heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Ecole d'Ingénieurs de l'état de Vaud
Eric Lefrançois
Mars 2018

Génie Logiciel- Le cycle de vie

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

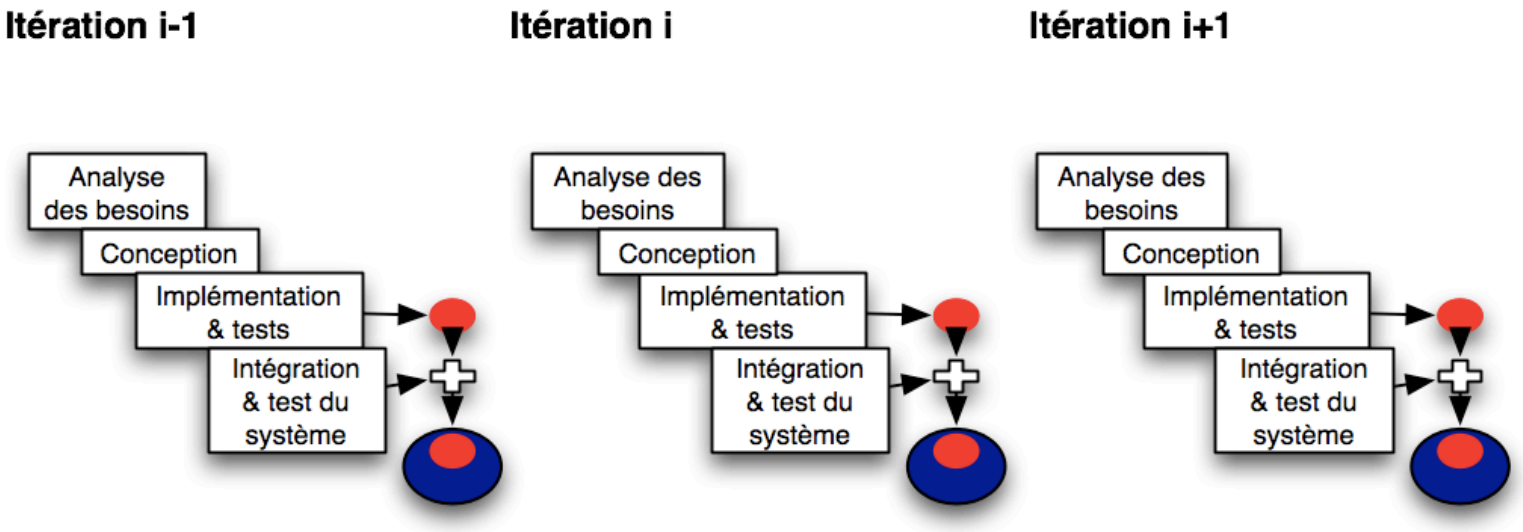
Sommaire

Cycles de vie et méthode UP

Le développement itératif est à la base de UP (*U*nified *P*rocess)

Définition: **Itération**

- Aboutit à un système exécutable et testé
- Un **mini-projet** en soi

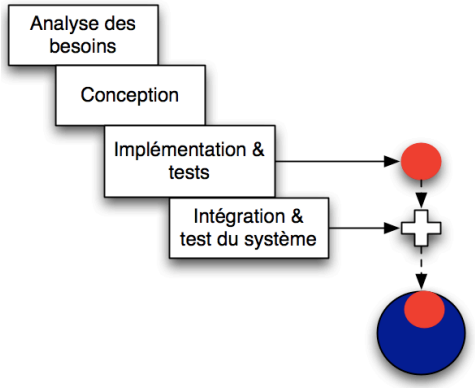


EXEMPLE

Itération de deux semaines à mi-chemin du développement d'un projet..

- **Lundi**
  - clarification et distribution des tâches
  - en parallèle, une personne effectue le reverse-engineering de l'itération précédente (⇨ UML)
- **Mardi**
  - Conception de diagrammes UML grossiers
    - conçus et dessinés au tableau en «binôme»
    - enregistrés avec une caméra numérique
  - Ecriture de fragments de pseudocode, etc.
- **Les 8 derniers jours..**
  - Implémentation et test d'unités
  - Intégration dans le système

  ✚ *démonstrations aux différents partenaires, planification des prochaines itérations*



LES DIFFÉRENTES PHASES DU CYCLE DE VIE: ANALYSE ET CONCEPTION

Définition ⇨ **Cycle de vie**

Modélise l'enchaînement des différentes activités du processus technique de développement du logiciel

Selon le modèle, 3 grandes activités vont interagir différemment :

- **L'analyse** ⇨ comprendre le problème, les besoins
- **La conception** ⇨ trouver une architecture
- **La réalisation** ⇨ mettre en œuvre, fabriquer

IL N'Y A PAS QU'EN INFORMATIQUE ..

Par exemple, pour fabriquer un modèle mécanique du système solaire,

il faut:

1. **Analyser** le problème en observant que les planètes suivent des orbites

2. **Concevoir** en inventant un mécanisme qui déplace des sphères sur de telles orbites

3. puis **Réaliser** l'assemblage des sphères, ressorts et engrenages

NOTION DE CYCLE DE VIE – SUITE..

Toute **méthode de développement**

- ⇨ Propose une démarche (le **cycle de vie**)
- ⇨ Des notations pour aborder ces problèmes

La **démarche** (le cycle de vie) décrit :

- Quelles étapes élémentaires doivent être suivies
- Quelles questions doivent être soulevées et à quel moment
- Quels sont les «objets» qui doivent être mis en évidence
- etc.

La **notation**

⇨ Permet de présenter et formaliser des «objets» solution aux informaticiens et aux clients

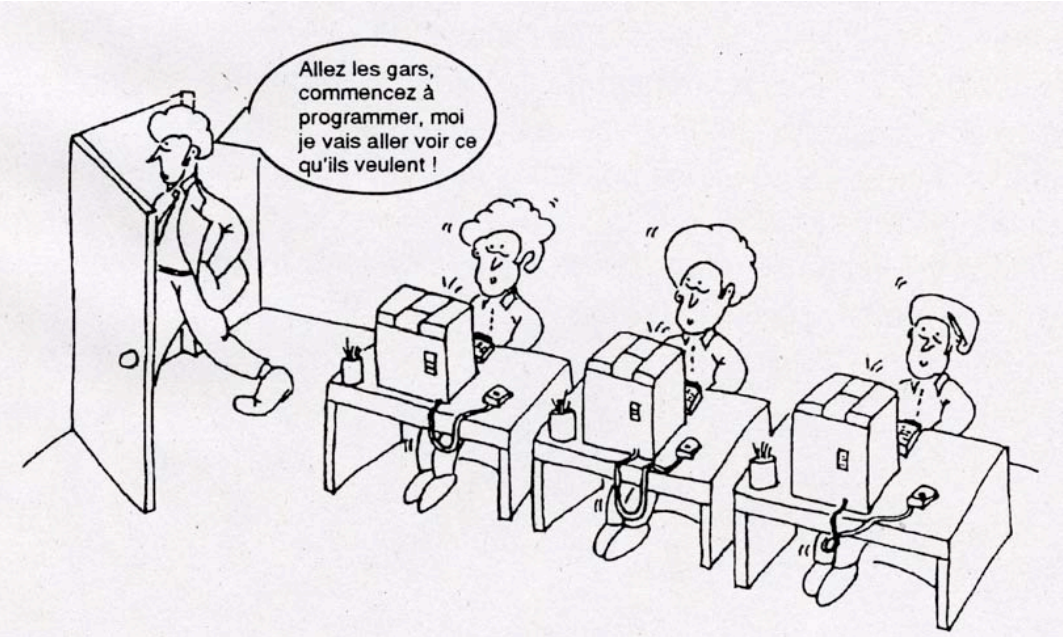
⇨ Doit être compréhensible par des non-spécialistes.

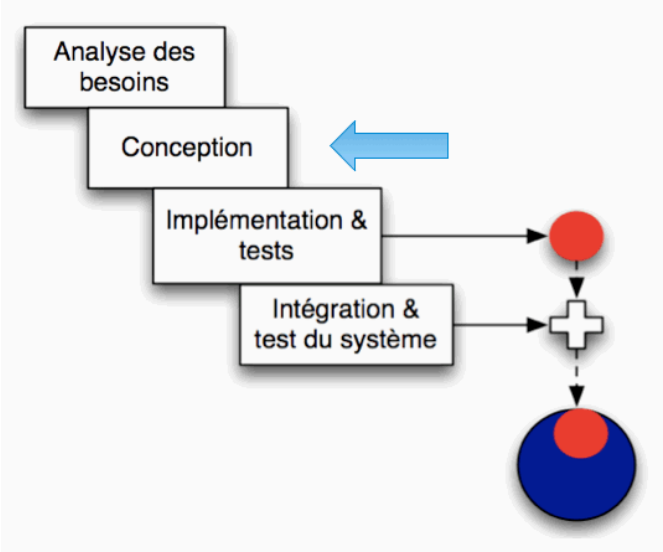
ANALYSE ET CONCEPTION

Définition ⇨ **Analyse**: le «**QUOI**»

- Activité qui s'attache à mener une **investigation sur le problème et les besoins**.
- Ne recherche pas une solution !

La question ⇨ «Qu'est-ce qu'on désire et **comment ce sera utilisé**?»



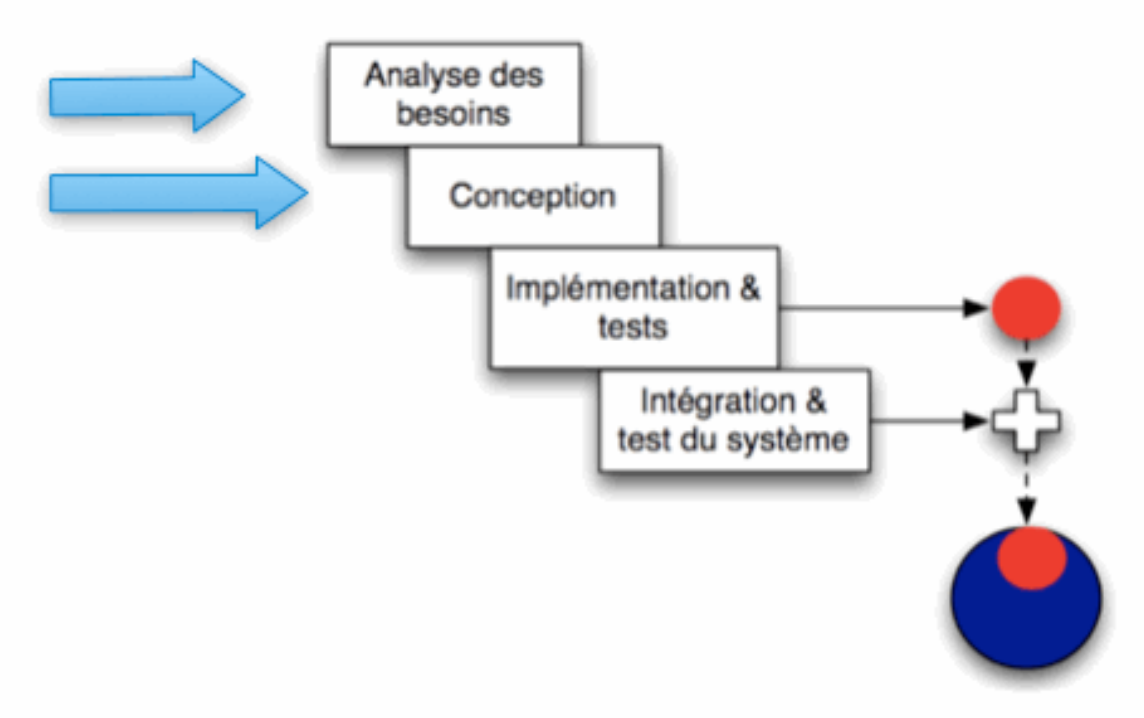


CONCEPTION: LE «COMMENT»

Définir une **solution conceptuelle** susceptible de répondre aux besoins de l'analyse

- Schéma conceptuel de la base de données
- Architecture des classes (au moyen d'un diagramme UML).
- Etc.

Ne s'attache pas à définir une implémentation du problème !

