienté « programme », «logiciel »**

Les « stories »

Dans un projet Agile

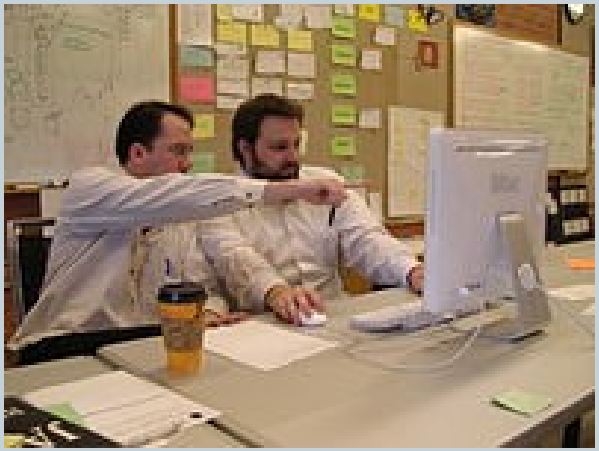
En tant que collectionneur je veux pouvoir mettre en ligne sur le site mes timbres… (pour…)

En tant que collectionneur je veux pouvoir contacter d’autres collectionneurs… (pour …)

En tant qu’administrateur du système (du site) je voudrais pouvoir obtenir des rapports d’activités pour pouvoir contrôler les enchères

Orienté <u>utilisateur</u> du « programme »…

Pair Programming (une pratique XP)



On apercoit en arrière plan ce qui pourrait être un tableau des réalisations (colonnes à faire, en cours, réalisé, ...)

La programmation en binôme (ou *pair programming*) est une méthode de travail dans laquelle deux développeurs travaillent ensemble sur la même partie de code, en binôme sur un même poste de travail.

La programmation se fait en binôme :

le premier, appelé pilote (driver), a le clavier. C'est lui qui va travailler sur la portion de code à écrire.

le second, appelé copilote (partner), est là pour l'aider, en suggérant de nouvelles possibilités ou en décelant d'éventuels problèmes.

Les rôles s'échangent régulièrement pendant la séance de programmation.

Définir le périmètre fonctionnel

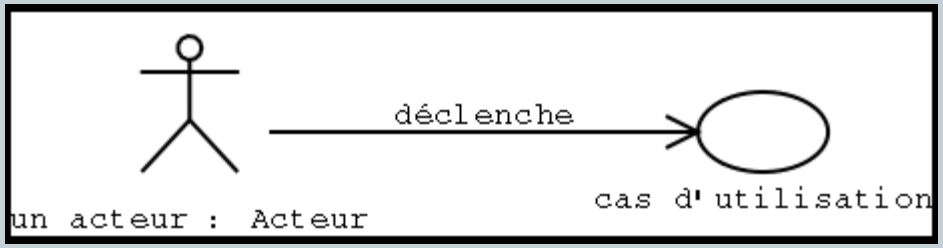
Permet d'avoir une vision d'ensemble des objectifs à atteindre (fonctionnalités à implémenter)

Leur associer une valeur d'importance (pour une ordonnancement des taches)

… et de repérer ainsi ce qui pourrait être en dehors du périmètre.

Représentation UML : diagramme des cas d’utilisation

Représentation des Uses Cases (UML)



Un cas d'utilisation spécifie une séquence d'actions avec un acteur, génératrice d'une plus value.

Vue synthétique :

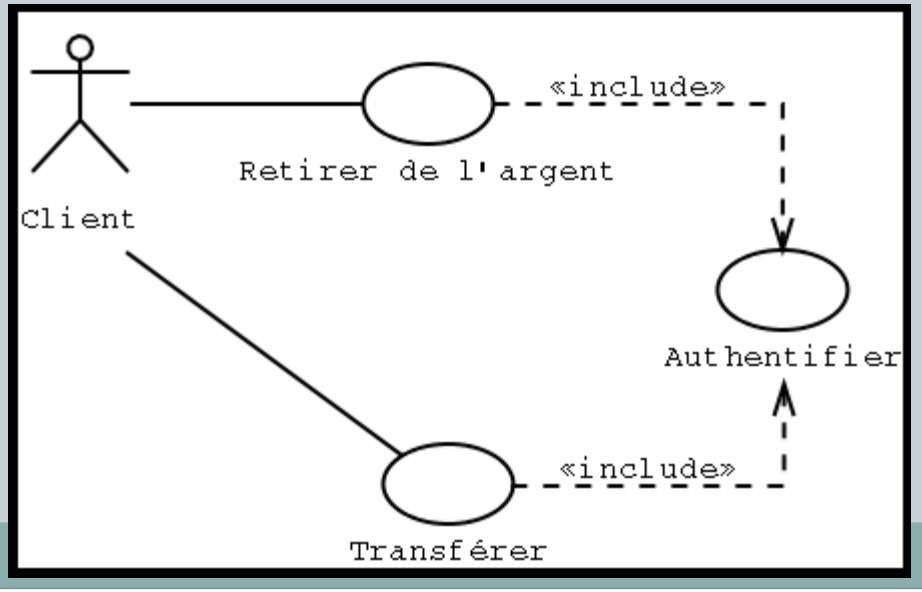
Un acteur est une entité externe au système, en interaction avec ce dernier. Ce peut être un rôle joué par un utilisateur, par exemple un comptable, ou par un autre système, un capteur par exemple. Il est d'usage, mais absent de la norme UML, de distinguer les acteurs **principaux** des acteurs **secondaires**. Les fonctionnalités principales du système ont été définies pour les acteurs principaux.

