

# Développement Orienté Objet

Christine Solnon

INSA de Lyon - 4IF

2011

# Contexte de l'U.E.

## Domaines d'enseignement de la formation IF à l'INSA de Lyon :

- ▶ Système d'Information
- ▶ Réseaux
- ▶ ...

- ▶ **Développement logiciel**

Unités d'Enseignement (U.E.) du domaine :

- ▶ C++ (3IF)
- ▶ Génie logiciel (3IF)
- ▶ Modélisation UML (3IF)
- ▶ Qualité logiciel (4IF)
- ▶ Grammaires et langages (4IF)
- ▶ Ingénierie des IHM (4IF)
- ▶ **Méthodologie de développement objet** (4IF)
  - ▶ s'appuie sur : UML, C++/Java, génie logiciel
  - ▶ est utilisé par : IHM

# Objectifs de l'U.E.

**Vous connaissez déjà (entre autres...) :**

- ▶ La programmation orientée objet (C++, Java)
- ▶ Le langage de modélisation UML
- ▶ Le génie logiciel

**Vous allez connaître un processus de développement logiciel**

↪ Qui fait quoi et quand pour **bien construire de bons logiciels** ?

- ▶ Processus de développement itératif et incrémental
  - ↪ Unified Software Development Process (USDP)
  - ↪ Sensibilisation aux méthodes agiles et extreme programming
- ▶ Méthodologie orientée objet
  - ↪ Patterns de conception (*Design Patterns*)
- ▶ Mise en pratique dans un projet

# Organisation de l'U.E.

## Cours

- ▶ du 20/09 au 22/09 : 3 cours Développement objet (1ère partie)  
~> Présentation de USDP
- ▶ du 21/09 au 05/10: 7 cours Ingénierie des IHM  
~> Utilisation de USDP
- ▶ du 07/10 au 20/10 : 3 cours Développement objet (2ème partie)  
~> Ingénierie des modèles

## Mise-en-pratique : "Gestion des bagages dans un aéroport"

- ▶ 3 séances de 4h ~> IHM
- ▶ 4 séances de 4h ~> Développement objet

## Evaluation : DS le 9 janvier + note de projet

## Quelques livres à emprunter à DOC'INSA

- ▶ Le processus unifié de développement logiciel  
Ivar Jacobson, Grady Booch, James Rumbaugh  
↪ *Description d'USDP par ses concepteurs*
- ▶ UML 2 et les design patterns  
Craig Larman  
↪ *Mise en œuvre d'USDP dans un esprit AGILE*  
↪ *Bonne introduction aux design patterns*
- ▶ Modélisation Objet avec UML  
Pierre-Alain Muller, Nathalie Gaertner  
↪ *Bon rappel sur UML pour l'analyse et la conception OO*  
↪ *Brève présentation d'USDP*
- ▶ ...et plein d'autres !

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Plan du cours

## Introduction

### Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Crise du logiciel ? (1/3)

## 1979 : Etude du *Government Accounting Office* sur 163 projets

- ▶ 29% des logiciels n'ont jamais été livrés
- ▶ 45% des logiciels ont été livrés... mais n'ont pas été utilisés
- ▶ 19% des logiciels ont été livrés mais ont dû être modifiés pour être utilisés
- ▶ ... ce qui laisse 7% de logiciels livrés et utilisés en l'état

## 1994 : **Système de convoyage des bagages / aéroport de Denver**

- ▶ 193 millions de dollars
- ▶ Livré (partiellement opérationnel) avec 16 mois de retard  
↪ 1,1 million de dollars par jour de retard
- ▶ Remplacé par un système manuel en 2005

## Et quelques bugs qui ont coûté très cher...

- ▶ 1996 : Explosion d'Ariane 5
- ▶ 1999 : Perte de NASA Mars Climate Orbiter

## Crise du logiciel ? (2/3)

### Réussite des projets informatiques (étude du Standish Group)

Année	Succès	Mitigé	Echec
1995	16%	53%	31%
2000	28%	49%	23%
2004	29%	53%	18%
2009	32%	44%	24%

- ▶ Succès : livré à temps, sans dépassement de budget et avec toutes les fonctionnalités initialement spécifiées
- ▶ Mitigé : livré et opérationnel, mais avec moins de fonctionnalités que prévu et un dépassement de budget et/ou de temps
- ▶ Echec : abandonné en cours de route

⇒ **On progresse... mais il reste de la marge pour s'améliorer !**

# Crise du logiciel ? (3/3)

## Principales causes des échecs (étude Standish Group)

1. Manque d'implication de l'utilisateur
2. Exigences et spécifications incomplètes
3. Changements des exigences et spécifications

## Extrait des conclusions de l'étude :

“Research at the Standish Group indicates that **smaller time frames, with delivery of software components early and often, will increase the success rate**. Shorter time frames result in an **iterative process** of design, prototype, develop, test and deploy small elements. This process is known as **growing software** as opposed to the old concept of developing software. Growing software **engages the user earlier**.”

## ↪ **Processus unifié de développement logiciel (USDP)**

- ▶ Processus itératif populaire pour le développement orienté objet
- ▶ Processus flexible et ouvert ↪ Agile et XP compatible !

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Qu'est-ce qu'un logiciel ?

## Ensemble d'artefacts

- ▶ Codes : Sources, Binaires, Tests, ...
- ▶ Documentation pour l'utilisateur : Manuel utilisateur, manuel de référence, tutoriels, ...
- ▶ Documentation interne : Cas d'utilisation, Modèle du domaine, Diagrammes d'interaction, Diagrammes de classes, ...
- ▶ ...

## Conçus par et pour différents acteurs

- ▶ Utilisateurs
- ▶ Analystes et programmeurs
- ▶ Hotline
- ▶ ...

# Qu'est-ce qu'un **bon** logiciel ?

## Différents points de vue

- ▶ L'utilisateur Ce que ça fait ?
  - ↪ besoins fonctionnels ou non fonctionnels
  - ↪ besoins exprimés ou implicites, présents ou futurs, ...
- ▶ L'analyste/programmeur Comment ça le fait ?
  - ↪ architecture, structuration, documentation, ...
- ▶ Le fournisseur Combien ça coûte/rapporte ?
  - ↪ coûts de développement + maintenance, délais, succès ...
- ▶ La hotline Pourquoi ça ne le fait pas/plus ?
  - ↪ diagnostic, reproductibilité du pb, administration à distance, ...
- ▶ ...

# Activités d'un processus logiciel (cf 3IF)

## Capture des besoins

↪ Cahier des charges

## Analyse

↪ Spécification du logiciel

## Conception

↪ Architecture du logiciel

## Réalisation

↪ Codage et tests unitaires

## Intégration et tests

↪ Logiciel livrable

## Maintenance

# Cycle de vie (cf 3IF)

## Modèle linéaire

- ▶ Cycle en cascade
- ▶ Cycle en V

## Modèle incrémental

- ▶ 3 premières activités exécutées en séquence  
    ↪ Spécification et architecture figées
- ▶ Réalisation, intégration et tests effectués incrémentalement

## Critique

Ces modèles supposent que

- ▶ l'analyse est capable de spécifier correctement les besoins
- ▶ ces besoins sont stables

Or, 90% des dépenses concernent la maintenance et l'évolution !

# Maintenance et évolution

## Utilisation des fonctionnalités spécifiées / cycle en cascade

[selon C. Larman]

▶ Jamais.....	45%
▶ Rarement.....	19%
▶ Parfois.....	16%
▶ Souvent.....	13%
▶ Toujours.....	7%

## Répartition des coûts de maintenance [selon C. Larman]

▶ Extensions utilisateur .....	41,8%
▶ Correction d'erreurs .....	21,4%
▶ Modification format de données .....	17,4%
▶ Modification de matériel .....	6,2%
▶ Documentation .....	5,5%
▶ Efficacité .....	4%

**If you think writing software is difficult, try re-writting software**

Bertrand Meyer

# Pourquoi une approche objet ?

## Unicité et universalité du paradigme

- ▶ Réduire le décalage entre monde réel et logiciel

↪ Objets conceptuels ⇒ Objets logiciels

## Réutilisabilité et évolutivité facilitées par différents mécanismes

- ▶ Encapsulation et Modularité
- ▶ Abstraction et Classification
- ▶ Polymorphisme et Héritage
- ▶ ...

