BCO (M2104) :
Bases de la conception
« orientée objet »

DUT Informatique
Semestre 2

Mourad Ouziri
mourad.ouziri@parisdescartes.fr

IUT de Paris Descartes

Bases de la conception objet
Quelques informations générales

- Contrôles des connaissances
  - 2 DST : 1er DST en mi-semestre et 2ème DST en fin de semestre
  - TP noté en fin de semestre
  - Coefficients : DST1 x 2, DST2 x 2, TP noté x 1
- Bibliographie
  - Object-oriented Modeling and Design. *J. Rumbaugh Prentice Hall, 1991*
  - UML Resource Page. http://www.uml.org
  - UML 2 : Guide de référence. *Booch, Jacobson, Rumbaugh, ampusPress, 2004*
  - UML 2**.** *P. Roques, Eyrolles, 2008*

Bases de la conception objet

Aperçu de la matière

- Objectif : développer des logiciels de qualité
- Introduction à la conception d'applications à objets
- Analyse et conception d'application à objets
  - Le langage UML : cas d'utilisation, classes, séquence, états-transitions
  - Le langage de contraintes OCL – *Object Constraint Language*
- Implémentation des modèles de conception en Java : développement dirigé par les modèles
- Tests et validation : tests dirigés par les modèles

Introduction au

« développement logiciel »

Développement logiciel

Motivation (1) – de la programmation au développement

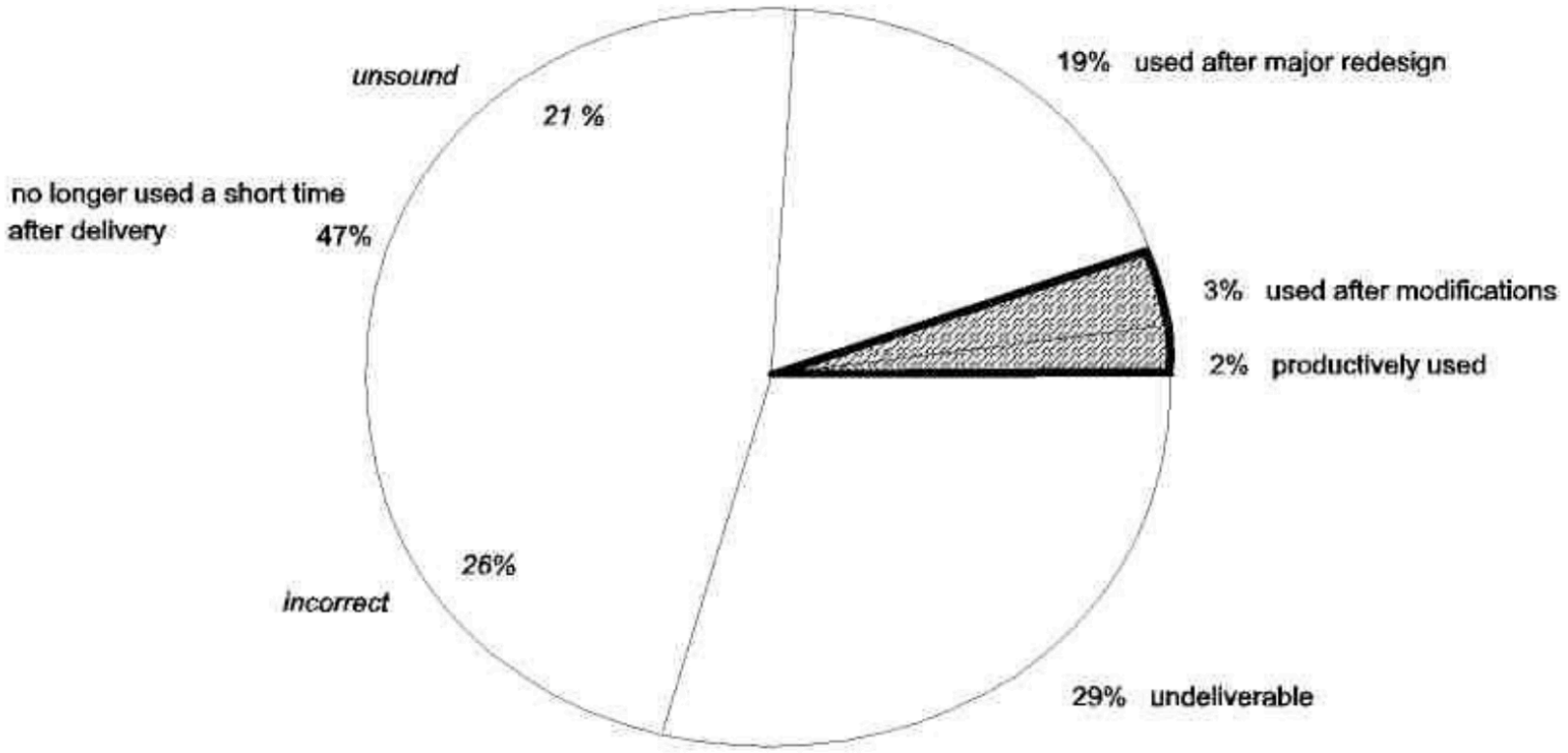
- Développer un petit programme
  - Écrire un algorithme/programme : expression du *COMMENT*
  - Travail à l'échelle de l'individu : *Programming-in-the-small*
- Ne convient pas au développement logiciel !
  - Le *QUOI* n'est pas indiqué : que font les résultats ?
  - Pose problème pour le travail en équipe : faible lisibilité et peu transmissible !
  - Résultats produits qu'à la fin du développement !
  - Maintenance difficile et coûteuse !
- Production du logiciel : *Programming-in-the-large*