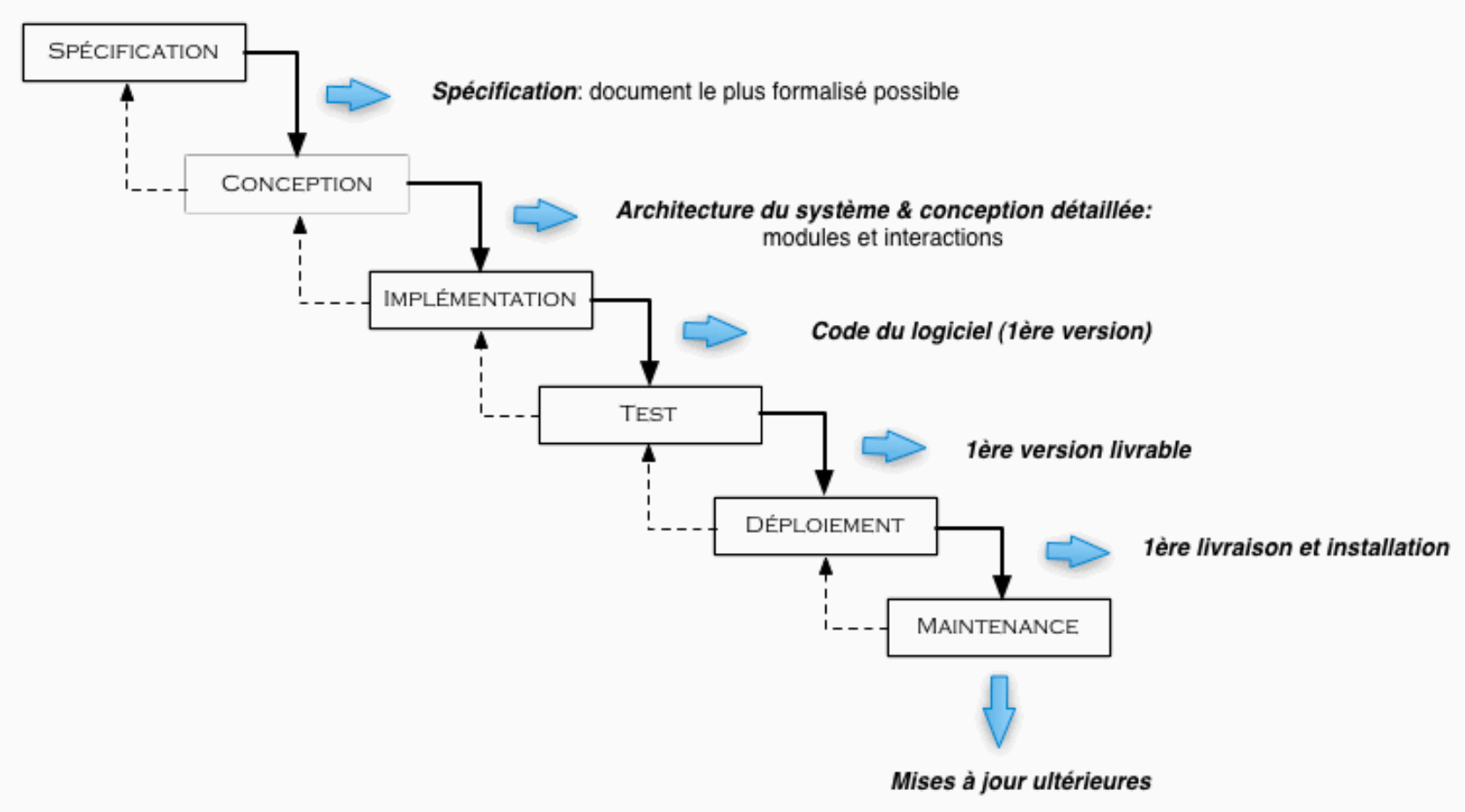


EN RÉSUMÉ...

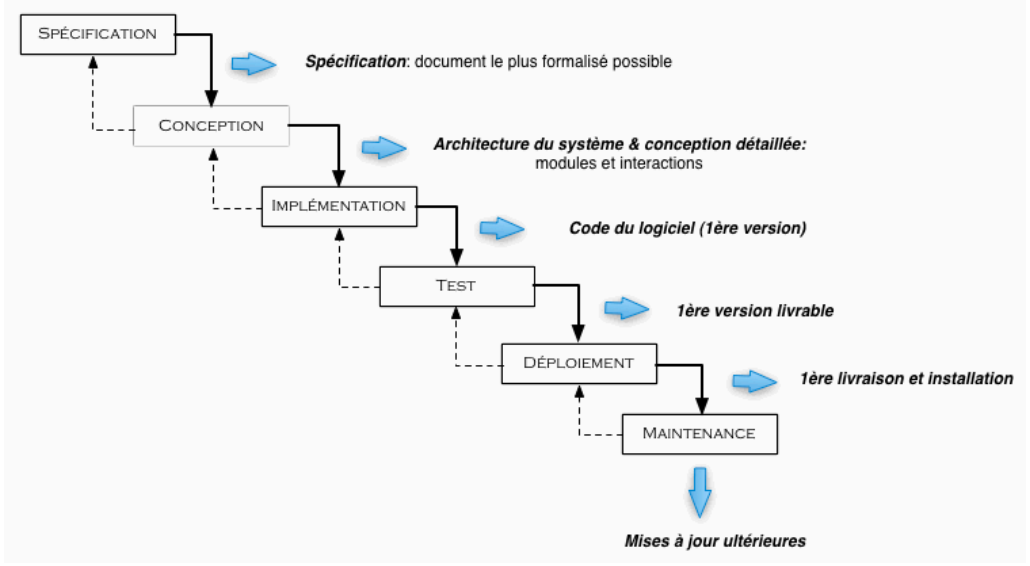
Do the *right thing* (analysis), and do the *thing right* (conception)

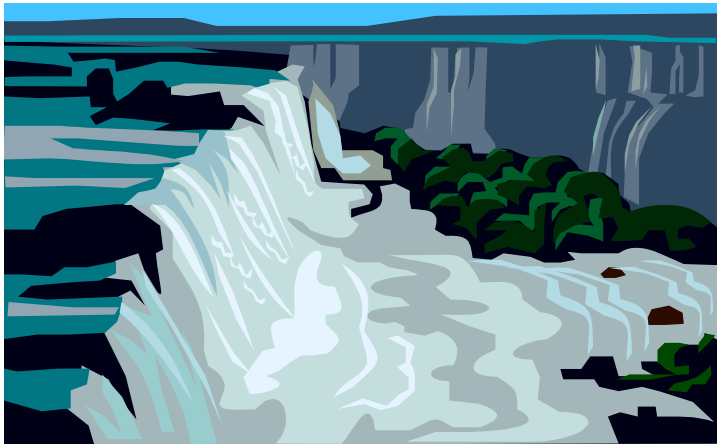
LE CYCLE DE VIE EN CASCADE (WATERFALL)

Les balbutiements du Génie Logiciel.. dû à Royce (1970)



Modèle très simple ! Inspiré des autres domaines de l'ingénierie

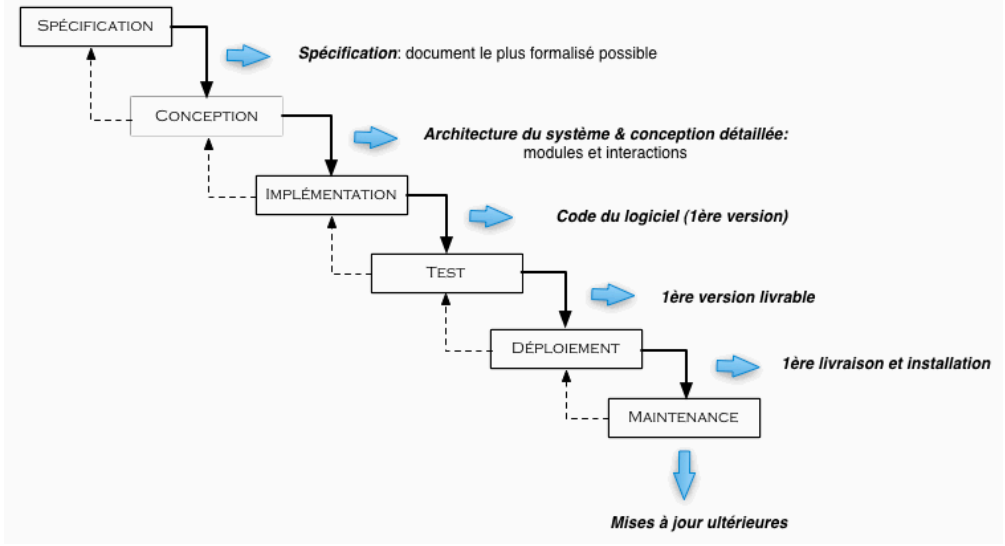


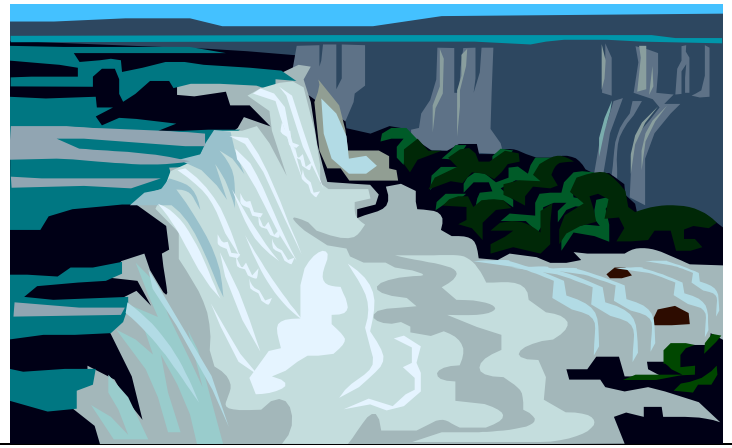


LES AVANTAGES

☺ Développement très linéaire

☺ Il permet de planifier aisément le développement



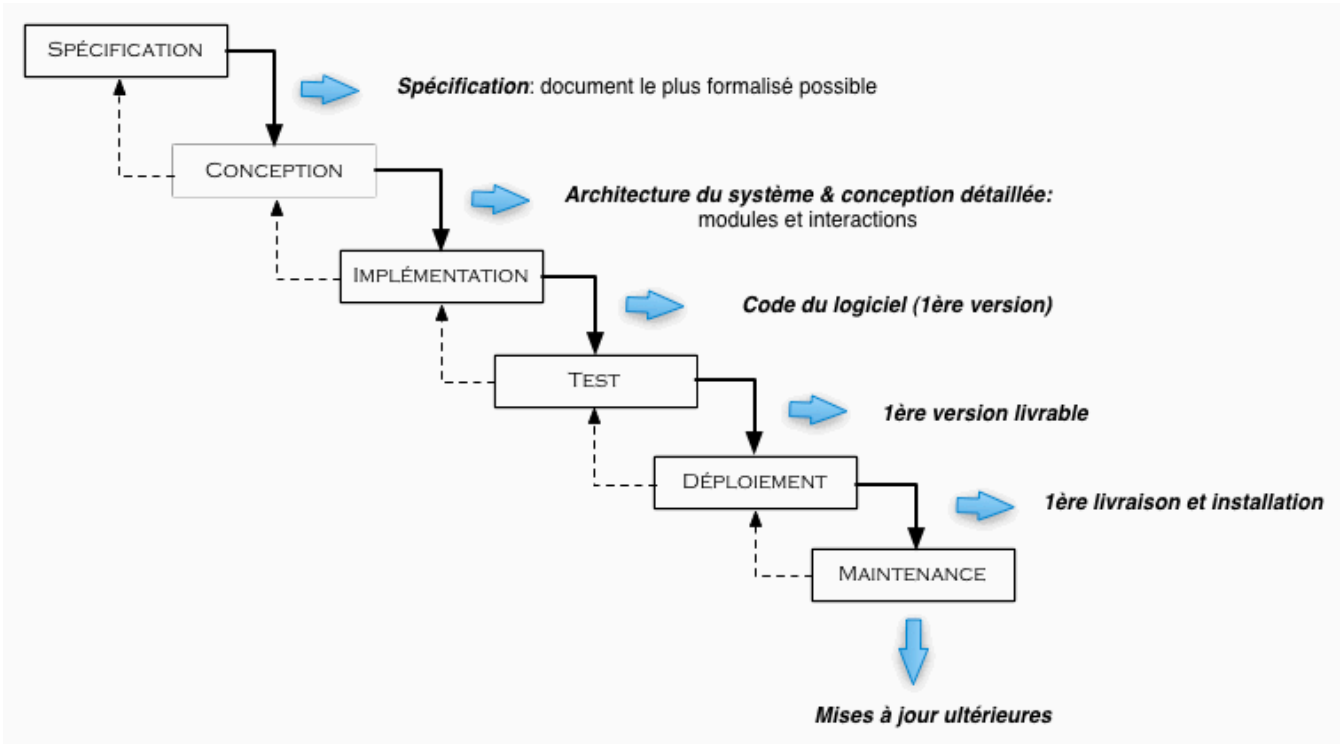


PAR CONTRE

☹ Permet difficilement de gérer les changements en cours de projets

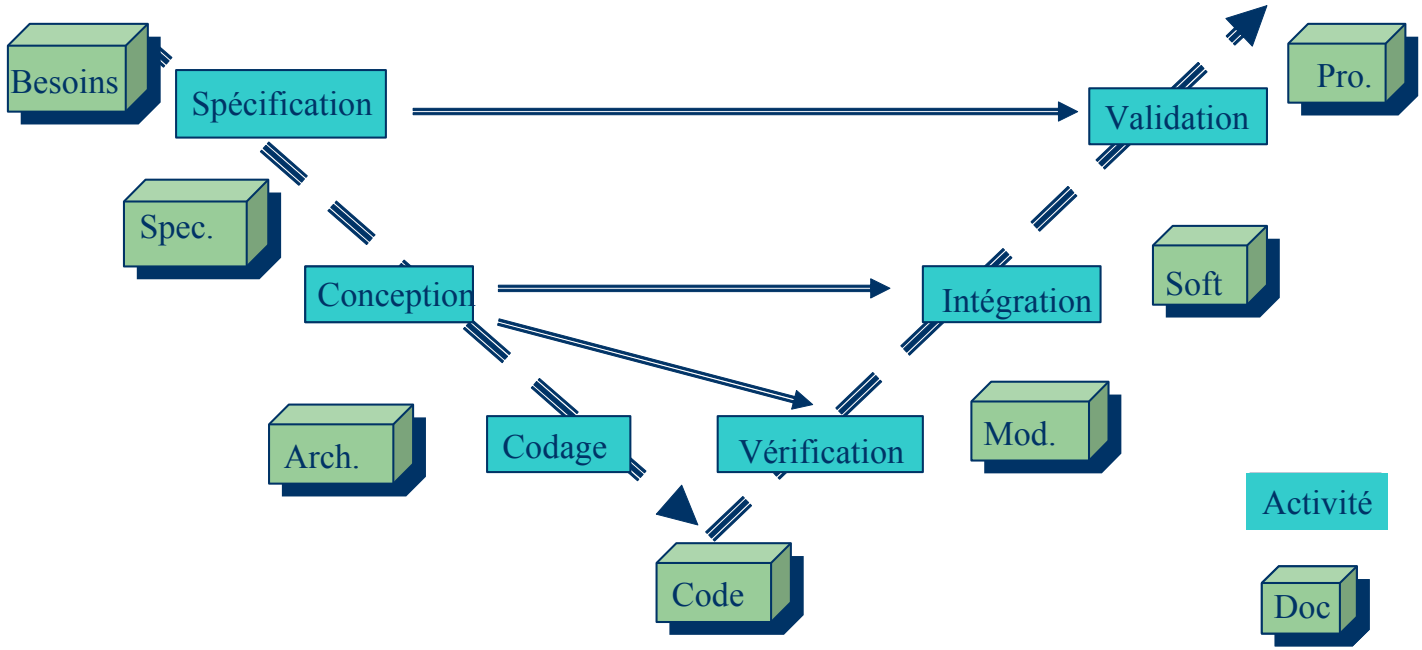
MAIS SURTOUT

Le client ne voit le système réalisé qu'à la fin..