↪ Faible couplage inter-objets / Forte cohésion intra-objet

## Paradigme qui arrive à maturité

- ▶ Bibliothèques de classes
- ▶ Principes et patterns de conception (GRASP, GoF)
- ▶ Langage de modélisation graphique (UML)
- ▶ Processus de développement (USDP)

↪ Environnements intégrés

# Le manifeste Agile (2001)

[www.agilealliance.com](http://www.agilealliance.com)

**Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire. Ces expériences nous ont amenés à valoriser :**

- ▶ **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

**Nous reconnaissons la valeur des seconds éléments...  
...mais privilégions les premiers.**

# Quelques principes (choisis) du manifeste Agile

- ▶ Satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
- ▶ Accueillir positivement les changements de besoins, même tard.
- ▶ Livrer fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois.
- ▶ Faire travailler ensemble utilisateurs et développeurs tout au long du projet.
- ▶ Un logiciel opérationnel est la principale mesure d'avancement.
- ▶ La simplicité (l'art de minimiser le travail inutile) est essentielle.
- ▶ À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

Attention : "Etre agile" ne signifie pas "ne pas modéliser"

↪ Modéliser permet de comprendre, de communiquer et d'explorer

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Vue globale de la vie d'un logiciel avec USDP

## La vie d'un logiciel est composée de cycles

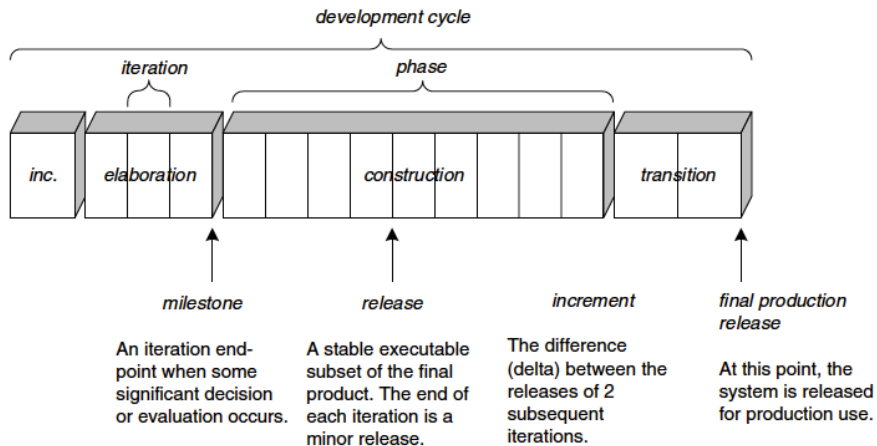
- ▶ 1 cycle  $\Rightarrow$  1 nouvelle version du logiciel
- ▶ Chaque cycle est composé de 4 phases :
  - ▶ Etude préliminaire (*Inception*)
  - ▶ Elaboration
  - ▶ Construction
  - ▶ Transition

## Chaque phase d'un cycle est composée d'itérations

- ▶ 1 itération  $\Rightarrow$  1 incrément
- ▶ Chaque itération est composée d'activités
  - ▶ Capture des besoins
  - ▶ Analyse
  - ▶ Conception
  - ▶ Réalisation
  - ▶ Test

...en proportions variables en fonction du temps

# Vue graphique d'un cycle avec USDP



[figure extraite du livre de C. Larman]

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Phase 1 : Etude préliminaire (*inception*)

- ▶ Phase très courte (souvent une seule itération)
- ▶ Etape préliminaire à l'élaboration
  - ↪ Déterminer la faisabilité, les risques et le périmètre du projet
    - ▶ Que doit faire le système ?
    - ▶ A quoi pourrait ressembler l'architecture ?
    - ▶ Quels sont les risques ?
    - ▶ Estimation approximative des coûts et des délais

↪ **Accepter le projet ?**

## Phase 2 : Elaboration

Quelques itérations courtes et de durée fixe, pilotées par les risques

- ▶ Identification et stabilisation de la plupart des besoins
  - ↪ Spécification de la plupart des cas d'utilisation
- ▶ Conception de l'architecture de base
  - ↪ Squelette du système à réaliser
- ▶ Prog. et test des éléments d'architecture les + importants
  - ↪ Réalisation des cas d'utilisation critiques (<10% des besoins)
  - ↪ Tester au plus tôt, souvent et de manière réaliste
- ▶ Estimation globale du calendrier et des ressources

↪ **Besoins et architecture stables ? Risques contrôlés ?**

### Phase 3 : Construction

Phase la plus coûteuse (>50% du cycle)

- ▶ Développement par incréments
  - ↪ Architecture stable malgré des changements mineurs
- ▶ Le produit contient tout ce qui avait été planifié
  - ↪ Il reste quelques erreurs

↪ **Produit suffisamment correct pour être installé ?**

### Phase 4 : Transition

- ▶ Produit délivré (version bêta)
- ▶ Correction du reliquat d'erreurs
- ▶ Essai et amélioration du produit, formation des utilisateurs, installation de l'assistance en ligne...

↪ **Tests suffisants ? Produit satisfaisant ? Manuels prêts ?**

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

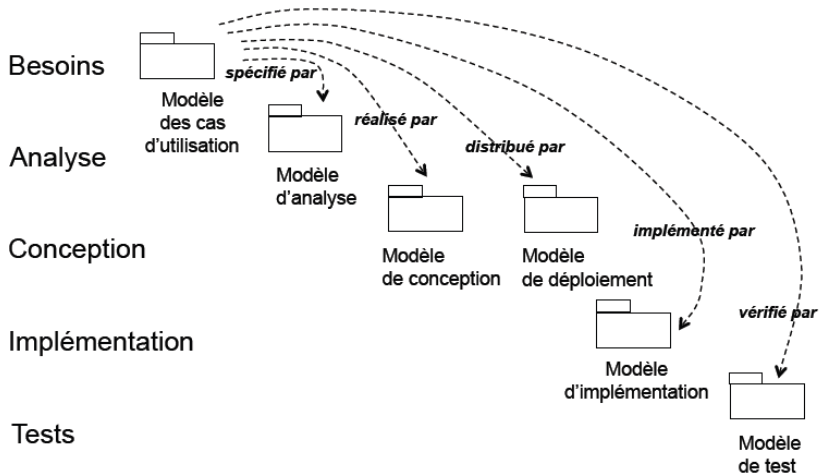
Synthèse des activités par phase

# Processus guidé par les cas d'utilisation

## Les cas d'utilisation :

- ▶ Exemples d'interactions entre les acteurs et le système  
~> Scénarios d'utilisation compréhensibles par tous
- ▶ Permettent de capturer les vrais besoins des utilisateurs  
~> Réfléchir en terme d'acteurs et buts plus que fonctionnalités
- ▶ Guident tout le processus de développement  
~> Garantissent la cohérence du processus
- ▶ Favorisent un développement itératif  
~> Chaque itération est centrée sur un sous-ensemble de cas
- ▶ Développés en tandem avec l'architecture

# Cas d'utilisation / activités



[figure extraite du livre de I. Jacobson, G. Booch, J. Rumbaugh]

# Processus centré sur l'architecture

## Architecture :

- ▶ Vue des aspects les plus significatifs du système  
~> Abstraction des principaux modèles du système
- ▶ Conçue et stabilisée lors des premières itérations  
~> Traiter en premier les cas d'utilisation "pertinents" :
  - ▶ les plus risqués
  - ▶ les plus importants pour l'utilisateur
  - ▶ les plus représentatifs des différentes fonctionnalités

# Processus itératif et incrémental

Itération = mini projet donnant lieu à un incrément

## Avantages :

- ▶ Gestion des risques importants lors des premières itérations  
~> Construire et stabiliser le noyau architectural rapidement
- ▶ Feed-back régulier des utilisateurs  
~> Adaptation permanente du système aux besoins réels
- ▶ Feed-back régulier des développeurs et des tests  
~> Affiner la conception et les modèles
- ▶ Complexité mieux gérée  
~> Eviter la paralysie par l'analyse  
~> Etapes plus courtes et moins complexes
- ▶ Exploitation des erreurs des itérations précédentes  
~> Amélioration du processus d'une itération sur l'autre

# Ce qu'on va voir dans la suite du cours...

- ▶ Description des différentes activités / Itération "générique"
  - ▶ Que faut-il décrire ?
  - ▶ Sous quelle forme ?
  - ▶ Comment faire cela ?