Développement logiciel

Motivation (2) – crise de l'industrie du logiciel

THE SOFTWARE CRISIS

© M. Bettoni, 1995



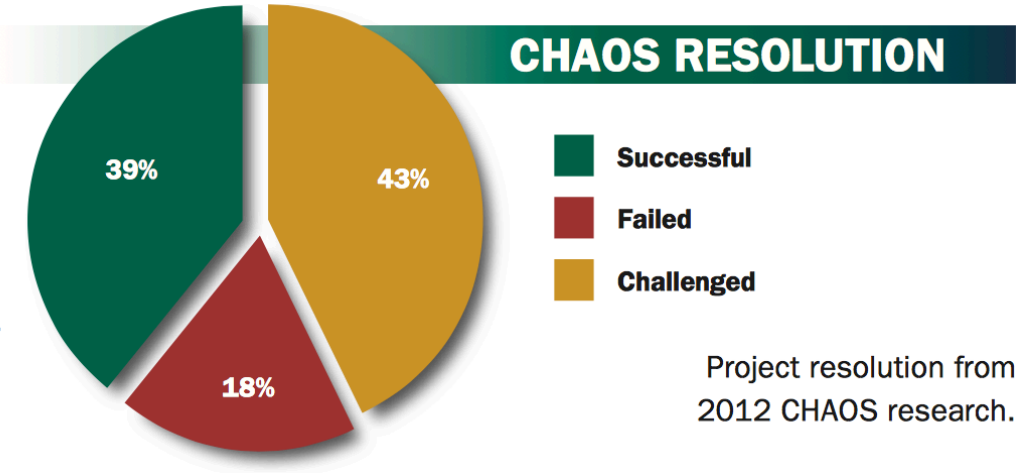
Source: Sage A. & Palmer, J., Software Systems Engineering. John Wiley & Sons, New York, 1990.

Développement logiciel

Motivation (3) – crise de l'industrie du logiciel

☞ Taux d'échec des projets de développement logiciel

The 2012 CHAOS results show another increase in project success rates, with 39% of all projects succeeding (delivered on time, on budget, with required features and functions); 43% were challenged (late, over budget, and/or with less than the required features and functions); and 18% failed (cancelled prior to completion or delivered and never used). These numbers represent an uptick in the success rates from the previous study, as well as a decrease in the number of failures. The low point in the last five study periods was 2004, in which only 29% of the projects were successful. This year's results represent a high watermark for success rates in the history of CHAOS research.



Project resolution from 2012 CHAOS research.

RESOLUTION

	2004	2006	2008	2010	2012
Successful	29%	35%	32%	37%	39%
Failed	18%	19%	24%	21%	18%
Challenged	53%	46%	44%	42%	43%

Project resolution results from CHAOS research for years 2004 to 2012.