LE CYCLE EN V

Une amélioration du cycle en cascade..



Nouvelles notions ..

- **Découpage modulaire ⇨ vérification**
- **Intégration ⇨ validation**

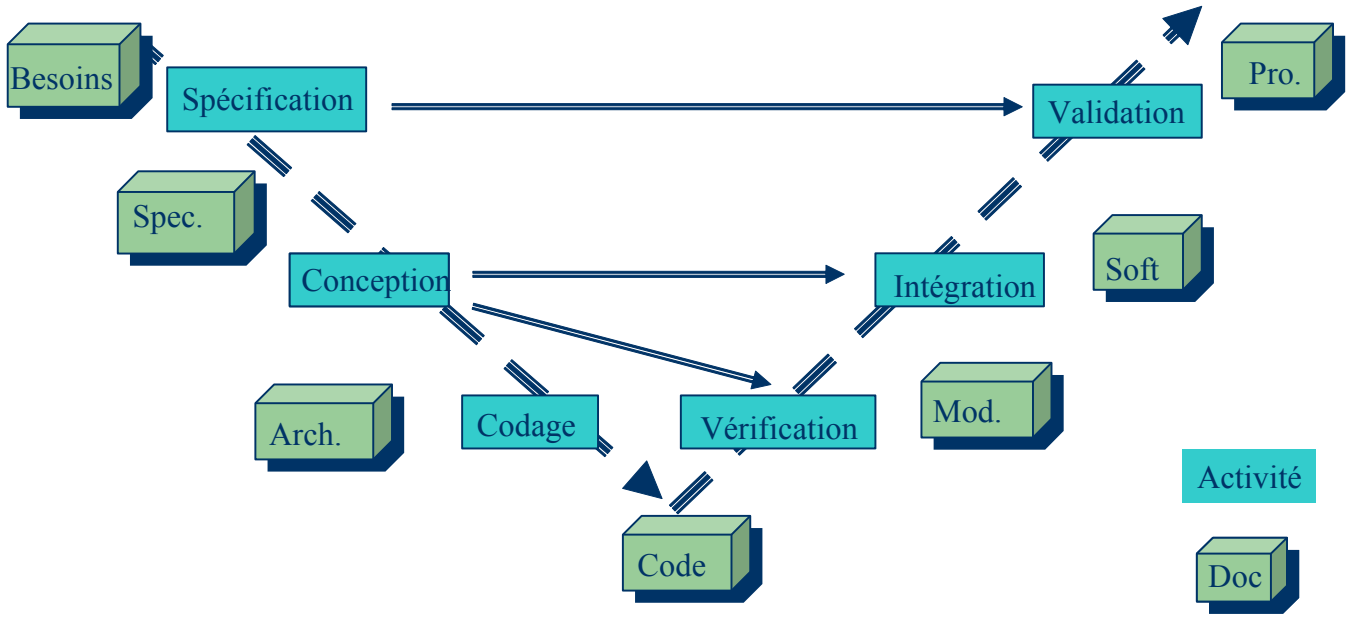
Définition ⇨ Validation et Vérification (Boehm 76)

Validation: Am I building the right system?

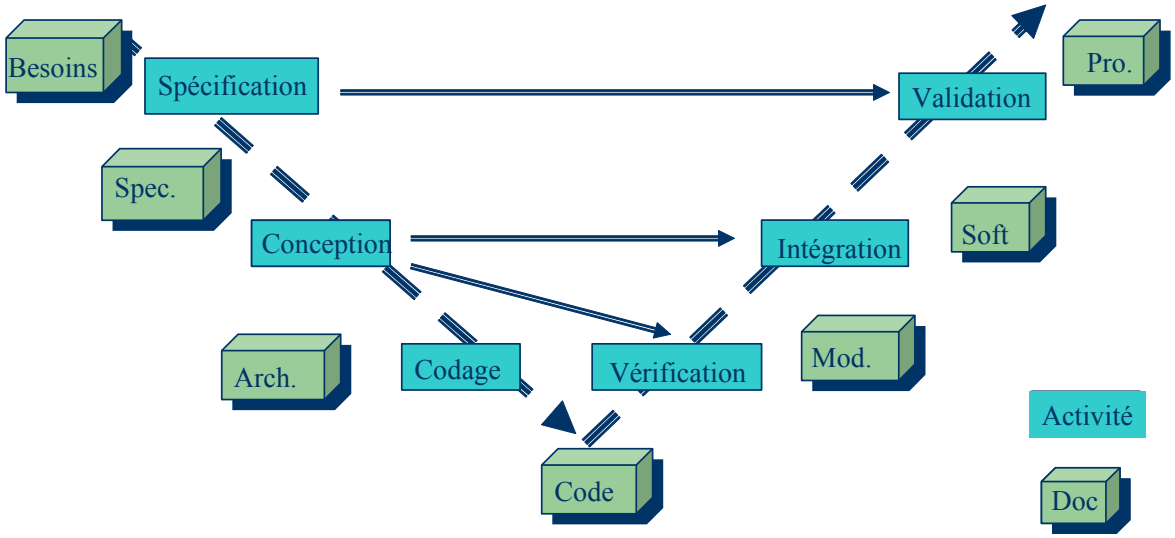
⇨ *Le système correspond-il aux besoins du client ?*

Vérification: Am I building the system right?

⇨ *Le système est-il conçu de manière correcte ? absence de bugs, etc..*



En bref



C’est mieux que le cycle en cascade..

☹ **Mais ..**

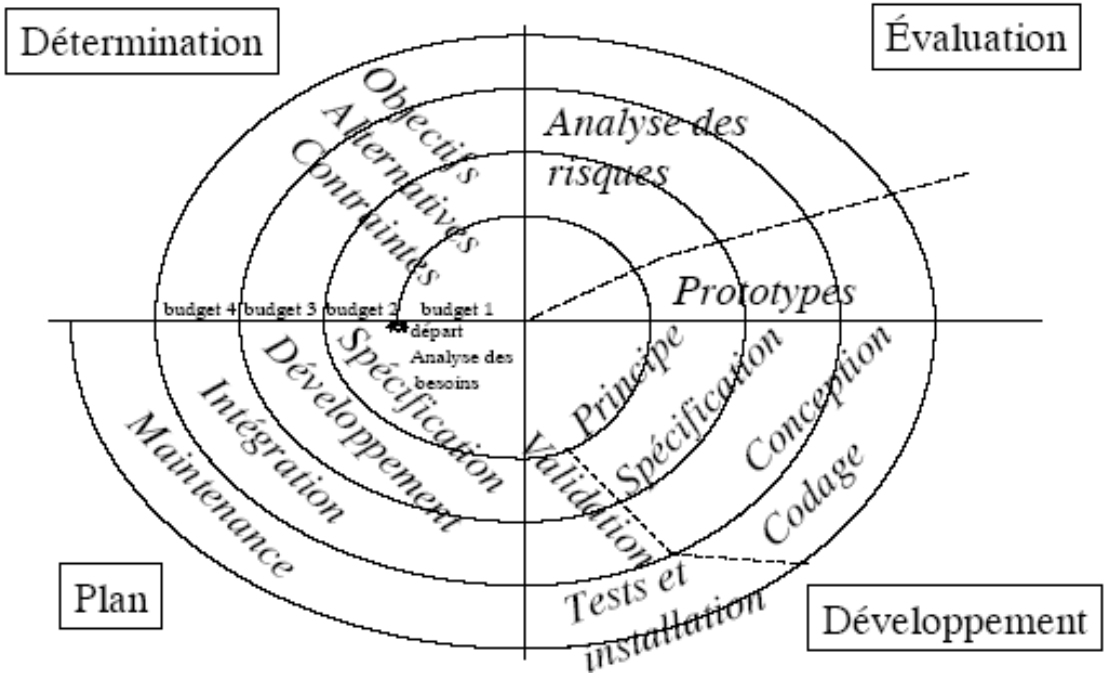
Le client ne voit le système réalisé qu’à la fin..

CYCLE EN SPIRALE

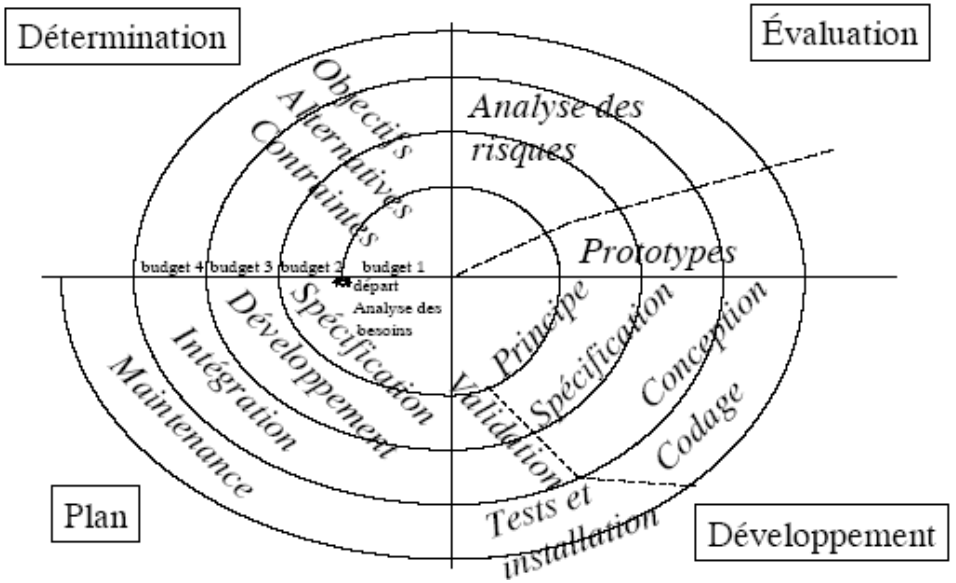
Amélioration du cycle en V : 1988, K. Boehm

- Introduction de la notion de **gestion des risques**
- Révision continuelle de la planification avec implication du client

☺ Ce modèle implique fréquemment les futurs utilisateurs



PRINCIPE



- Succession de cycles en cascade
- A chaque cycle, le système est :
  - analysé,
  - conçu,
  - réalisé
  - et testé

Un peu plus qu'à l'itération précédente

LA GESTION DE RISQUES

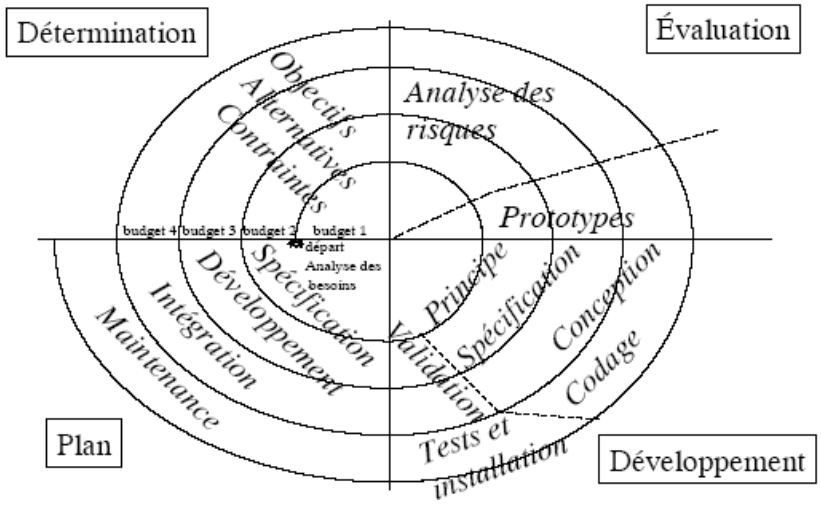
Définition ⇨ *Gestion des risques*

Placer la **priorité** sur l'analyse et le développement des éléments dont le développement présente des **risques** importants

Comme par exemple, les points:

- où le client n'est pas très au clair avec sa demande
- où l'équipe n'a pas d'expérience sur ce type de développement
- où la technologie à utiliser est inconnue et peut réserver de mauvaises surprises,
- *etc.*