↪ Description technique de la méthode

- ▶ Description des différentes phases
  - ▶ Quel est le but à atteindre ?
  - ▶ Quels sont les livrables ?
  - ▶ Quel niveau d'abstraction ?

↪ Description du déroulement du projet

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Vue globale d'une itération "générique"

## Mini cascade qui enchaîne différentes activités :

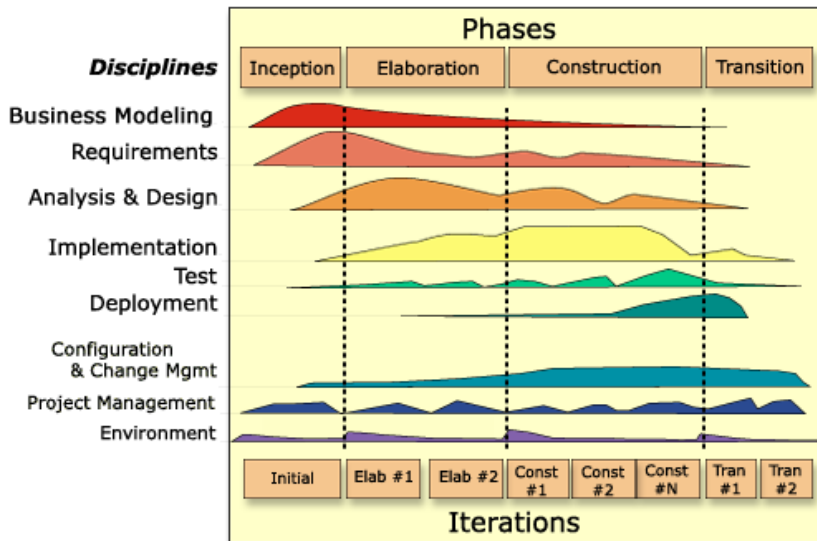
- ▶ Capture et analyse des besoins
  - ↪ Modéliser le système vu de l'extérieur
- ▶ Analyse et Conception Orientées Objet
  - ↪ Modéliser le système vu de l'intérieur
- ▶ Réalisation et tests
  - ↪ Implémenter le système
  - ↪ Valider l'implémentation / besoins

↪ La part de ces activités varie en fonction des phases

## Itérations agiles

- ▶ Itérations de courte durée
  - ↪ Quelques semaines (typiquement entre 2 et 6)
- ▶ Itérations de durée fixée
  - ↪ Réduire les objectifs de l'itération si nécessaire
  - ↪ Reporter une partie des objectifs aux itérations suivantes

# Part des activités d'une itération / Phases



[Figure extraite de <http://www.ibm.com/developerworks/rational/>]

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Capture et analyse des besoins

**Objectif : se mettre d'accord sur le système à construire**

- ▶ Besoins fonctionnels (↔ comportementaux)
- ▶ Besoins non fonctionnels (↔ tous les autres)

## Capture vs analyse des besoins

- ▶ Capture :
  - ▶ Langage du client ↔ Informel, redondant
  - ▶ Structuration par les cas d'utilisation
- ▶ Analyse :
  - ▶ Langage du développeur ↔ Formel, non redondant
  - ▶ Structuration par les classes

## Activités difficiles car :

- ▶ Les utilisateurs ne connaissent pas vraiment leurs besoins
- ▶ Les développeurs ne connaissent pas le domaine de l'appli
- ▶ Utilisateurs et développeurs ont des langages différents
- ▶ Les besoins évoluent
- ▶ Il faut trouver un compromis services/coûts/délais

# Buts et Artefacts

## Buts :

- ▶ Appréhender les besoins fonctionnels  
↪ Artefact : Modèle des cas d'utilisation
  - ▶ Description textuelle
  - ▶ Diagrammes de séquences système
  - ▶ Diagramme des cas d'utilisation
- ▶ Appréhender les besoins non fonctionnels et architecturaux  
↪ Artefact : Exigences supplémentaires
- ▶ Comprendre le contexte du système  
↪ Artefact : Modèle du domaine / de l'entreprise / du métier

## Artefacts transversaux :

- ▶ Vision
- ▶ Glossaire
- ▶ Maquette de l'IHM

# Appréhender les besoins fonctionnels

## Utilisation de cas d'utilisation

Procédé simple permettant au client de décrire ses besoins

- ▶ Parvenir à un accord (contrat) entre clients et développeurs
- ▶ Point d'entrée pour les étapes suivantes du développement

## Qu'est ce qu'un cas d'utilisation ?

- ▶ Usage qu'un acteur (entité extérieure) fait du système
  - ↪ Séquence d'interactions entre le système et les acteurs
- ▶ Généralement composé de plusieurs scénarios (instances)
  - ↪ Scénario de base et ses variantes (↪ cas particuliers)

## Comment découvrir les cas d'utilisation ?

- ▶ Délimiter le système
- ▶ Identifier les acteurs principaux
  - ↪ Ceux qui atteignent un but en utilisant le système
- ▶ Identifier les buts de chaque acteur principal
- ▶ Définir les cas d'utilisation correspondant à ces buts

↪ Atelier d'expression des besoins

# Description textuelle d'un cas d'utilisation

Histoire d'un acteur qui utilise le système **pour atteindre un but** :

- ▶ Abrégée : scénario de base dans un paragraphe
- ▶ Informelle : différents scénarios décrits dans plusieurs parag.
- ▶ Détaillée : description structurée ( cf [alistair.cockburn.us](http://alistair.cockburn.us))

## Exemple : Description abrégée de “Traiter une vente”

Un client arrive à la caisse avec les articles qu'il souhaite acheter. Pour enregistrer les articles, le caissier utilise le système PdV, lequel présente le détail des articles et le montant total des achats, hors taxe et avec taxe. Le client fournit les informations nécessaires pour le règlement. Le système valide et enregistre ces informations, puis met à jour les quantités en stock et imprime le ticket de caisse destiné au client. La vente est terminée et le client peut quitter le magasin.

# Description textuelle d'un cas d'utilisation

Histoire d'un acteur qui utilise le système **pour atteindre un but** :

- ▶ Abrégée : scénario de base dans un paragraphe
- ▶ Informelle : différents scénarios décrits dans plusieurs parag.
- ▶ Détaillée : description structurée ( cf [alistair.cockburn.us](http://alistair.cockburn.us))

## Exemple : Description informelle de “Traiter une vente”

Décrire succinctement les variantes possibles :

- ▶ si le système tombe en panne...
- ▶ si le client est exempté de TVA...
- ▶ si le client change d'avis et enlève un article...
- ▶ ...

# Description textuelle d'un cas d'utilisation

Histoire d'un acteur qui utilise le système **pour atteindre un but** :

- ▶ Abrégée : scénario de base dans un paragraphe
- ▶ Informelle : différents scénarios décrits dans plusieurs parag.
- ▶ Détaillée : description structurée ( cf [alistair.cockburn.us](http://alistair.cockburn.us))

## Exemple : Description détaillée de “Traiter une vente”

- ▶ Nom du cas d'utilisation : Traiter une vente
- ▶ Périmètre : Application XXX
- ▶ Niveau : but utilisateur
- ▶ Acteur principal : le caissier
- ▶ Parties prenantes et intérêts :
  - ▶ Le Caissier : il veut ...
  - ▶ Le Gérant : il veut ...
  - ▶ Le Client : il veut ...
  - ▶ Les Services fiscaux : ils veulent ... ; etc.
- ▶ Préconditions : le Caissier est identifié et authentifié
- ▶ Postconditions : La vente est enregistrée, les quantités en stock sont mises-à-jour, la facture est générée, ...