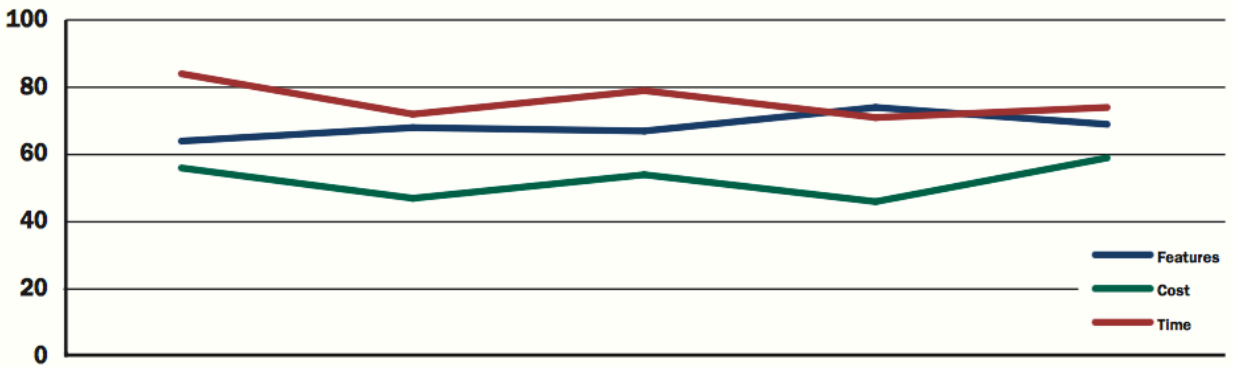
Développement logiciel

Motivation (4) – crise de l'industrie du logiciel

☞ Dépassement de temps et budget et non complétude fonctionnelle

OVERRUNS AND FEATURES

Time and cost overruns, plus percentage of features delivered from CHAOS research for the years 2004 to 2012.



	2004	2006	2008	2010	2012
TIME	84%	72%	79%	71%	74%
COST	56%	47%	54%	46%	59%
FEATURES	64%	68%	67%	74%	69%

Développement logiciel

Motivation (5) – crise de l'industrie du logiciel

- Quelques raisons d'échec des logiciels
  - **Mauvaise compréhension des besoins des utilisateurs**
  - **Découverte tardive des erreurs/problèmes**
  - Logiciel difficile à maintenir ou à faire évoluer
  - Des modules qui ne fonctionnent pas ensemble
  - Problème de coordination dans les équipes
  - Procédures de tests coûteuses
  - …

Développement logiciel
Définitions

- Nécessité de surmonter la crise de l'industrie du logiciel
- Naissance du domaine du Génie Logiciel (1968)
- Génie logiciel – *Software Engineering*

  Le génie logiciel désigne l'ensemble des méthodes, des techniques et outils concourant à la **production de logiciels de qualité**
- Logiciel – *Software*

  Ensemble des programmes, procédés et règles (et éventuellement de la documentation) relatifs au fonctionnement d'un ensemble de traitements de l'information *(IEEE Std 729)*

Développement logiciel
Qualité du logiciel (1)

- Quelques facteurs de qualité *(McCall – US Air Force, FURPS – HP)*
  - **Conformité fonctionnelle** : satisfaire les besoins fonctionnels des utilisateurs
  - **Maintenabilité** : minimiser l'effort pour localiser et corriger les erreurs
  - **Evolutivité** : minimiser l'effort nécessaire pour le modifier par suite d'évolution des spécifications
  - **Maniabilité** : minimiser l'effort nécessaire pour son apprentissage et son utilisation
  - **Portabilité** : facilité de le transférer d'une plateforme/environnement à un autre
  - **Efficacité** : se limiter à l'utilisation des ressources strictement nécessaires à l'accomplissement de ses fonctions
- CISQ – Consortium for IT Software Quality (créé en 2009 aux USA) pour établir un standard mondial de la qualité du logiciel

Développement logiciel
Qualité du logiciel (2)

☞ Que faire pour obtenir cette qualité du logiciel ?