POUR CHAQUE CYCLE



1. **Détermination** des objectifs du cycle
2. **Evaluation** : Analyse des risques et fabrication éventuelle de prototypes
3. **Développement** et tests
4. **Plan** du prochain cycle, puis.. *Prochain cycle..*
    i. Détermination des objectifs du cycle
    ii. Evaluation : Analyse des risques et fabrication éventuelle de prototypes
    iii. Développement et tests
    iv. Plan du prochain cycle
        1. Détermination des objectifs du cycle
        2. …

COMPARAISON ENTRE WATERFALL ET SPIRALE

- Systèmes connus
  ⇨ Le modèle Waterfall (en V) convient très bien

- Pour les systèmes «à risques»,
  pour les systèmes dont la spécification est incomplète
  ⇨ Modèle en spirale

☺ *Rien n'empêche d'utiliser des modèles hybrides pour différentes parties du projet !*

COMPARAISON ENTRE WATERFALL ET SPIRALE

- Systèmes connus ⇨ Modèle Waterfall (en V)
- Systèmes «à risques» ⇨ Modèle en spirale

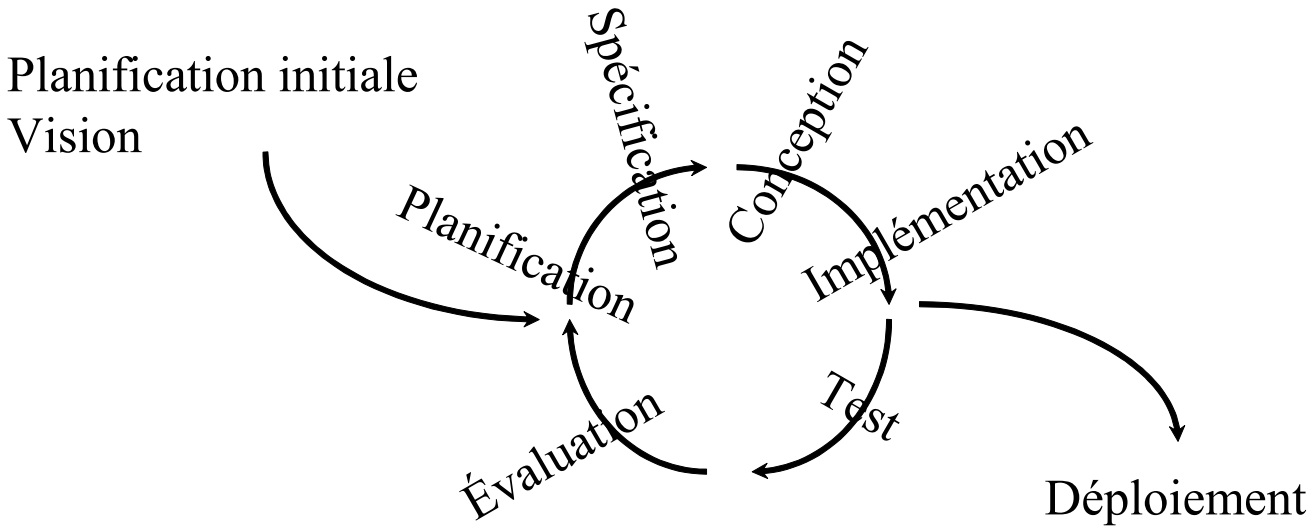
Mais finalement, que le système soit en cascade, en V ou encore en spirale...

⇨ Le client ne voit le système réalisé qu'à la fin!

⇨ Bref...

LE MODÈLE INCRÉMENTAL ITÉRATIF

⇨ Aboutissement du modèle en spirale



Aujourd'hui utilisé par les méthodes :

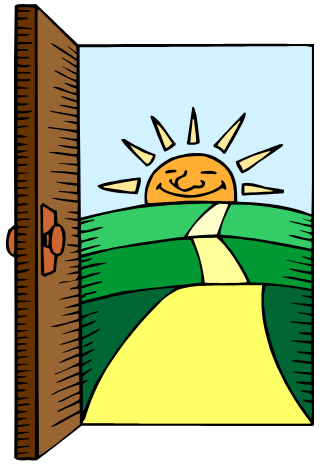
- «**agiles**» ⇨ **XP**
- ou «**semi-agiles**» comme **UP**

A LA DIFFÉRENCE DU MODÈLE EN SPIRALE

- Chaque incrément donne lieu à un produit fini, **exécutable**, testé et intégré.
- Chaque incrément à taille humaine, en limitant le temps de réalisation et en limitant le nombre de fonctionnalités implémentées

LES AVANTAGES PAR RAPPORT AU MODÈLE EN SPIRALE

- Le client peut valider en tout temps
- Peut influencer son développement dans le bon sens



⇨ Le système correspond mieux aux besoins du client

⇨ Les erreurs sont identifiées rapidement ⇨ efficacité

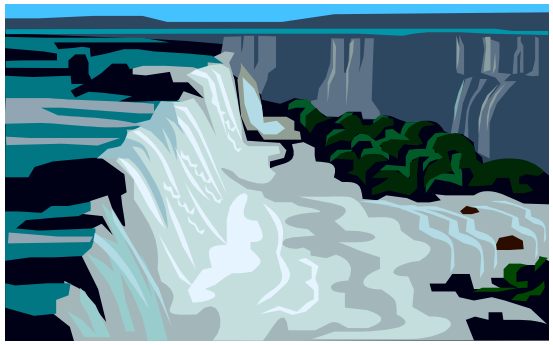
MAIS … EXPÉRIENCE SOUHAITÉE..

Au départ :

- Etablir une vision de haut niveau
- Estimer les coûts et les temps de développement

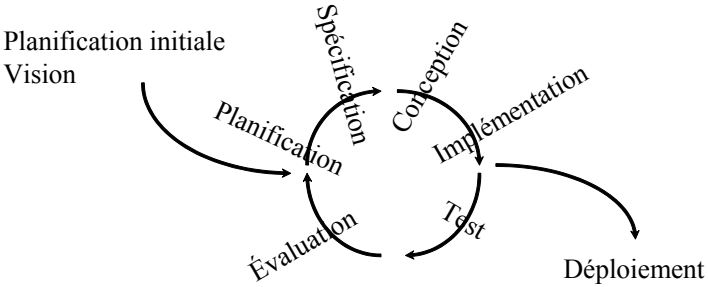
⇨ Demande beaucoup d'expérience

LE DEVELOPPEMENT EN CASCADE EST-IL ENTERRE?



- Si l'équipe de développement bénéficie d'une grande expérience dans le domaine
- Si le travail ne présente par ailleurs aucun risque

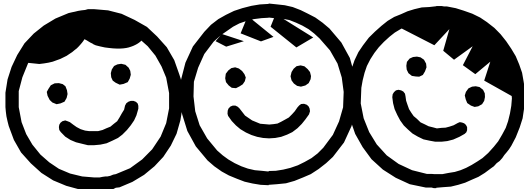
⇨ **Modèle cascade plus efficace !**



Dans les autres cas..

LA MÉTHODE UP (PROCESSUS UNIFIÉ)

Proposée en 2001 par l'équipe UML des 3 amigos..



Booch, Rumbaugh et Jakobson

(Entreprise Rational)

UP s'appuie essentiellement sur deux principes:

- Développement incrémental itératif
- Utilisation de la **technologies objet**, que ce soit pour l'analyse, la conception et la programmation.

CARACTERISTIQUES REMARQUABLES DE UP

Développement itératif...

- Prise en compte quasi immédiate des éléments à risques
- Implication permanente des utilisateurs : évaluation & test
  + Pratique des demandes de changement

Plus spécifiquement ..

- Mise en œuvre d'une architecture centrale dès les premières itérations

- **Approche Objet** et modélisation graphique (Diagrammes **UML**)

  ⇨ Besoin de construire des modèles **plus proches du monde réel**

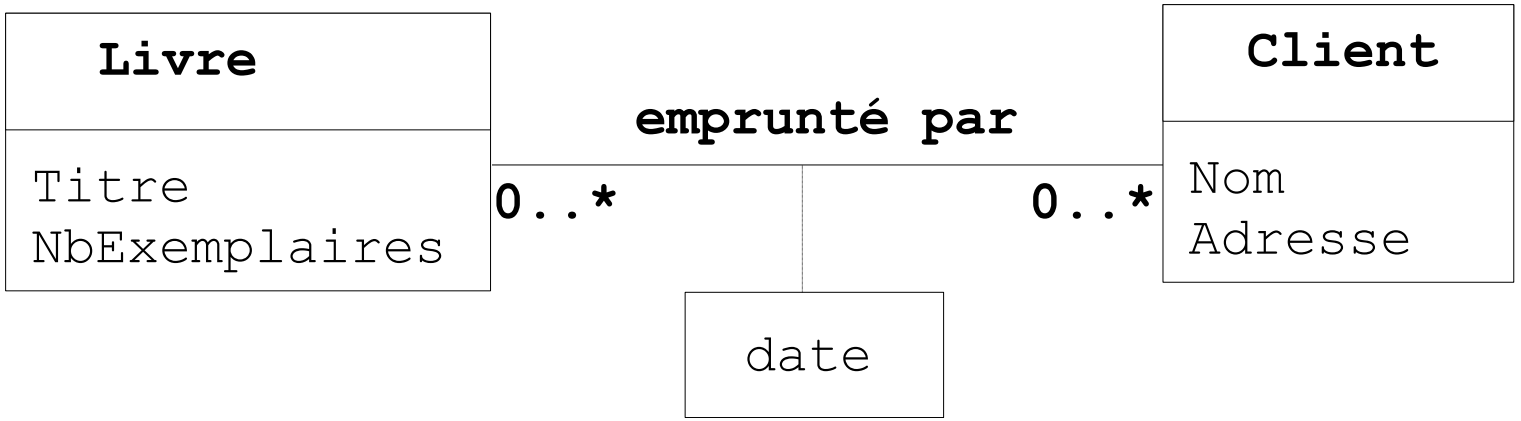
APPROCHE OBJET ⇨ ANALYSE ORIENTÉE OBJET

⇨ Recherche et description des objets ou des concepts qui appartiennent au **domaine du problème**

Exemple

Un système de gestion d'une bibliothèque ⇨ `Livre` et `Client`

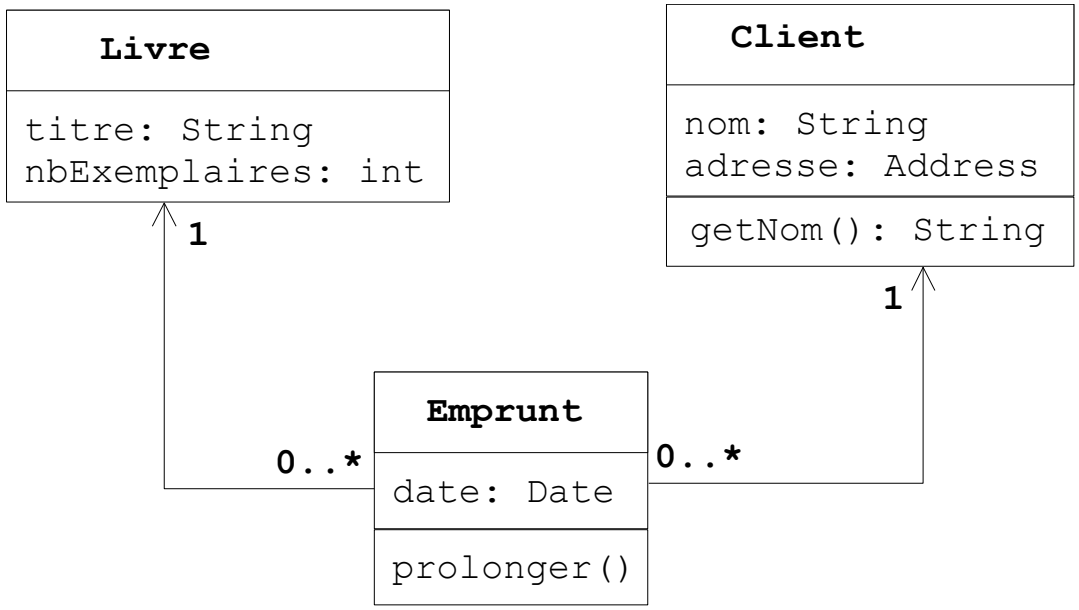
⇨ Un **modèle de domaine** (UML)



APPROCHE OBJET ⇨ CONCEPTION ORIENTÉE OBJET

⇨ Recherche et définition des **objets logiciels** (tels qu'ils seront implémentés)

⇨ Définition de la manière dont ces derniers vont collaborer




```
class Client {
   private String nom;
   private String adresse;

   public String getNom() {return "";}
}
```