# Description textuelle d'un cas d'utilisation

Histoire d'un acteur qui utilise le système **pour atteindre un but** :

- ▶ Abrégée : scénario de base dans un paragraphe
- ▶ Informelle : différents scénarios décrits dans plusieurs parag.
- ▶ Détaillée : description structurée ( cf `alistair.cockburn.us`)

## Exemple : Description détaillée de "Traiter une vente" (suite 1)

▶ Cas nominal :

1. Le Client arrive à la caisse avec des articles
2. Le Caissier commence une vente
3. Le Caissier entre le code d'un article et une quantité
4. Le Système présente la description de l'article, son prix unitaire HT et TTC et le total en cours HT et TTC

*Le Caissier répète les étapes 3 et 4 jusqu'à ce que tous les articles soient saisis*

5. Le Système présente le total HT et TTC
6. Le Caissier dit le total au Client et en demande le paiement
7. ...

# Description textuelle d'un cas d'utilisation

Histoire d'un acteur qui utilise le système **pour atteindre un but** :

- ▶ Abrégée : scénario de base dans un paragraphe
- ▶ Informelle : différents scénarios décrits dans plusieurs parag.
- ▶ Détaillée : description structurée ( cf `alistair.cockburn.us` )

## Exemple : Description détaillée de “Traiter une vente” (suite 2)

▶ Extensions :

**2-4a.** Le Client dit qu'il bénéficie d'un taux de TVA différent

1. Le Caissier saisit le code correspondant
2. Le Système enregistre le code

**3a.** Code article invalide

1. Le Système signale l'erreur et rejette la saisie
2. Le Caissier traite l'erreur :

**2a.** Il existe un code lisible par un être humain

...

**2b.** Pas de code, mais le prix est sur l'article

...

**2c.** ...

...

# Description textuelle d'un cas d'utilisation

Histoire d'un acteur qui utilise le système **pour atteindre un but** :

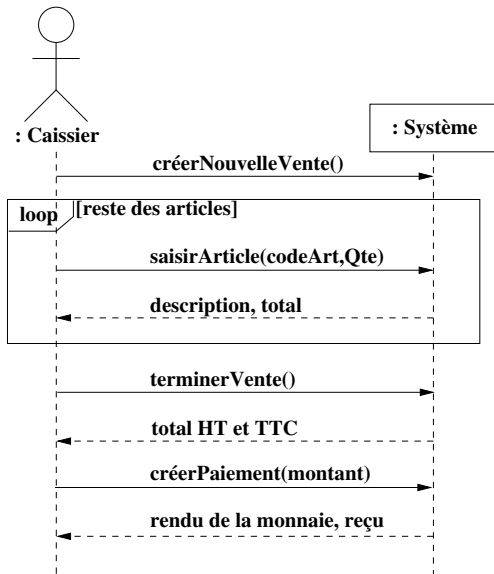
- ▶ Abrégée : scénario de base dans un paragraphe
- ▶ Informelle : différents scénarios décrits dans plusieurs parag.
- ▶ Détaillée : description structurée ( cf `alister.cockburn.us` )

## Exemple : Description détaillée de "Traiter une vente" (suite 3)

- ▶ Spécifications particulières :
  - ▶ Interface utilisateur à écran tactile
  - ▶ Temps de réponse ...
  - ▶ ...
- ▶ Fréquence d'occurrence : potentiellement continue
- ▶ Divers :
  - ▶ Le caissier doit-il emporter son tiroir-caisse lorsqu'il se déconnecte ?
  - ▶ ...

# Diagrammes de séquence d'un cas d'utilisation

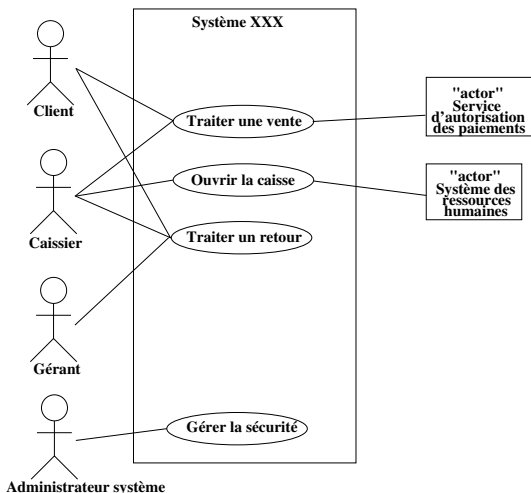
↪ Représentation graphique d'un scénario d'un cas d'utilisation



# Diagramme des cas d'utilisation

“Table des matières” graphique des cas d'utilisation

↔ Relations entre cas d'utilisation et acteurs



# Appréhender les besoins non fonctionnels

## Exigences supplémentaires

↪ Description des besoins non fonctionnels

- ▶ Certains de ces besoins sont déjà dans les cas d'utilisation
  - ↪ Les regrouper pour éviter les doublons
- ▶ Lister également tous ceux qui ne sont pas dans les cas d'util.

## Moyen mémotechnique (?) pour recenser les besoins : FURPS+

**Functionality** : Fonctions, capacités, sécurité

**Usability** : Facteurs humains, aide, documentation

**Reliability** : Fréquence des pannes, possibilité de récupération

**Performance** : Temps de réponse, débit, exactitude, ressources

**Supportability** : Adaptabilité, maintenance, configurabilité

**+** : Facteurs complémentaires

- ▶ Langages et outils, matériel, etc
- ▶ Contraintes d'interfaçage avec des syst. externes
- ▶ Aspects juridiques, licence
- ▶ ...

# Besoins non fonctionnels et Analyse architecturale

## Objectif :

- ▶ Identifier les facteurs pouvant influencer sur l'architecture :
  - ▶ Points de variation
  - ▶ Fiabilité et possibilités de récupération
  - ▶ Performance
  - ▶ ...
- ▶ Etudier leurs potentielles évolutions futures  
↪ Points d'évolution
- ▶ Attribuer des priorités
- ▶ Proposer des solutions architecturales  
↪ Décisions architecturales

## Artefact : Tableau des facteurs architecturaux

↪ Inclut dans les exigences supplémentaires

# Comprendre le contexte du système

## Modèle du domaine (Diagramme de classes)

Représentation visuelle

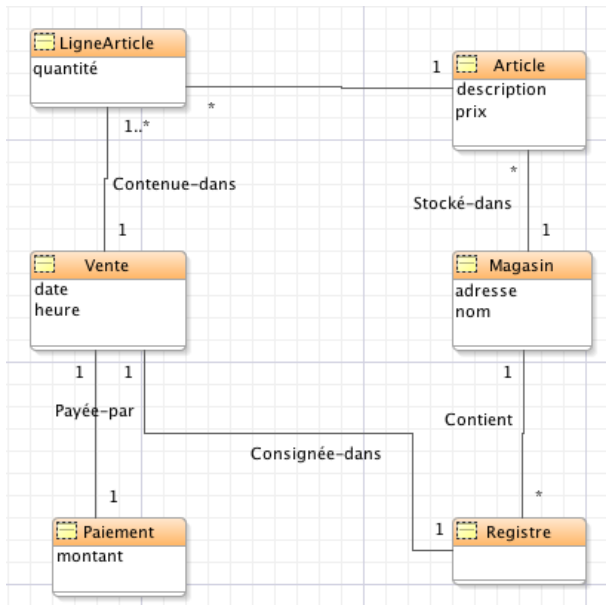
- ▶ des objets du domaine  $\rightsquigarrow$  classes conceptuelles et attributs
- ▶ de leurs relations  $\rightsquigarrow$  associations

$\rightsquigarrow$  Réduire le décalage des représentations avec les objets logiciels

## Comment construire un modèle du domaine ?

- ▶ Réutiliser (et modifier) des modèles existants !
- ▶ Utiliser une liste de catégories :
  - ▶ Classes : *Transactions métier, Lignes de transactions, Produits/services liés aux transactions, Acteurs, Lieux, ...*
  - ▶ Associations : *est-une-description-de, est-membre-de, ...*
- ▶ Identifier les noms et groupes nominaux
  - $\rightsquigarrow$  Utiliser les descriptions textuelles des cas d'utilisation

# Exemple de Modèle du domaine



# Autres artefacts de la capture et l'analyse des besoins

## Modèle du métier / Modèle de l'entreprise

- ▶ Décrit les activités et processus de l'entreprise  
~> Diagrammes d'activité et de flots de données

## Glossaire

- ▶ Définit les termes les plus importants  
~> Evite les ambiguïtés
- ▶ Dérivé des cas d'utilisation et modèles du domaine/métier

## Vision

- ▶ Vue globale synthétique du projet  
~> Résumé des cas d'utilisation et exigences supplémentaires