– **Analyser la situation réelle :** étudier la situation et les spécifications fonctionnelles

– **Modéliser :** représenter le résultat d'analyse par des schémas formels (et visuels)

– **Valider avant de programmer :** valider des modèles coûte moins cher que de valider du code

– **Structurer** : définir une architecture de maintenabilité pour le logiciel

– Suivre **une démarche** de développement : définir les étapes à suivre

Développement logiciel
Modélisation

- Modèle
  - Représentation abstraite et facilement compréhensible de la réalité
  - Schématisation de la solution avec des symboles (graphiques) intuitifs
  - Vue subjective mais pertinente de la réalité
- Pourquoi modéliser ?
  - Comprendre les besoins ainsi que le monde réel avant de réaliser le logiciel
  - Concevoir le logiciel indépendamment des langages de programmation objets
  - Faciliter la communication entre les contractants du projet
  - Répartir efficacement les tâches de développement
  - Réduction des coûts et des délais
  - Préparer la documentation

- Quel langage ? UML : langage de modélisation d'applications objets

Développement logiciel

Étapes de développement et diagrammes UML (1)

- Modélisation des besoins fonctionnels des utilisateurs
  - Les modèles exprimant les besoins des utilisateurs sont une description précise de *ce que le client demande*, et non de comment on peut les réaliser
  - Compréhensible par les experts du métier qui ne sont pas informaticiens
  - Ne comporte aucune décision d'implantation
- Diagramme UML correspondant
  - **Diagramme de cas d'utilisation – DCU**

    + Fiches descriptives des CU

1 cours

Développement logiciel

Étapes de développement et diagrammes UML (2)

- Modéliser l'application en *Objet*
  - Identifier les traitements et les données nécessaires au fonctionnement de l'application
  - Analyser le comportement du logiciel face à un besoin
  - Comprendre et tester le comportement des objets constituant le logiciel
- Diagrammes UML correspondants
  - **Diagramme de classes – DC** 3 cours
  - **Diagramme de séquences – DS** 2 cours
  - **Diagramme d'états-transitions – DE** 1 cours

En conclusion

- Modéliser (schématiser) avant de réaliser (programmer)
- Le modèle objet est intuitif
- A noter que la programmation ne représente qu'une infime part dans le développement logiciel !