Relations entre cas d'utilisation

UML propose trois types de relations standard entre cas d'utilisation, <<**include**>> , <<**extend**>> et **généralisation** . Les deux premières sont représentées par un stéréotype de dépendance, l'autre étant la relation de généralisation représentée en UML par une flèche creuse à pointe fermée.

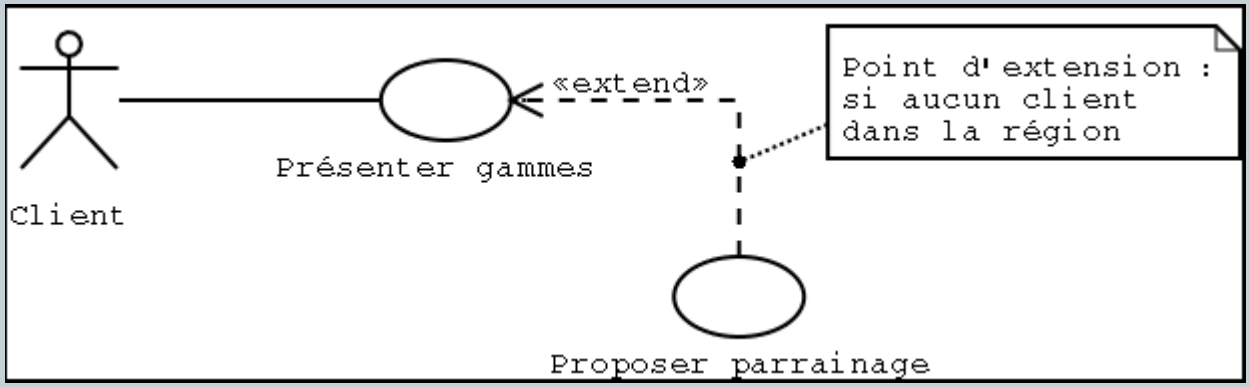
<<include>> : Stéréotype représentant le fait qu'un cas d'utilisation inclut un autre cas d'utilisation. On utilise ce stéréotype lorsque que l'on souhaite factoriser un cas d'utilisation partagé par plusieurs autres cas d'utilisation. Par exemple, une opération de retrait et une opération de transfert nécessitent toutes deux une opération de vérification de l'identité du client.