## Maquette de l'IHM

- ▶ Uniquement si IHM complexe  
~> Validation du client

# Synthèse des artefacts de la capture et l'analyse des besoins

## Compréhension du domaine

- ▶ Modèle du domaine / du métier
- ▶ Glossaire

## Spécification des besoins

- ▶ Vision
- ▶ Modèle des cas d'utilisation
  - ▶ Description textuelle
  - ▶ Diagrammes des cas d'utilisation et de séquences système
  - ▶ Classement des cas d'utilisation / Risque et Difficulté
- ▶ Exigences supplémentaires
  - ↔ incluant le tableau des facteurs architecturaux
- ▶ Maquettes IHM

## Modélisation Agile :

- ▶ Artefacts créés en équipe et en parallèle
  - ↔ Ateliers d'expression des besoins
- ▶ Ne pas se focaliser sur les détails de notation

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Buts et Artefacts

## Appréhender la logique comportementale (vue dynamique)

- ▶ Comprendre et décrire les interactions entre objets via l'envoi de messages (corps des méthodes)
- ▶ Conception pilotée par les responsabilités

↪ Artefact : Diagrammes d'interaction (séquence et communication)

## Appréhender la logique structurelle (vue statique)

- ▶ Structurer en classes  
(attributs, signatures des méthodes, relations inter-classes)

↪ Artefact : Diagrammes de classes

- ▶ Concevoir l'architecture logique (packages)

↪ Artefact : Diagrammes de packages

- ▶ Concevoir l'architecture de déploiement

↪ Artefact : Diagramme de déploiement

## Documenter l'architecture

↪ Artefact : Document d'architecture du logiciel (N+1 vues)

## **Influence des artefacts de “Capture et analyse des cas d'utilis.”**

### **Diagramme de classes du Modèle du domaine**

- ▶ Point de départ pour le Diagramme de classes conceptuelles
- ▶ Mais ajout de nouvelles classes logicielles, interfaces, attributs, visibilité, contraintes, affectation des méthodes aux classes, ...

### **Descriptions et Diagrammes de séquence des Cas d'utilisation**

- ▶ Point de départ des Diagrammes d'interaction
- ▶ Mais apparition de nouveaux objets logiciels, nouveaux messages, granularité plus fine, ...

### **Tableau des facteurs architecturaux des Exigences sup.**

- ▶ Guide le choix des Architectures logique et de déploiement

**Réduire le décalage des représentations...**

# Conception orientée objet et Conception Agile

## Objectif premier de la modélisation :

Comprendre et communiquer :

- ▶ Quelles sont les responsabilités des objets ?
- ▶ Quelles sont les collaborations entre objets ?
- ▶ Quels patterns de conception peut-on utiliser ?

↪ La doc. peut être générée à partir du code (reverse engineering)

## Modélisation Agile

- ▶ Modéliser en groupe
- ▶ Créer plusieurs modèles en parallèle
  - ↪ Diagrammes dynamiques (interactions)
  - ↪ Diagrammes statiques (classes, packages et déploiement)
- ▶ Concevoir avec les programmeurs et non pour eux !

# Diagrammes d'interaction

**Objectif : illustrer les interactions entre objets**

- ▶ Diagrammes de séquence : structuration en termes de
  - ▶ temps  $\rightsquigarrow$  axe vertical
  - ▶ objets  $\rightsquigarrow$  axe horizontal
- ▶ Diagrammes de communication : structuration en multigraphe
  - $\rightsquigarrow$  Numérotation des arcs pour modéliser l'ordre des interactions

$\rightsquigarrow$  **Réfléchir à l'affectation de responsabilités aux objets**

- ▶ Qui crée les objets ?
- ▶ Qui permet d'accéder à un objet ?
- ▶ Quel objet reçoit un message provenant de l'IHM ?
- ▶ ...

de façon à avoir un faible couplage et une forte cohésion

**Elaborés en parallèle avec les diagrammes de classes**

# Diagrammes de classes

## Réfléchir à la structuration

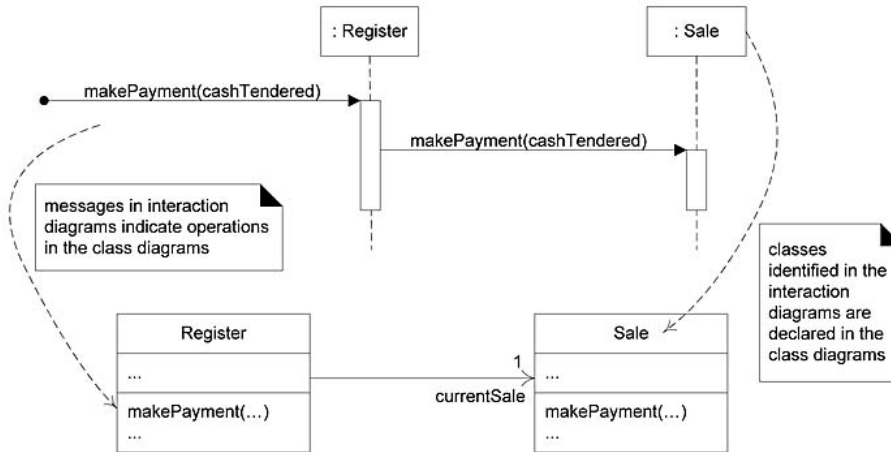
- ▶ Classes
  - ↪ 3 stéréotypes : frontière, contrôle et entité
  - ↪ Mots clés (“actor”, “interface”, {abstract})
- ▶ Attributs et Opérations (signatures)
  - ↪ Visibilité (+,-)
- ▶ Associations, Relations d'héritage et de dépendance
- ▶ Contraintes

de façon à avoir un faible couplage et une forte cohésion

**Elaborés en parallèle avec les diagrammes d'interaction**

↪ Contrôler la cohérence des diagrammes !

# Liens entre diag. de classes et d'interaction



[Figure extraite du livre de C. Larman]

# Quelques patterns de conception GRASP

## Qu'est-ce qu'un pattern ?

Description nommée d'un problème (générique) et de sa solution

↪ Réutilisation de solutions

↪ Facilite le dialogue

## Différents types de patterns :

- ▶ Patterns de conception orientée objet
  - ▶ GRASP (General Responsibility Assignment Software Patterns)
    - ↪ 9 patterns pour affecter des responsabilités aux objets
    - ↪ Principes de base (de bon sens ?)
  - ▶ GoF (Gang of Four : Gamma, Helm, Johnson, Vlissides)
    - ↪ 23 patterns de conception
  - ▶ ...
- ▶ Patterns architecturaux (cf suite du cours)
- ▶ ...

# Exemples de patterns GRASP (1/2)

## Créateur

- ▶ Pb : Qui crée un A ?
- ▶ Sol : Affecter à un B la responsabilité de créer un A si
  - ▶ un B contient ou agrège des A
  - ▶ et/ou un B enregistre des A
  - ▶ et/ou un B utilise étroitement des A
  - ▶ et/ou un B possède les informations d'initialisation des A

## Expert en Information (“faire soi-même”)

- ▶ Pb : Comment affecter des responsabilités aux objets ?
- ▶ Sol : Affecter la responsabilité à la classe qui possède les informations nécessaires pour s'en acquitter

## Faible Couplage

- ▶ Pb : Comment réduire l'impact des modifications
- ▶ Sol : Affecter les resp. de sorte d'éviter les couplages inutiles

↔ Expert et Créateur soutiennent Faible Couplage

# Exemples de patterns GRASP (2/2)

## Forte cohésion

- ▶ Pb : Comment assurer que les objets soient compréhensibles, faciles à gérer et à réutiliser ?
- ▶ Sol : Affecter les resp. de sorte que la cohésion soit forte

## Polymorphisme

- ▶ Pb : Comment gérer des alternatives dépendantes des types ?
- ▶ Sol : Affecter la responsabilité (en utilisant des opérations polymorphes) aux types pour lesquels le comportement varie

## Indirection

- ▶ Pb : Comment éviter le couplage entre différentes entités
- ▶ Sol : Affecter les resp. à des objets intermédiaires

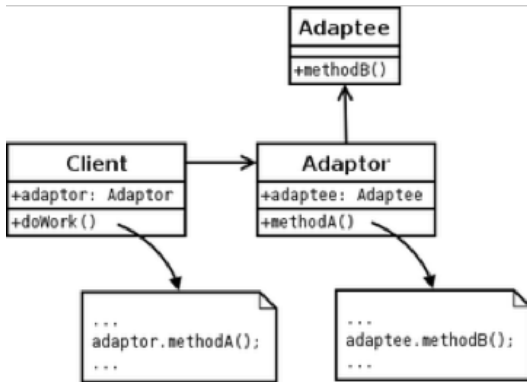
↔ Souvent combiné avec polymorphisme

**et aussi : Contrôleur, Protection des variations, Fabrication pure**

# Exemple de pattern GoF

## Adapter (wrapper)

- ▶ Pb : Comment fournir une interface stable à un composant dont l'interface peut varier (Adaptee)
- ▶ Sol : Ajouter un adaptateur (Adaptor) intermédiaire



[Figure extraite de [http://en.wikipedia.org/wiki/Adapter\\_pattern](http://en.wikipedia.org/wiki/Adapter_pattern)]

Comment faire s'il y a plusieurs composants (Adaptee) différents, et que l'on veut pouvoir choisir en fonction d'une variable système ?