Le langage UML

Langage de modélisation d'applications à objets

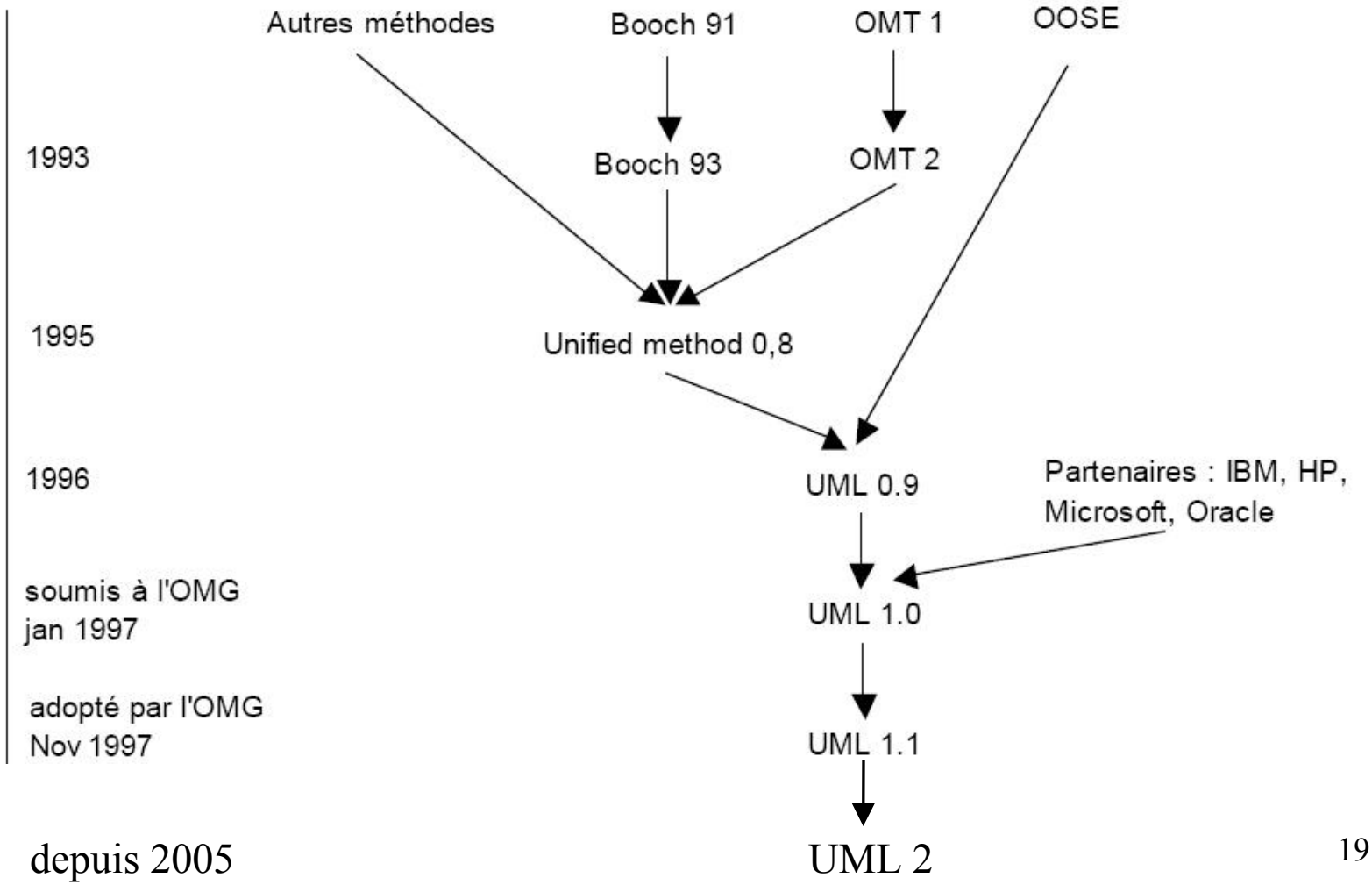
UML

Quelques caractéristiques

- UML : *Unified Modeling Language*
  - Norme de l'OMG (Object Managment Group) depuis 1996
  - Langage de modélisation objet et pas une méthode
  - Universel : indépendant des méthodes et langages de programmation
  - Support de discussion, de communication et de répartition des tâches
  - Intervient à toutes les étapes du développement logiciel : les diagrammes UML
  - Diagramme UML : notation graphique représentant un aspect précis du logiciel

UML

Genèse


Autres méthodes
Booch 91
OMT 1
OOSE
1993
Booch 93
OMT 2
1995
Unified method 0,8
1996
UML 0.9
Partenaires : IBM, HP, Microsoft, Oracle
soumis à l'OMG jan 1997
UML 1.0
adopté par l'OMG Nov 1997
UML 1.1
depuis 2005
UML 2