Un diagramme de cas d'utilisation



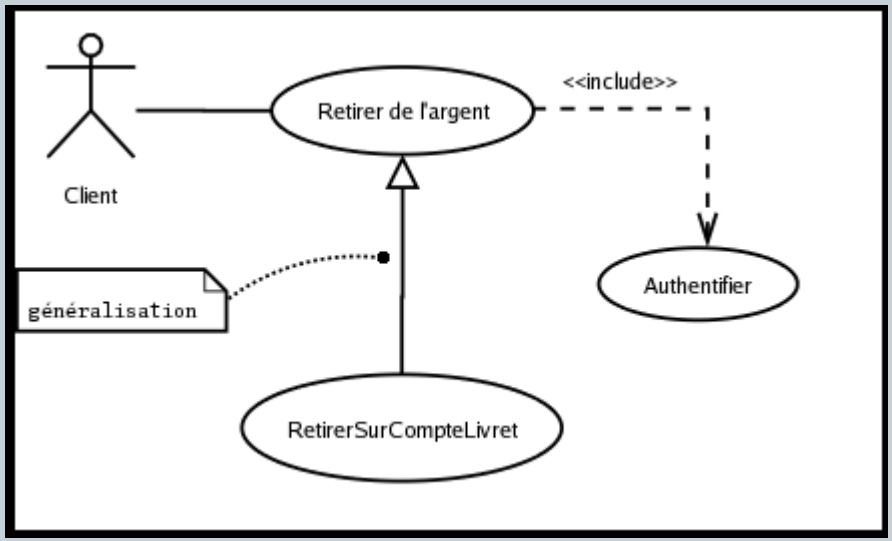
Relations entre cas d'utilisation

<<**extend**>> : Un cas d'utilisation peut déclarer des points d'extension (extension point). Un point d'extension localise un endroit (un point) unique dans le cas d'utilisation. C'est dans les limites de ce point que d'autres cas d'utilisation pourront étendre (extend) le comportement initial du cas d'utilisation. C'est un moyen pratique de mettre en avant une fonctionnalité optionnelle. Par exemple, lors de la conception d'un site marchand pour un fabricant de produit de beauté, on souhaite proposer à certains visiteurs de promouvoir la marque dans leur région.



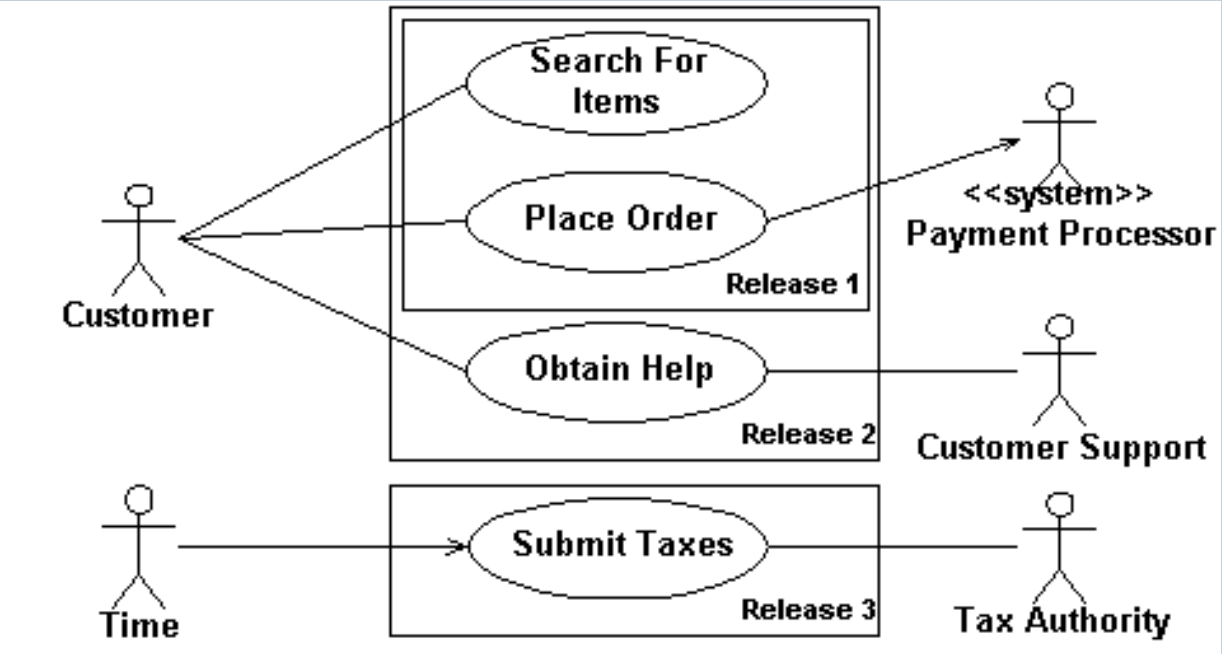
Relations entre cas d'utilisation

Généralisation : Une relation de généralisation d'un cas d'utilisation B vers un cas d'utilisation A signifie que B est une spécialisation de A. Contrairement aux deux autres relations, la relation de généralisation n'est pas un stéréotype. Elle indique qu'un cas d'utilisation est une variation d'un autre. Cette relation se différencie de <<extend>> par le fait que le cas d'utilisation peut varier en tout point de celui hérité. Par exemple dans l'UC "Retirer de l'argent", si il s'agit de retirer de l'argent sur un compte sur livret le comportement de l'UC peut être tout à fait différent.



Un exemple simple

Exercice d'interprétation d'un diagramme de cas d'utilisation
(http://www.agilemodeling.com/artifacts/useCaseDiagram.htm)



Dans la pratique SIO

Il est essentiel que tous les acteurs d'un projet (développeur, UIDesigner, …) partagent la même vision globale du projet. Pour cela vous devrez, par équipe :

1/ Identifier les acteurs du projet en cours
2/ Leur attentes (au moins un user-story par acteur)
3/ Modéliser le tout par un diagramme simple des cas d'utilisation puis un *product backlog*
4/ Définir des objectifs de *releases* (maquette, prototypes) : *sprint backlog*
5/ Vous répartir le travail, de préférence réaliser les tâches en binôme