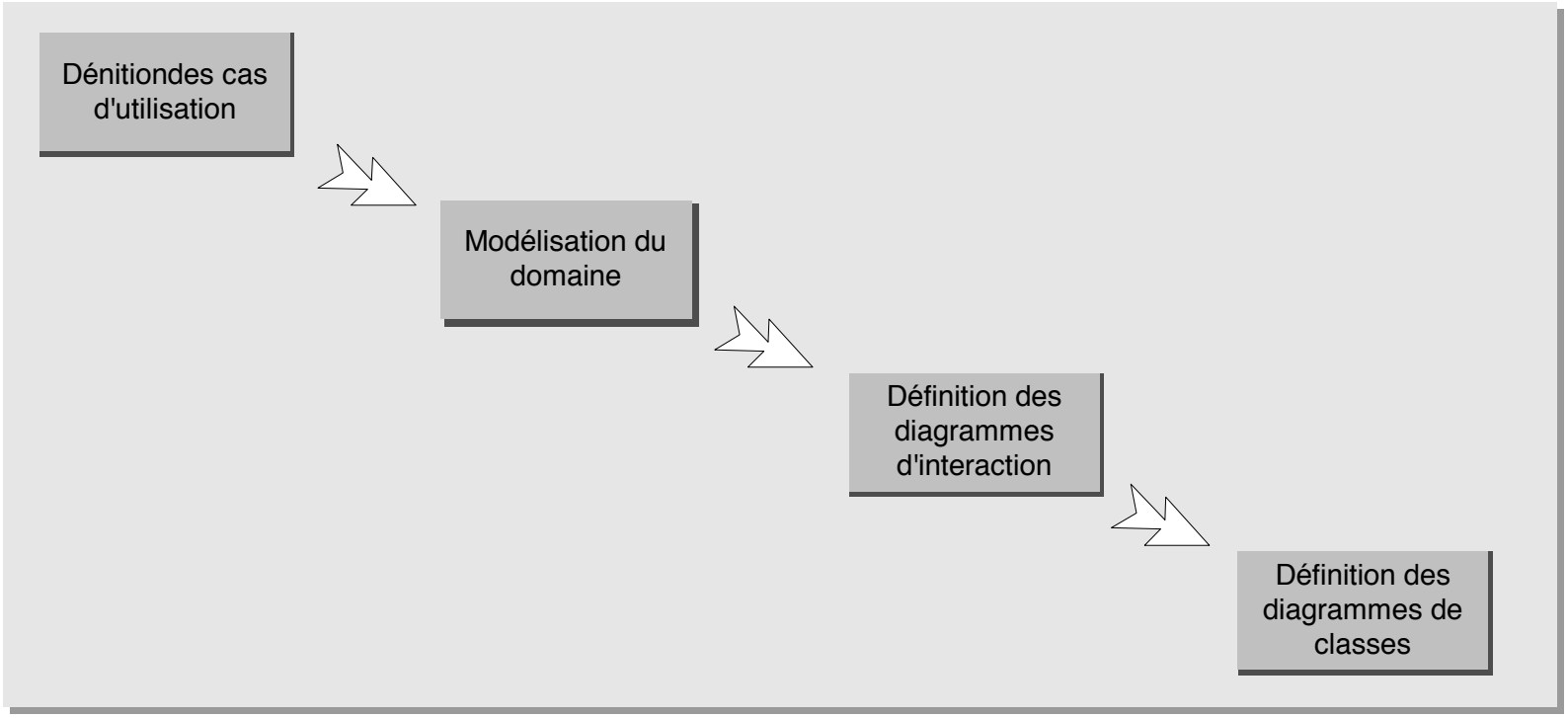
UN SECOND EXEMPLE: LE JEU DE DÉS

Le jeu est le suivant..

Les dés sont roulés..

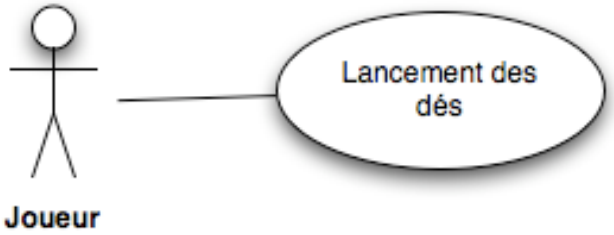
Si le total fait 7 on a gagné, sinon on a perdu !

LES 4 ETAPES DU DEVELOPPEMENT DU JEU DE DES



DÉFINITION DES CAS D'UTILISATION

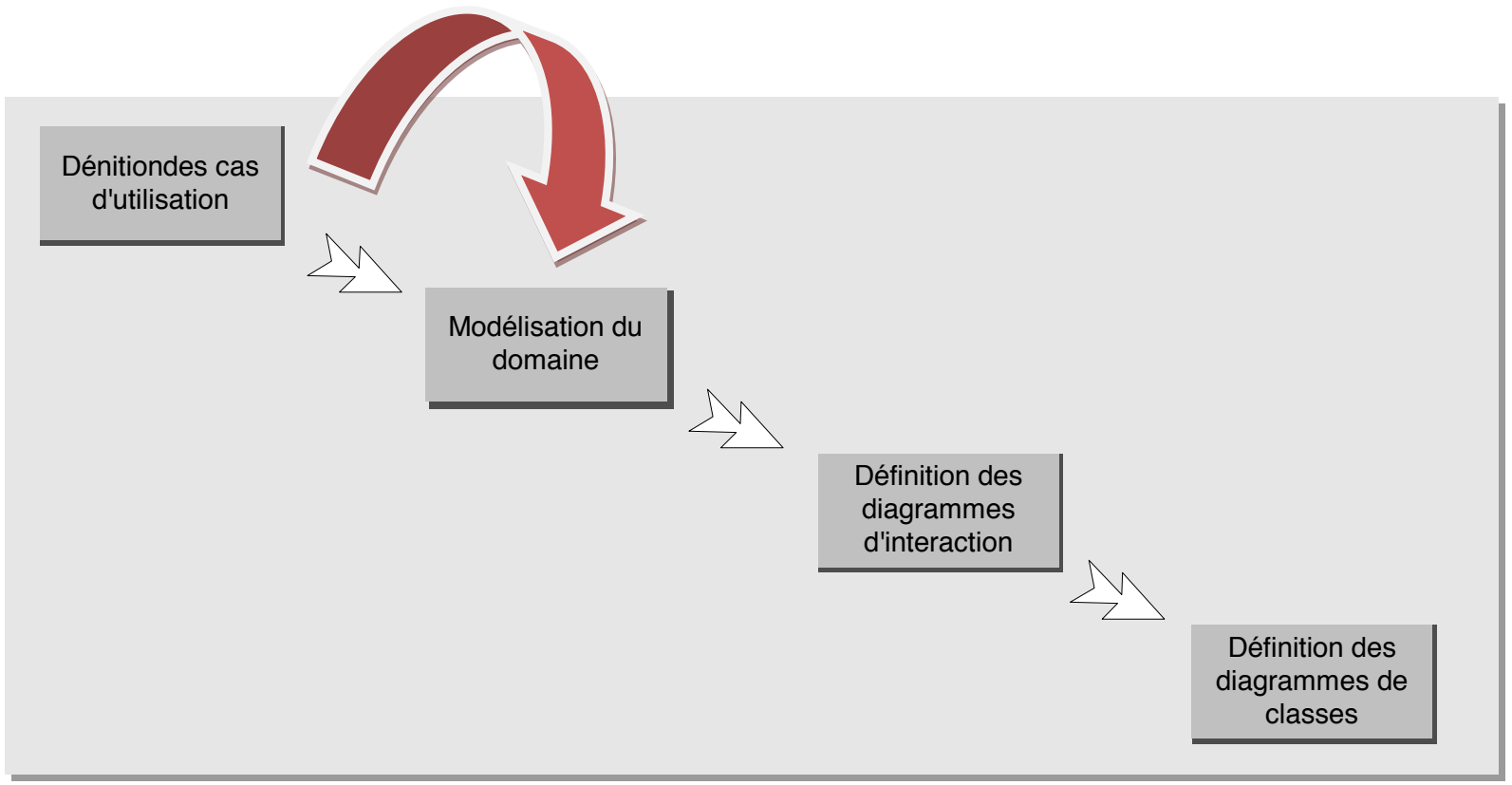
Analyse des besoins ⇨ définition **des cas d'utilisation**



Un **cas d'utilisation** représente **une fonctionnalité**

> **Scénario** : Lancement des dés
>
> 1. Le joueur lance les dés.
> 2. Si le total est égal à sept, il a gagné. Dans les autres cas, il a perdu.

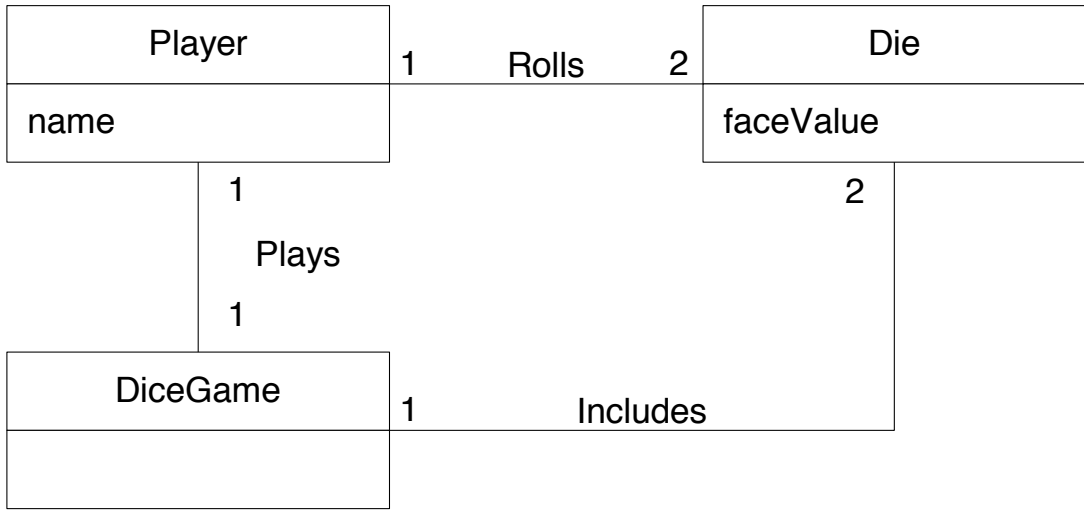
⇨ Le système est décrit en termes de **fonctionnalités !!** ~~(pas d'objets)~~

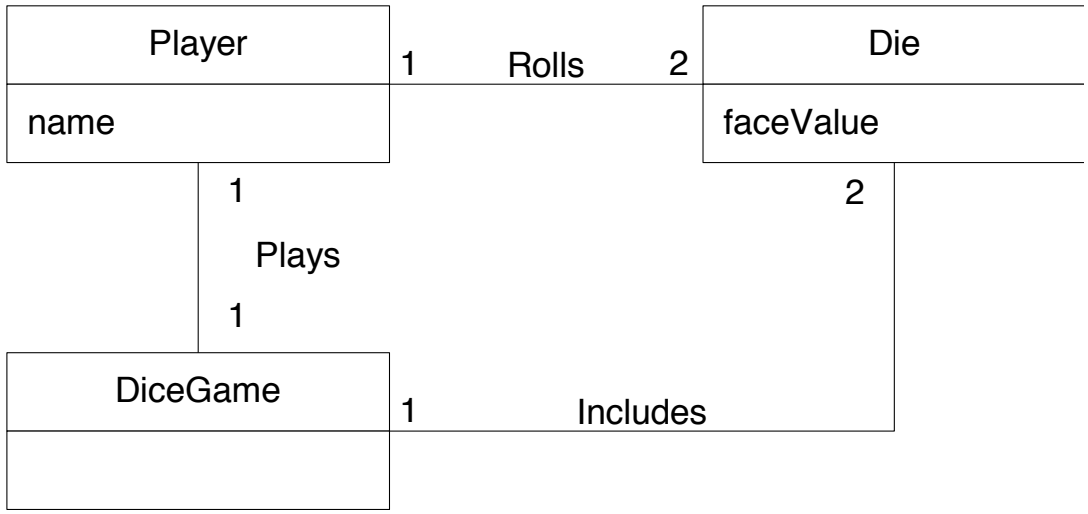

Dénitiondes cas d'utilisation
Modélisation du domaine
Définition des diagrammes d'interaction
Définition des diagrammes de classes

MODÉLISATION DU DOMAINE

✍ Le modèle du domaine n'est pas une description d'objets logiciels, mais une représentation des concepts appartenant au domaine du monde réel

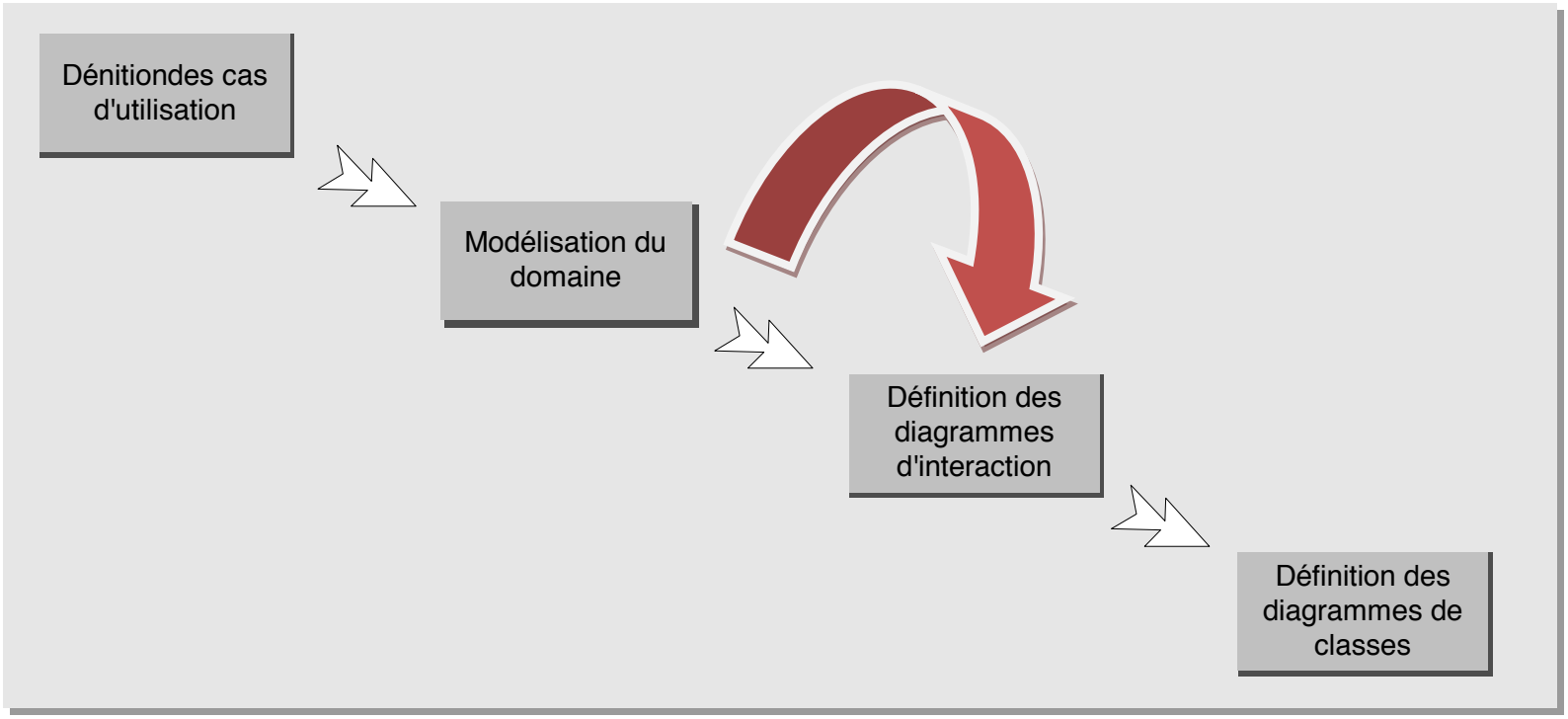
Le monde réel correspond au monde « humain », le monde que l'on désire informatiser, simuler, etc..