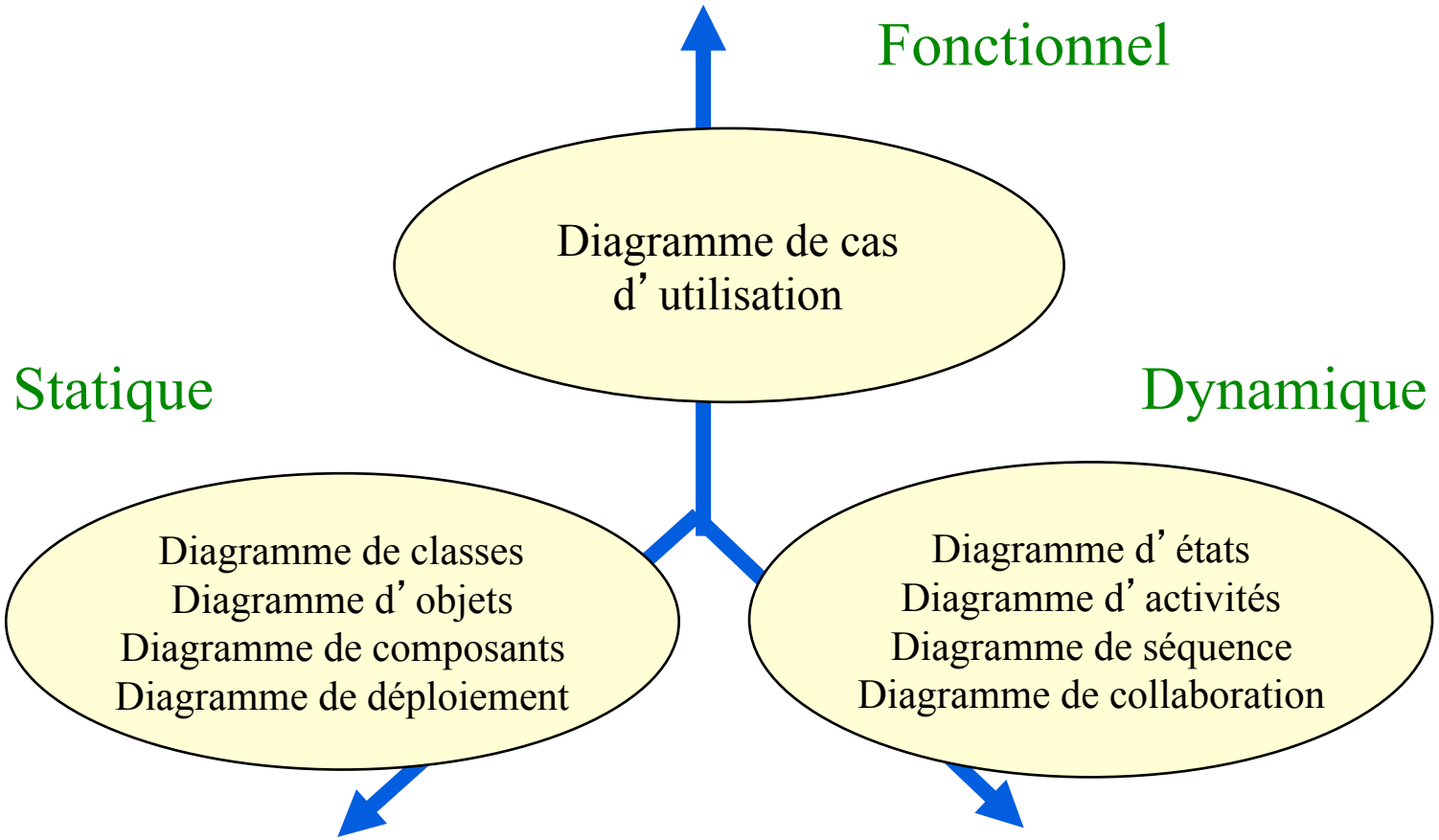
UML
Objectif

- Objectifs de UML
  - Bien formaliser les besoins utilisateurs et montrer les frontières du projet de développement logiciel
  - Illustrer les réalisations (déroulements) des fonctions principales du logiciel
  - Représenter la structure statique (squelette) du logiciel « orienté objet »
  - Modéliser la dynamique et le comportement des objets
  - Révéler l'implantation physique de l'architecture
- Un langage compréhensible par l'homme et la machine
  - Conception (intellectuelle) humaine et manipulation par la machine
  - Atelier de génie logiciel – AGL
  - Ce qui permet la génération automatique du code

UML

Les diagrammes UML

☞ UML : les 9 diagrammes



UML
Quelques diagrammes

- UML : les quatre diagrammes que nous étudierons
  - Diagrammes des cas d'utilisation (DCU) : fonctions de l'application du point de vue des utilisateurs
  - Diagrammes de classes (DC) : structure statique d'une application objet
  - Diagrammes de séquence (DS) : modèle dynamique représentant les interactions entre les objets d'une application
  - Diagrammes d'états-transitions (DE) : comportement des objets d'une classe en terme d'états et de transitions valides

Modélisation fonctionnelle :

« Diagramme de Cas d'Utilisation UML »

UML

DCU – Diagramme de Cas d'Utilisation

- Diagramme de cas d'utilisation
  - But : recueillir, comprendre et structurer les besoins utilisateurs
  - DCU : Représentation graphique, simple et compréhensible des besoins utilisateurs
  - Définir les frontières du système à modéliser (préciser le but à atteindre)
  - Identifier les besoins fonctionnels requis par les utilisateurs potentiels du logiciel
  - Fournir un document de discussion entre la maîtrise d'ouvrage et la maîtrise d'œuvre