# Architecture logique

## Qu'est-ce qu'une architecture logique ?

- ▶ Regroupement des classes logicielles en packages  
↪ Point de départ pour un découpage en sous-systèmes

## Objectifs d'un découpage en packages

- ▶ Encapsuler et décomposer la complexité
- ▶ Faciliter le travail en équipe
- ▶ Favoriser la réutilisation et l'évolutivité

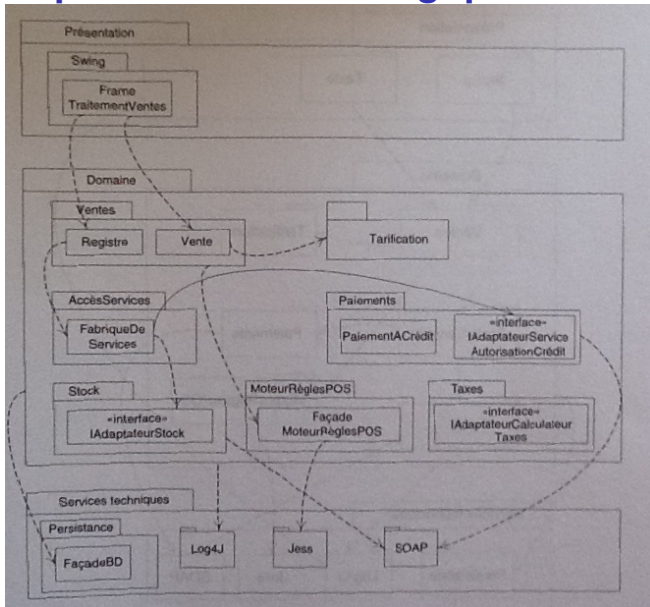
**Forte cohésion, Faible couplage et Protection des variations**

## Utilisation d'UML pour décrire une architecture logique

Diagramme de packages :

- ▶ Relations d'imbrication entre classes et packages ou entre packages
- ▶ Dépendances entre packages

# Exemple d'architecture logique en couches



[Figure extraite du livre de C. Larman]

# Patterns d'architectures logiques

## Exemples de patterns architecturaux

- ▶ Layers (stricte ou lâche)  
    ↔ Présentation, Application, Domaine, ServicesTech., ...
- ▶ Pipes and filters
- ▶ Multitiers
- ▶ Model-View-Controller
- ▶ Peer-to-peer
- ▶ Blackboard
- ▶ ...

# Architecture de déploiement

## Objectif

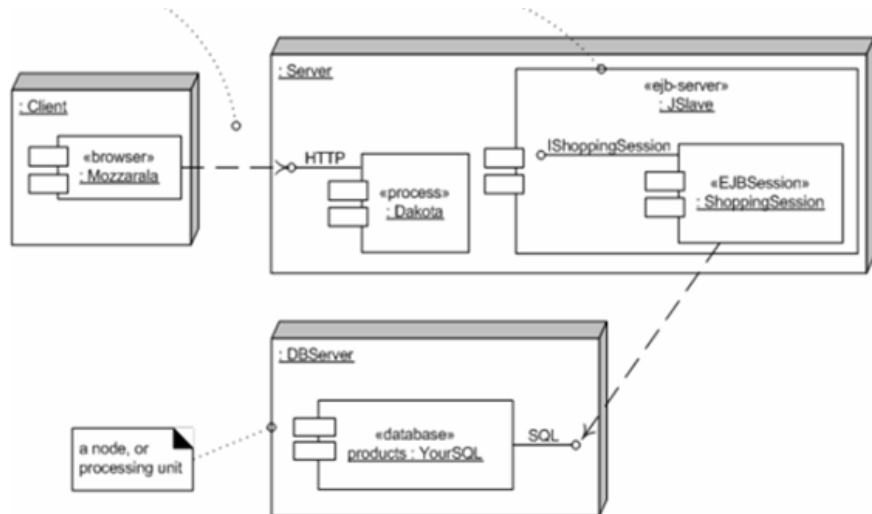
Décrire :

- ▶ la distribution des éléments logiciels sur les composants physiques
- ▶ la communication entre les composants physiques (réseau)

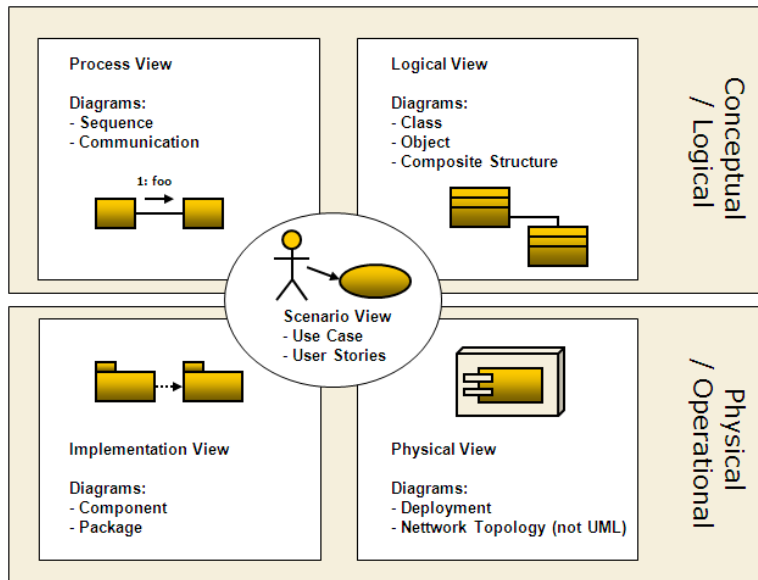
## Utilisation de diagrammes de déploiement

- ▶ Nœuds physiques  
↪ dispositifs physiques (ordinateur, téléphone, ...)
- ▶ Nœuds d'environnement d'exécution  
↪ Ressource logicielle :
  - ▶ s'exécutant au sein d'un nœud externe
  - ▶ offrant un service pour héberger/exécuter d'autres logiciels  
(par ex. : O.S., Machine virtuelle, Navigateur Web, SGBD, ...)
- ▶ Liens  
↪ Moyens de communication entre les nœuds

# Exemple de diagramme de déploiement



# 4+1 vues architecturales [P. Kruchten 95]



[Figure extraite de [wiki.cantara.no/display/dev/4+plus+1+View+Model](http://wiki.cantara.no/display/dev/4+plus+1+View+Model)]

# Documenter l'architecture

## Objectif : Décrire les grandes lignes de l'architecture

↪ Comprendre rapidement les idées essentielles du système

## Structure du Document d'Architecture du Logiciel

- ▶ Représentation architecturale : Comment l'architecture est décrite dans ce document
- ▶ Facteurs architecturaux : Ref. au Tableau des facteurs
- ▶ Décisions architecturales : ensemble de mémos techniques
  - ▶ Nom du problème
  - ▶ Résumé de la solution
  - ▶ Facteurs architecturaux concernés
  - ▶ Solution
  - ▶ Motivation
  - ▶ Problèmes non résolus
  - ▶ Autres solutions envisagées
- ▶ N+1 vues : Logique, Processus, Impl., Physique, ..., + Cas d'util.  
Pour chaque vue :
  - ▶ Décrire les modèles les plus importants
  - ▶ Motiver et commenter les choix