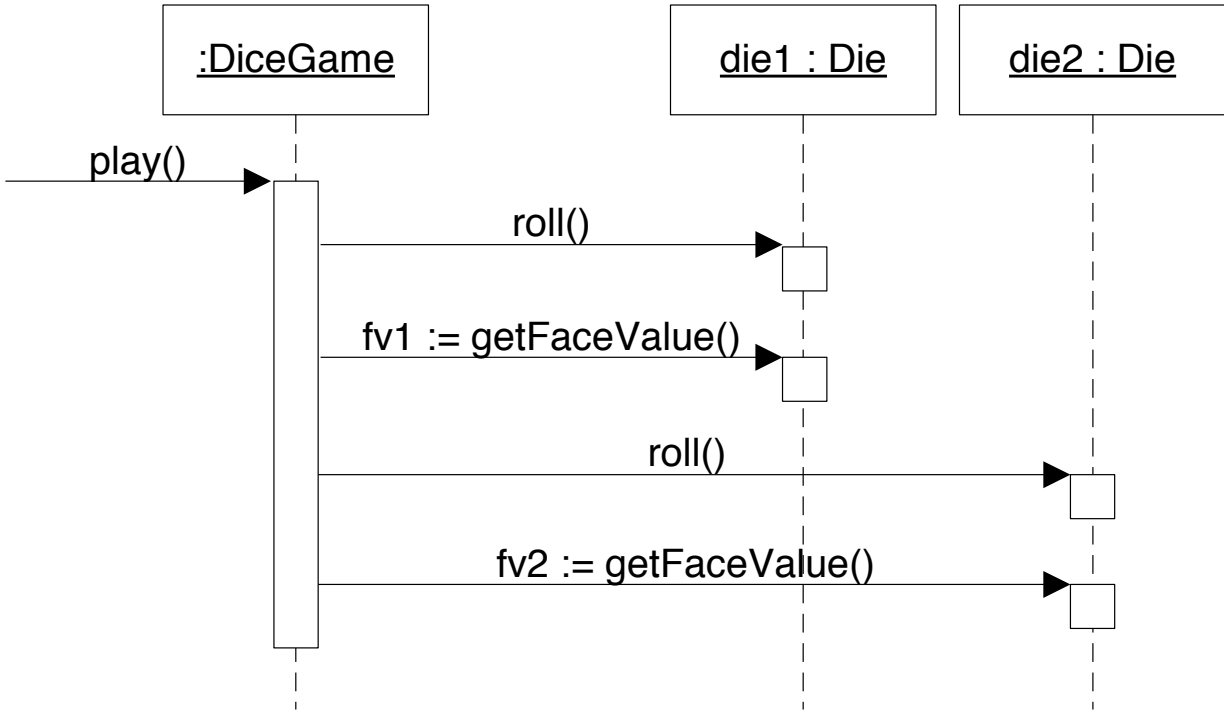


Quelques remarques sur le modèle de domaine..

- Peut faire intervenir le joueur, un **être humain**
- Les méthodes n'apparaissent pas
- Les associations sont bidirectionnelles, elles peuvent se lire dans les deux sens : Le jeu de dés inclut deux dés, ou Chaque dé est inclus dans le jeu de dés.

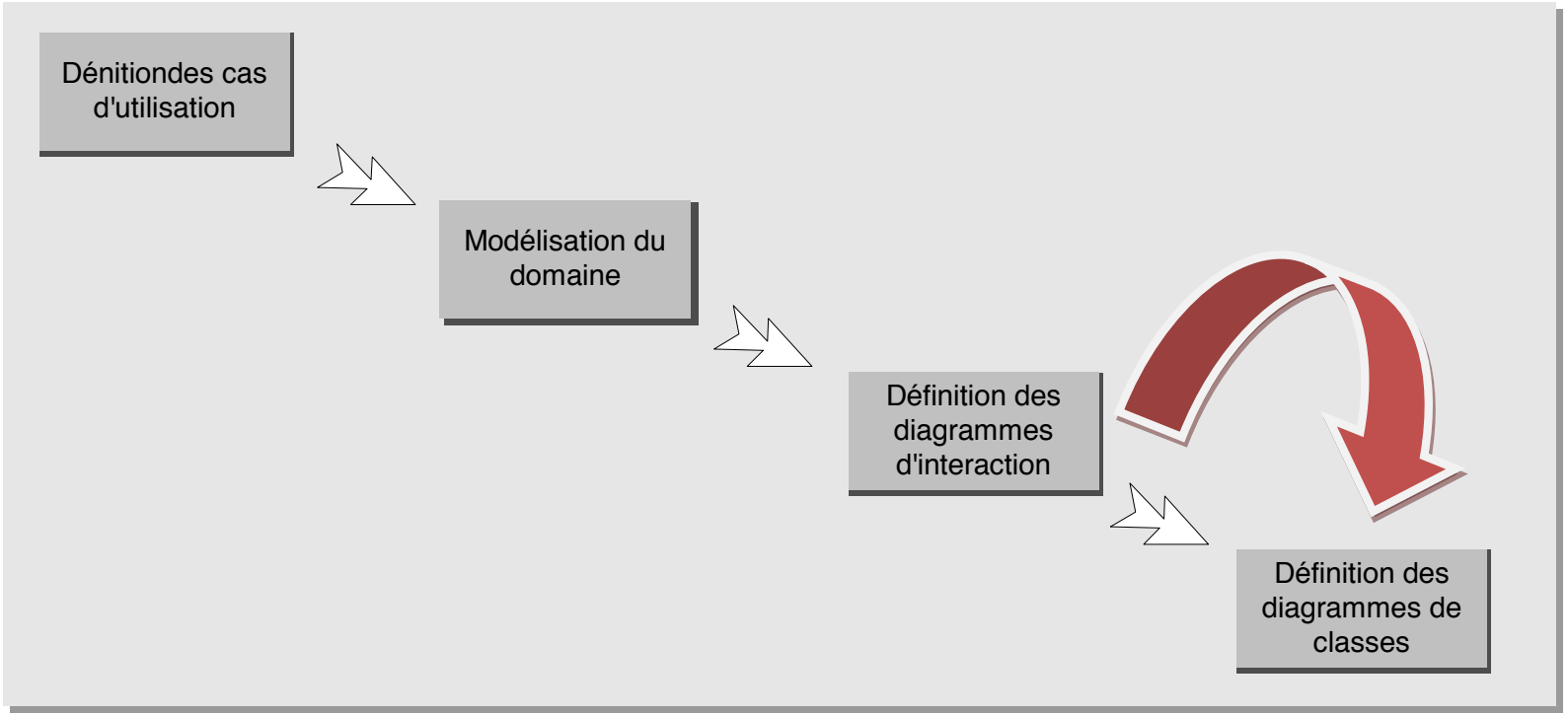

Dénitiondes cas d'utilisation
Modélisation du domaine
Définition des diagrammes d'interaction
Définition des diagrammes de classes

DÉFINITION DES DIAGRAMMES D'INTERACTION



- Représente une **vue dynamique** des collaborations entre objets.
  Modèle de domaine ⇔ **vue statique**

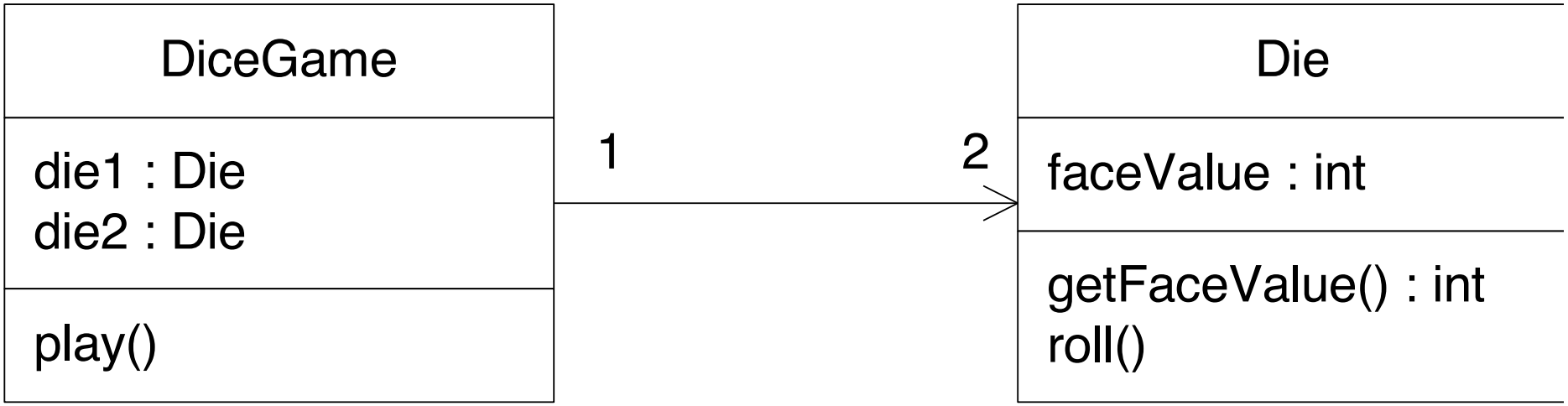
Permet d'identifier les messages (complément du modèle de domaine)



DÉFINITION DES DIAGRAMMES DE CLASSE DE CONCEPTION

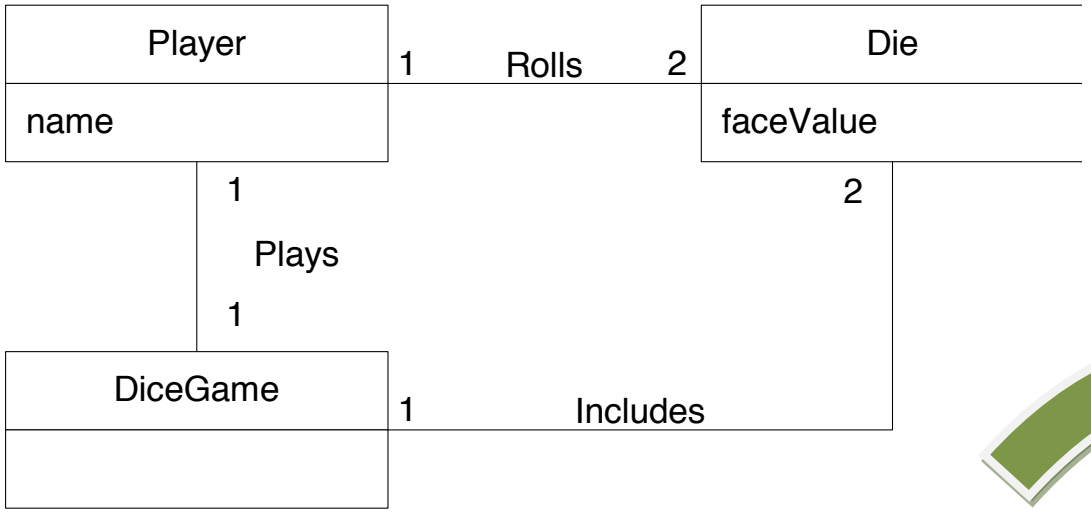
- Représente des classes logicielles
  ⇨ Non plus des concepts du monde réel à informatiser

- **Vue statique** ⇨ Complète le diagramme d'interaction

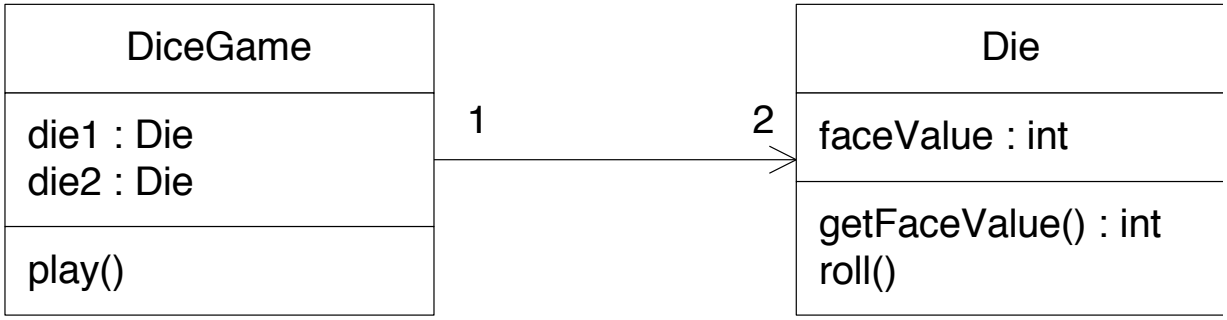


☺ Transition douce entre le modèle de domaine et le diagramme de classe !

DÉFINITION DES DIAGRAMMES DE CLASSE DE CONCEPTION



☺ Transition douce !



QUELQUES MOTS SUR UML

- UML ⇨ **Unified Modeling Language**

- Normalisé en 1997 par l'OMG - Object Management Group.

Définition de l'OMG

- UML est un langage essentiellement graphique

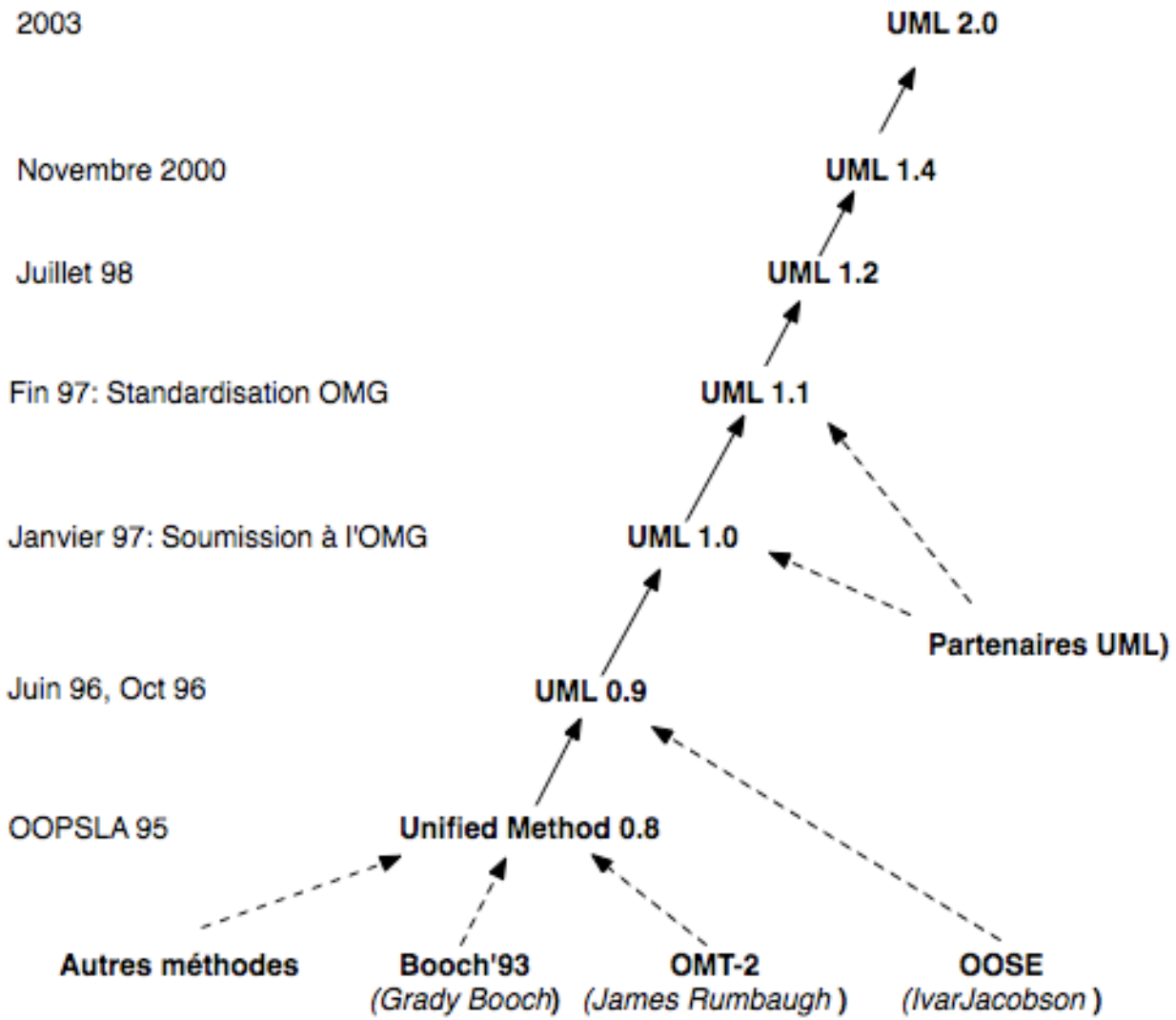
  Il ne s'agit pas d'un langage formel rigoureux !

  ⇨ Pas de génération de code automatique

  ⇨ De ce côté, voir le projet **MDA** (Model Driven Architecture), qui s'appuie sur UML

- **Il ne s'agit pas non plus d'une méthode** !
  UML est un simple outil graphique de modélisation..

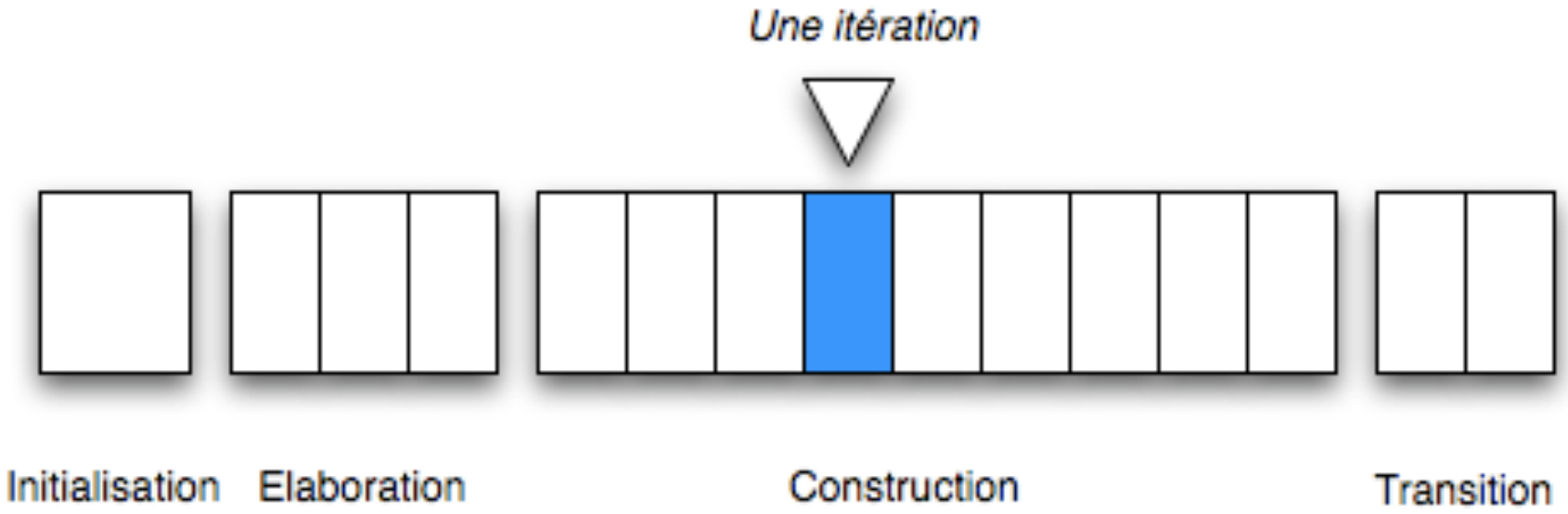
EVOLUTION D'UML



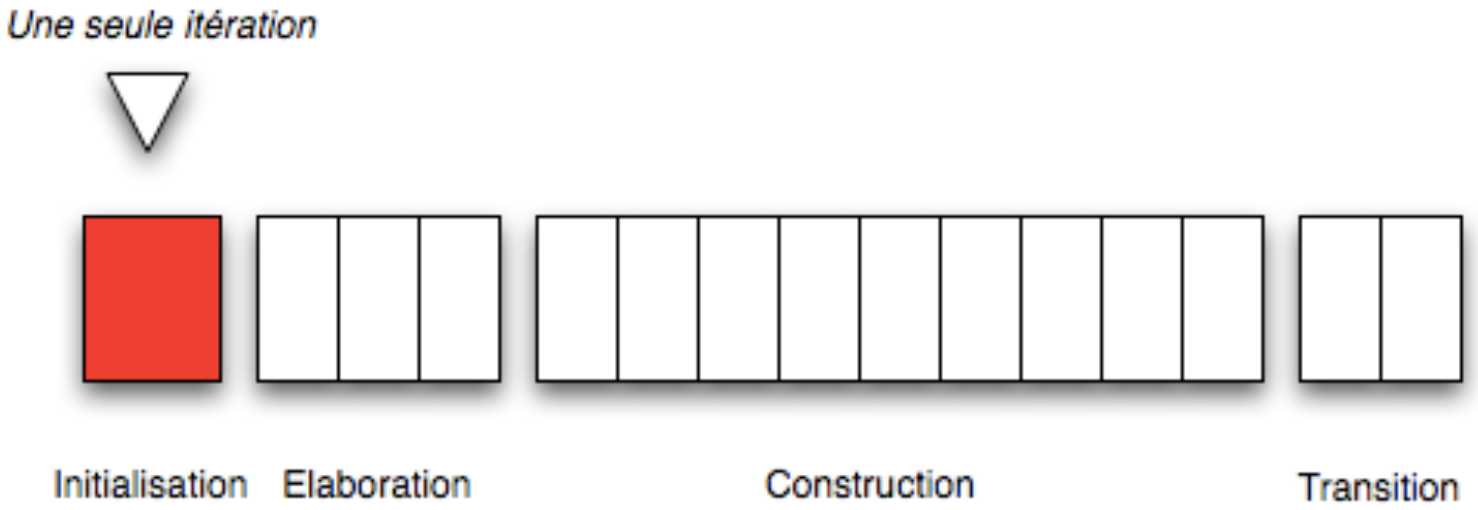
CYCLE DE VIE DE LA MÉTHODE UP

- Le projet est découpé en 4 phases majeures

- Chaque phase est découpée en **itérations** courtes et de « **durée fixe** »



INITIALISATION (*INCEPTION*)

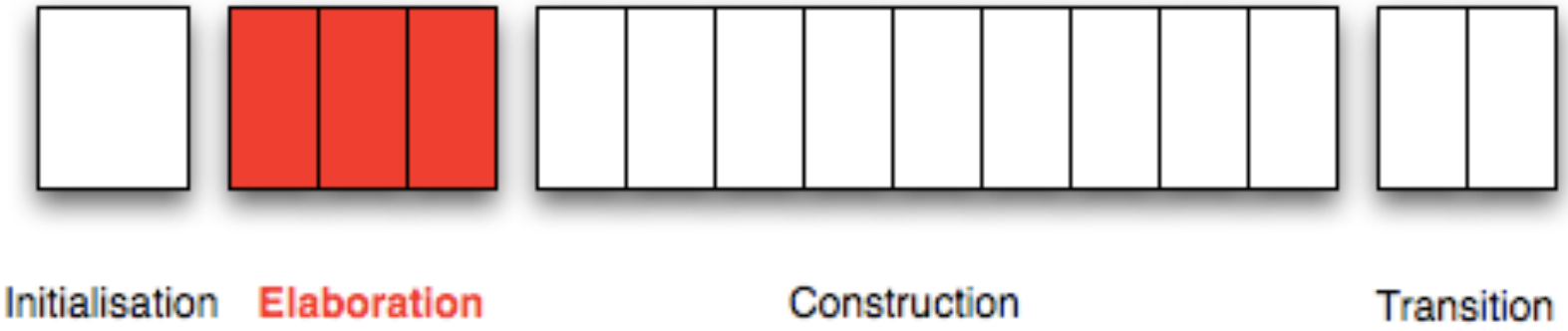


- Compréhension de la finalité du projet
- Etude de faisabilité
- Estimations globales et investigations nécessaires

  ⇨ Décider de la poursuite ou non du projet !

✍ Cette phase doit être la plus courte possible !

ELABORATION



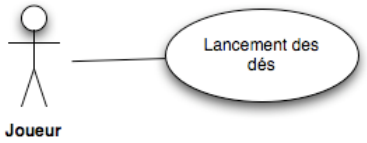
Le projet a été analysé (phase d'initialisation) et accepté !

⇨ Cette phase se focalise essentiellement

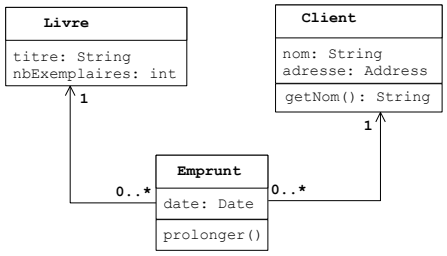
- sur l'analyse des besoins
- et sur la planification de la phase de construction.