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# De la conception à la réalisation

## Objectifs :

- ▶ Ecrire le code implémentant les cas d'utilisation ciblés
- ▶ Vérifier que ce code répond bien aux besoins

## Ordre d'implémentation des packages et classes :

↪ du moins couplé au plus couplé

## Génération automatique d'un squelette de code

- ▶ Diagrammes de classes
  - ▶ Définition des classes, attributs, signatures de méthodes, ...
  - ▶ Codage des associations 1-n par des collections
  - ▶ ...
- ▶ Diagrammes d'interaction :
  - ▶ Corps (partiel) des méthodes
  - ▶ Signature des constructeurs

# Développement itératif et Reverse engineering

## Le squelette généré automatiquement doit être complété

- ▶ Implémentation de la visibilité
  - ↪ Aptitude d'un objet  $a$  à envoyer un message à un objet  $b$ 
    - ▶ Visibilité persistante :
      - ▶ Visibilité d'attribut :  $b$  est attribut de  $a$
      - ▶ Visibilité globale :  $b$  est un Singleton
    - ▶ Visibilité temporaire :
      - ▶ Visibilité de paramètre :  $b$  est paramètre d'une méthode de  $a$
      - ▶ Visibilité locale :  $b$  est variable locale d'une méthode de  $a$
  - ▶ Traitement des exceptions et des erreurs
  - ▶ ...

## En général, il doit aussi être modifié

↪ Ajout de nouveaux attributs, méthodes, classes, ...

## Utiliser des outils de reverse engineering à la fin de l'itération

↪ Mise-à-jour des modèles de conception pour l'itération suivante

# Développement piloté par les tests

↪ Pratique popularisée par XP

## Principe :

- ▶ Ecrire le code des tests avant d'impl. les méthodes/classes
- ▶ Alternier l'écriture de tests et l'implémentation

## Avantages

- ▶ Les tests unitaires sont réellement écrits (!)
- ▶ Satisfaction du programmeur :  
Objectif clairement défini... défi à relever !
- ▶ Clarification de détails du comportement des méthodes
- ▶ Automatisation et répétabilité du processus de test  
↪ Utilisation d'outils (JUnit, CTest, ...)
- ▶ Assurance lors des modifications

# Refactoring régulier

## Objectif :

Réécrire/restructurer du code sans en modifier le comportement

↪ supprimer les “odeurs de code”

## Transformations par petites étapes

- ▶ Eliminer la duplication de code ↪ Factoriser
- ▶ Améliorer la lisibilité ↪ Renommer
- ▶ Raccourcir les méthodes trop longues ↪ Pliage
- ▶ Supprimer l'emploi des constantes codées en dur
- ▶ Réduire le nombre de variables d'instance d'une classe
- ▶ ...

Attention : réexécuter les tests après chaque étape

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Gestion de projet

## Planification à deux niveaux différents :

- ▶ Plan de phase
  - ↪ Macro plan visible de l'extérieur sur lequel l'équipe s'engage
    - ▶ Jalons et objectifs généraux
      - ↪ Peu de jalons : fins de phases, tests "pilotes" à mi-phase
    - ▶ 1ère estimation **peu fiable** des jalons à la fin de l'inception
    - ▶ Estimation **fiable et contractuelle** à la fin de l'élaboration
      - ↪ Engagement de l'équipe sur la livraison du projet
- ▶ Plan d'itération
  - ↪ Micro organisation interne d'une itération
    - ▶ Le plan de l'itération  $n$  est fait à la fin de l'itération  $n - 1$
    - ▶ Adapter les plans d'itérations en fonction des évolutions

## Gestion des versions

- ▶ Structurer les artefacts par discipline ↪ un répertoire / discipline (sauf implémentation qui est parfois sur plusieurs répertoires)
- ▶ Utiliser un outil de gestion de versions (par ex., SVN)
  - ↪ Créer un point de contrôle à la fin de chaque itération

# Plan d'itération

En fin d'itér.  $n$ , planifier l'itér.  $n+1$  avec toutes les parties prenantes :

↪ Clients, Développeurs, Architecte, Chef de projet, ...

- ▶ Déterminer la durée (en général de 2 à 6 semaines)
- ▶ Lister les objectifs potentiels :  
nouvelles fonctionnalités / cas d'utilisation / scénarios de cas d'utilisation, traitement d'anomalies, ...
- ▶ Classer les objectifs par ordre de priorité :  
↪ Obj. commerciaux du client / Obj. techniques de l'architecte
- ▶ Pour chaque objectif pris par ordre de priorité :
  - ▶ Etudier rapidement l'objectif
  - ▶ Estimer les tâches à faire pour l'atteindre
  - ▶ Quantifier temps nécessaire / ressources humaines dispo.

Jusqu'à ce que volume de travail total = durée de l'itération

Impliquer toute l'équipe dans le processus de planification, et non juste le chef de projet

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

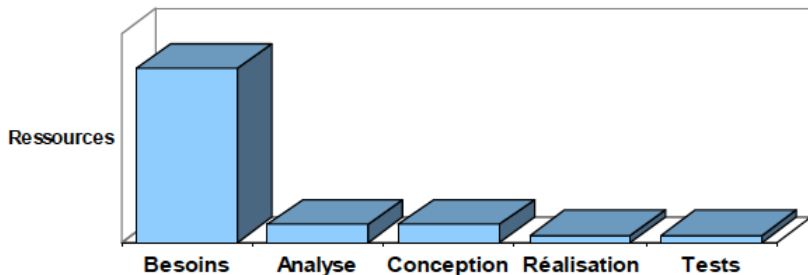
Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

## Phase 1 : Etude préliminaire (*Inception*)



### Objectif : lancer le projet

- ▶ Établir les contours du système et spécifier sa portée
- ▶ Estimer les risques les plus importants

↪ Phase très courte (1 itération)

**A la fin de cette phase on décide de continuer ou non**

[Figure empruntée à J.-L. Sourrouille]

# Activités principales de l'étude préliminaire

## Capture et analyse des besoins

- ▶ Comprendre le contexte du système (~ 50-70% du contexte)
- ▶ Identifier les cas d'utilisation (~ 80%)
- ▶ Identifier les besoins non fonctionnels (~ 80%)
- ▶ Détailler et analyser les cas les plus importants (~ 10%)

↪ Mieux comprendre le système à réaliser

↪ Guider le choix de l'architecture

## Analyse et Conception Orientées Objet

- ▶ Ebaucher la conception architecturale
  - ↪ Architectures logique et de déploiement
- ▶ Examen des aspects importants et à plus haut risque

# Livrables (potentiels) de l'étude préliminaire

- ▶ Liste des besoins potentiels
- ▶ Première version du modèle du domaine / métier
- ▶ Liste ordonnée de cas d'utilisation (description abrégée)
- ▶ Description détaillée de quelques cas
- ▶ Esquisse des architectures logique et de déploiement
- ▶ Liste ordonnée de risques
- ▶ Première estimation (peu fiable) des coûts et délais
- ▶ Planification de la première itération de l'élaboration
- ▶ ...

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

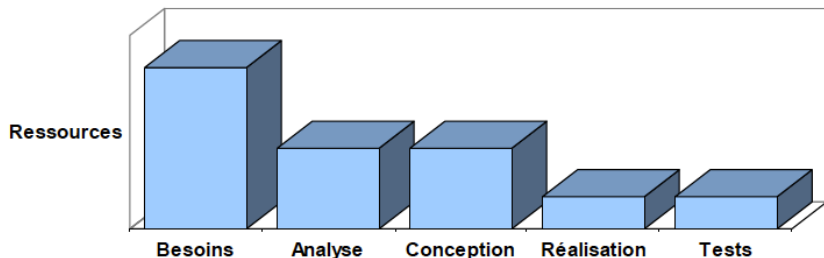
Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

## Phase 2 : Elaboration



### Objectif : Cerner le projet

- ▶ Identifier et stabiliser les besoins
- ▶ Atténuer ou éliminer les risques majeurs
- ▶ Planifier les phases du projet et estimer les coûts
- ▶ Établir les fondations de l'architecture
- ▶ Réaliser un squelette du système

↪ Phase courte : quelques itérations de quelques semaines

[Figure empruntée à J.-L. Sourrouille]

# Activités principales de l'élaboration

## Capture et analyse des besoins

- ▶ Compléter la liste des cas d'utilisation
  - ↪ Description abrégée de tous
  - ↪ Description détaillée / Diag. de séquences de 40 à 80%
- ▶ Faire un prototype de l'interface utilisateur (éventuellement)

## Analyse et Conception Orientées Objet

- ▶ Stabiliser la conception architecturale
  - ↪ Architectures logique et de déploiement
- ▶ Effectuer la conception des cas sélectionnés

## Réalisation et test

- ▶ Limités au squelette de l'architecture
  - ↪ Valider les choix

# Livrables de l'élaboration

- ▶ Modèle du domaine / métier quasi complet
- ▶ Modèle des cas d'utilisation quasi complet
- ▶ Modèles de conception (diagrammes de classes et d'interactions, diagramme de déploiement, ...) partiels
- ▶ Architecture de base exécutable
- ▶ Implémentation de quelques fonctionnalités
- ▶ Document d'architecture du logiciel / modèles en cours
- ▶ Planification des phases
- ▶ Évaluation du coût du projet
- ▶ ...