ELABORATION – OBJECTIFS 1/3

- 1/ Vision plus élaborée des finalités du projet (objectifs et fonctionnalités)
- 2/ Identification des besoins et des frontières réelles du problème
  - ⇨ Cas d'utilisation



  - ⇨ Modélisation de domaine

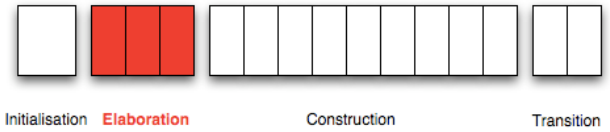


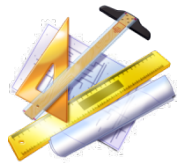
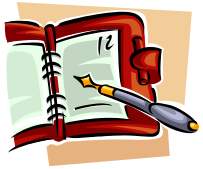

```
class Client {
   private String nom;
   private String adresse;

   public String getNom() {return "";}
}
```

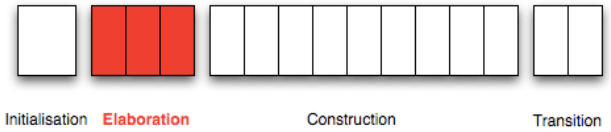
- 3/ Elaboration d'estimations plus réalistes (coût, planification globale)

ELABORATION – OBJECTIFS 2/3



➢ 4/ Implémentation de l'architecture centrale du système	
➢ 5/ Planification des itérations de la phase de construction	
➢ 6/ Résolution des éléments présentant des risques élevés	

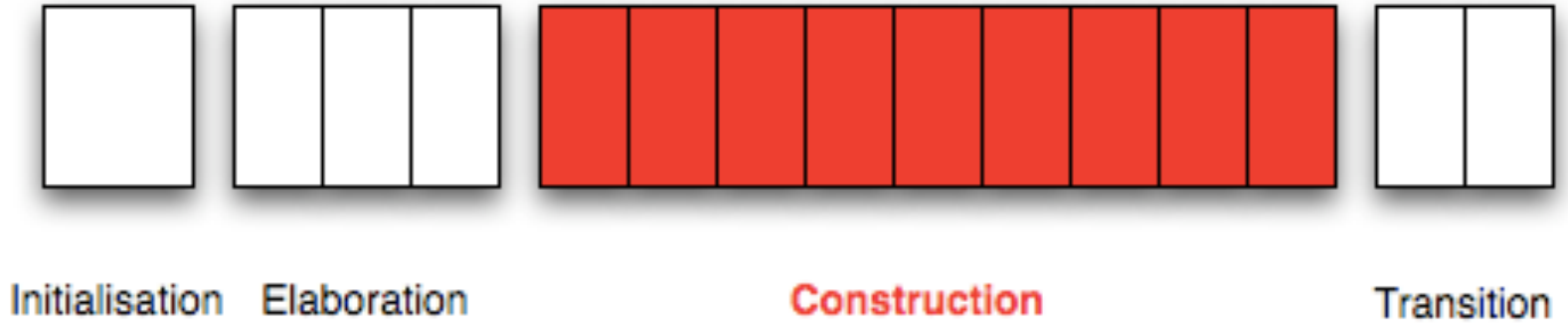
ELABORATION – OBJECTIFS 3/3



☺ On réalise déjà !!

○ 4/ Implémentation de l'architecture centrale du système	
○ 5/ Planification des itérations de la phase de construction	
○ 6/ Résolution des éléments présentant des risques élevés	

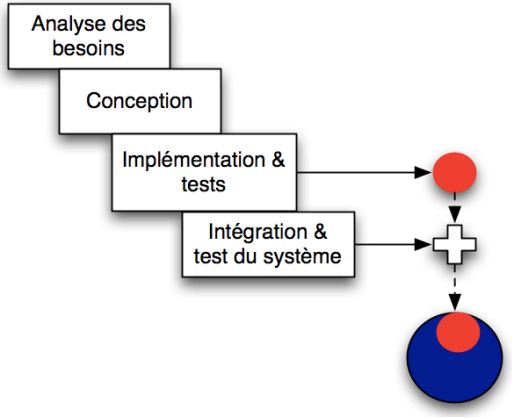
CONSTRUCTION



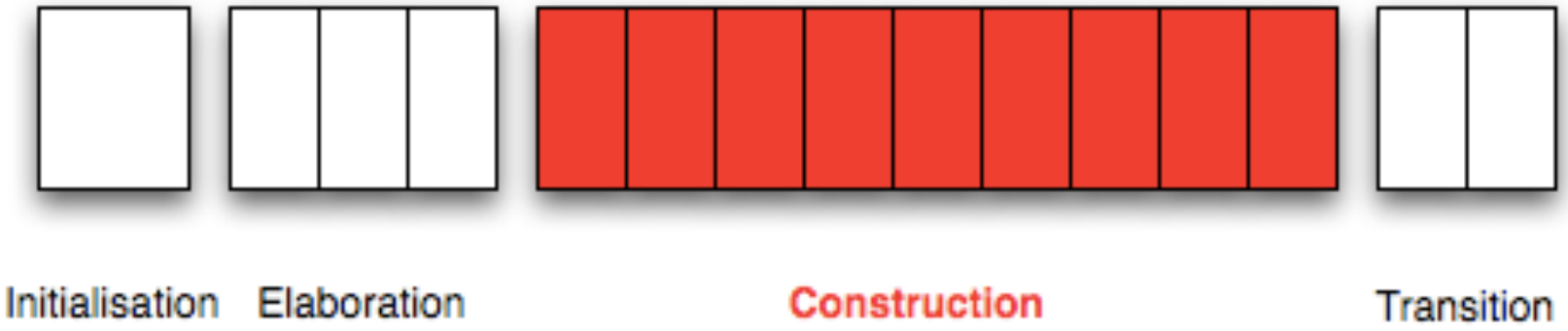
⇨ Développement itératif des différents éléments du système

☝ En commençant par les éléments présentant les plus grands risques

- A chaque itération : un sous-ensemble exécutable stable du produit final
- Toutes les disciplines abordées !
  Analyse (spécifications détaillées), *conception*, *codage* et *test* !



CONSTRUCTION – SUITE..

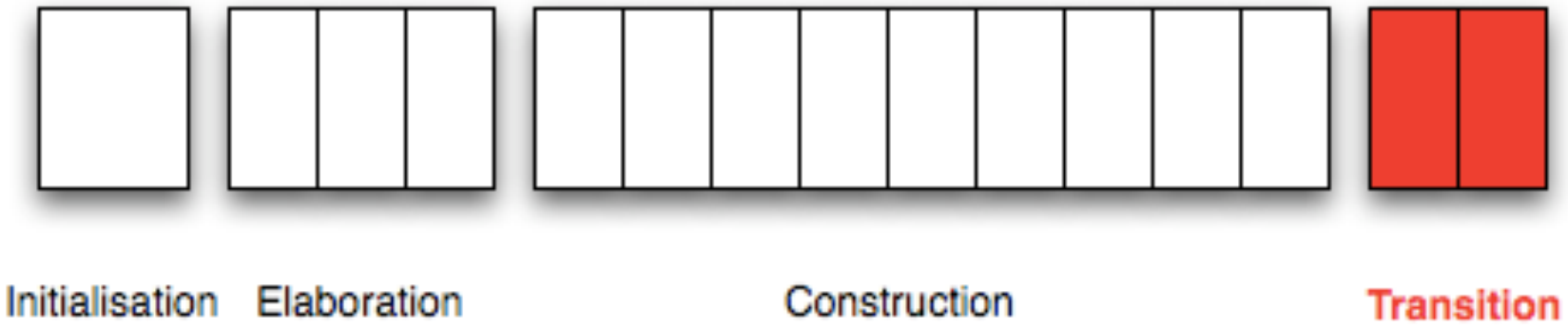


Et si l'équipe de développement se met en retard et ne termine pas à temps dans les délais prévus pour l'itération ?

⇨ L'itération se termine quel que soit le résultat à la date qui a été planifiée !!

⇨ Une replanification sera opérée au début de l'itération suivante

TRANSITION



- Bêta tests et déploiement
- Livraison finale en fin de phase de Transition

TERMINOLOGIE UP: DISCIPLINES, ACTIVITÉS & ARTEFACTS

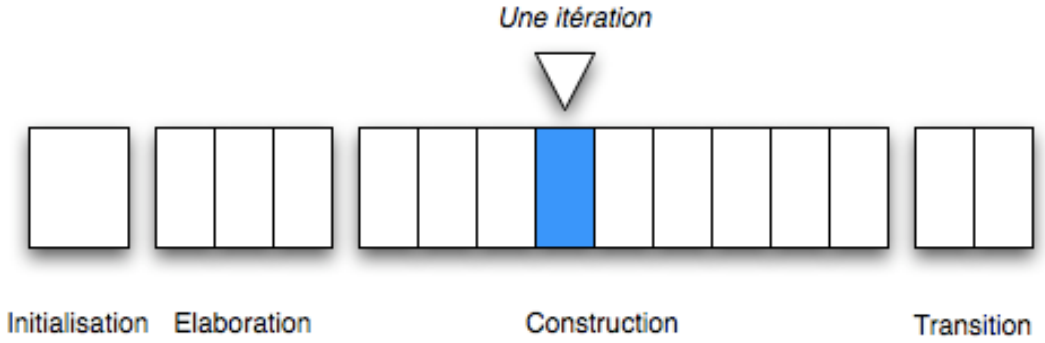
ACTIVITES ⇨ REPERTORIEES ET DECRITES DANS LA DOCUMENTATION UP

Exemples d'activités

- dessiner un diagramme de classes
- définir un cas d'utilisation
- etc..

Une activité est opérée au sein d'une itération

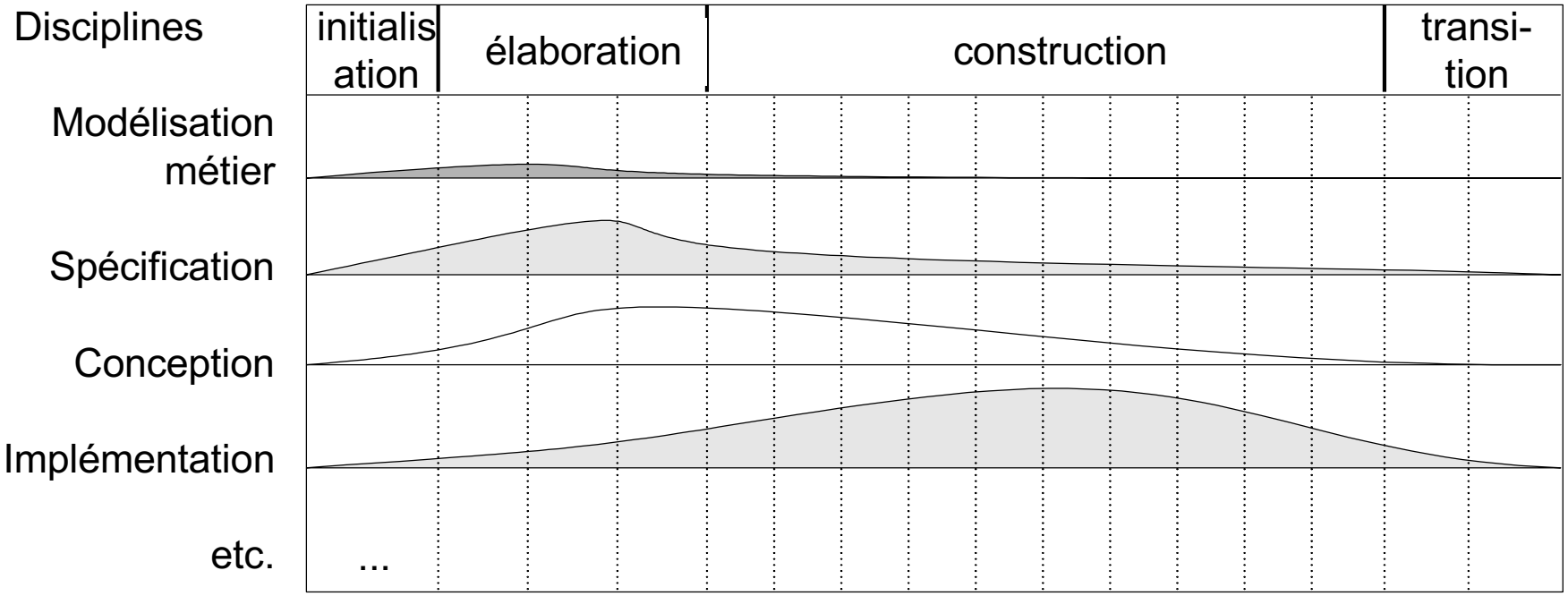
⇨ *Elle commence et se termine dans l'itération*



TERMINOLOGIE UP: DISCIPLINES, ACTIVITÉS & ARTEFACTS

DISCIPLINES ⇨ LE CADRE DANS LEQUEL S'INSCRIT L'ACTIVITÉ

Les disciplines s'étalent sur plusieurs itérations, voire plusieurs phases.



TERMINOLOGIE UP: DISCIPLINES, ACTIVITÉS & ARTEFACTS

ARTEFACT ⇨ UN TERME GÉNÉRIQUE DÉSIGNANT N'IMPORTE QUEL PRODUIT DU TRAVAIL

Exemples

- Un diagramme de classe
- Un diagramme de cas d'utilisation
- Un scénario
- Le code source d'une classe
- Un test unitaire
- Une spécification non fonctionnelle
- Etc..

RETOUR SUR LES DISCIPLINES

UNE ITERATION RECOUVRE LA PLUPART DES DISCIPLINES !