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

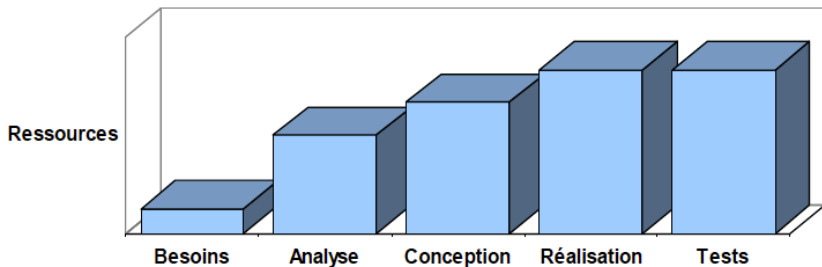
Phase 2 : Elaboration

**Phase 3 : Construction**

Phase 4 : Transition

Synthèse des activités par phase

## Phase 3 : Construction



**Objectif : Réaliser une version bêta**

[Figure empruntée à J.-L. Sourrouille]

# Activités principales de la construction

## Capture et Analyse des besoins

- ▶ Spécifier l'interface utilisateur (cf. cours IHM)
- ▶ Terminer l'analyse de tous les cas d'utilisation

## Analyse et Conception Orientées Objet

- ▶ L'architecture est fixée
- ▶ Concevoir les packages et classes pour réaliser les cas  
    ↪ Selon l'ordre de priorité

## Réalisation et Tests

- ▶ Réaliser et tester les cas, intégrer les incréments

# Livrables de la construction

- ▶ Un exécutable
- ▶ Tous les documents et les modèles du système
- ▶ Document d'architecture du logiciel à jour
- ▶ Un manuel utilisateur suffisamment détaillé pour les tests
- ▶ ...

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

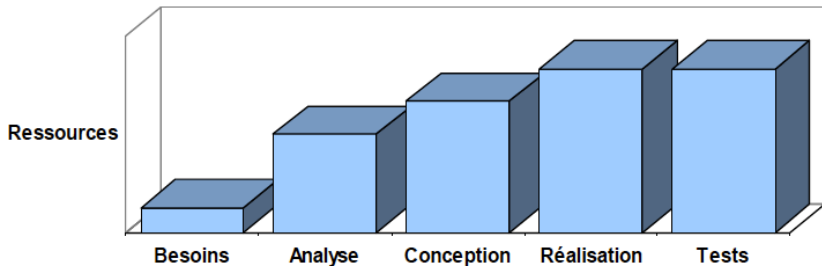
Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

## Phase 4 : Transition



### Objectif : Mise en service chez l'utilisateur

- ▶ Test de la bêta-version, correction des erreurs
- ▶ Préparer la formation, la commercialisation

[Figure empruntée à J.-L. Sourrouille]

# Activités principales de la transition

- ▶ Préparer la version bêta à tester
- ▶ Installer la version sur le site, convertir et faire migrer les données nécessaires...
- ▶ Gérer le retour des sites
  - ↪ Le système fait-il ce qui était attendu ?
  - ↪ Erreurs découvertes ?
  - ↪ ...
- ▶ Adapter le produit corrigé aux contextes utilisateurs (installation...)
- ▶ Terminer les livrables du projet (modèles, documents...)
- ▶ Déterminer la fin du projet
  - ↪ Reporter la correction des erreurs trop importantes
- ▶ ...

↪ Planifier le prochain cycle de développement !?

# Livrables de la transition

- ▶ L'exécutable et son programme d'installation
- ▶ Les documents légaux : contrat, licences, garanties...
- ▶ Un jeu complet de documents de développement à jour
- ▶ Les manuels utilisateur, administrateur et opérateur et le matériel d'enseignement
- ▶ Les références pour le support utilisateur (site Web...)
- ▶ ...

# Plan du cours

## Introduction

Motivations

Quelques rappels (rapides) sur le contexte

## Présentation générale d'USDP

Vue d'ensemble du processus

Les phases d'un cycle

Caractéristiques marquantes de la méthode

## Description détaillée des activités d'une itération générique

Activité "Capture et analyse des besoins"

Activité "Analyse et Conception Orientées Objet"

Activité "Réalisation et Tests"

## Gestion de projet

### Retour sur les phases

Phase 1 : Etude préliminaire (*Inception*)

Phase 2 : Elaboration

Phase 3 : Construction

Phase 4 : Transition

Synthèse des activités par phase

# Capture des besoins : activités par phase

	Étude préliminaire	Élaboration	Construction	Transition
Lister les besoins candidats	□□□□	□		
Modéliser le contexte de l'entreprise ou Modéliser le domaine	□□□□	□		
Identifier les cas d'utilisation (besoins fonctionnels)	□□	□□	□	●
Capter les besoins non fonctionnels	□□	□□	□	●
Faire le glossaire	□	□	□	
Identifier les cas critiques pour l'architecture	□	□□		
Spécification / Prototypage de l'interface utilisateur	□□	□□	□	
Construire le modèle des cas d'utilisation	□□	□□	□	
Structurer le modèle des cas d'utilisation	□	□□	□	
Classer les cas d'utilisation par priorité	□	□	□	
Décrire les scénarios des cas d'utilisation	□	□□□□	□	

□ *Proportion* de travail effectué (et non quantité)

[transparent emprunté à J.-L. Sourrouille]

# Analyse / Conception : activités par phase

		Étude préliminaire	Élaboration	Construction	Transition
Analyse	Identifier les paquetages et leurs dépendances	□	□□□	□	
	Identifier les classes critiques de l'architecture	□	□□		
	Identifier les besoins spéciaux communs		□	□□	
	Déterminer les besoins spéciaux des cas		□	□□	
	Identifier les classes, attributs et relations		□□	□□	
	Détailler les scénarios (interactions entre objets)	□	□□	□□	
	Analyser les classes		□	□□□	
	Analyser les paquetages		□	□□	
Conception	Construire le modèle de déploiement	□	□□□		
	Déterminer les dépendances entre sous-systèmes		□□	□	
	Identifier les classes significatives de l'architecture		□		
	Identifier les mécanismes génériques			□□	
	Détailler les cas (diagrammes d'interactions)		□	□□□	
	Détailler le modèle structurel		□	□□□	
	Détailler les besoins non fonctionnels (réalisation)			□□	
	Etablir un diagramme d'états			□□□	
Classer les sous-systèmes par priorité			□□		

[transparent emprunté à J.-L. Sourrouille]

## Réalisation / Test : activités par phase

		Étude préliminaire	Élaboration	Construction	Transition
Réalisation	Identifier les composants, les associer aux nœuds		□	□□□	
	Réaliser les classes (code des opérations...)		□	□□□	□
	Intégrer le système		□	□□□	□
	Faire les tests unitaires (boîte noire/ blanche)		□	□□□	□
Test	Rédiger un plan de test et concevoir les tests		□	□□□	
	Structurer les procédures de test, automatiser			□□□	□
	Réaliser les tests d'intégration		□	□□□	□
	Réaliser les tests du système		□	□□□	□□
	Evaluer les tests			□□□	□
	Installer sur le site, faire migrer les données...				□□
	Gérer les retours, adapter et corriger le produit				□□

[transparent emprunté à J.-L. Sourrouille]