UML

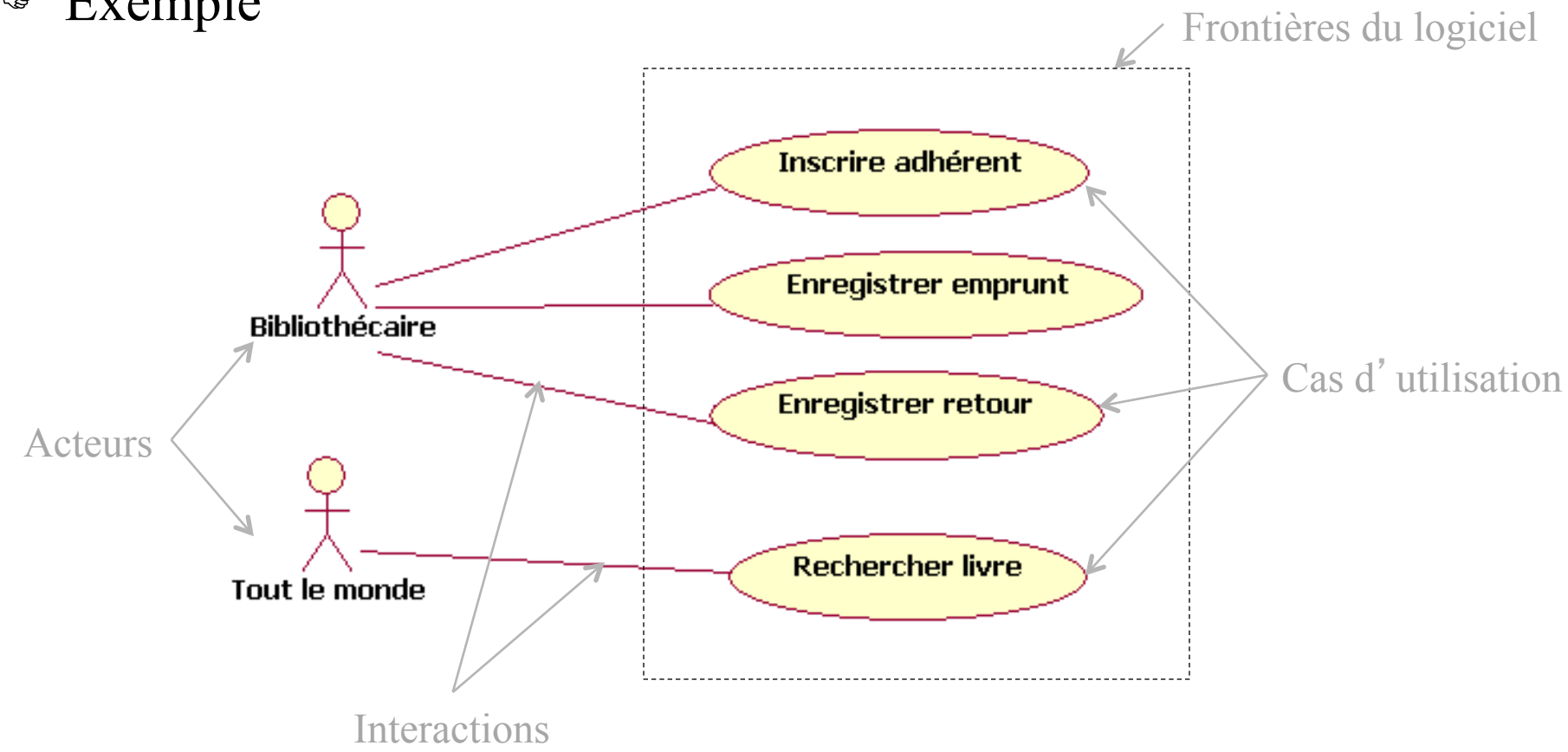
DCU – Diagramme de Cas d'Utilisation

- DCU : **Qui** utilisera le logiciel et **Quels** sont ses besoins fonctionnels ?
  - **Qui** : utilisateurs du logiciel analysé ayant une interaction direct avec lui
  - **Quoi** : fonctions du logiciel analysé
- Éléments de modélisation du diagramme de cas d'utilisation
  - **Acteur** (Qui?) : utilisateur du logiciel
  - **Cas d'utilisation** (Quoi?) : fonctionnalité utile (requise) pour au moins un acteur

UML

DCU – Diagramme de Cas d'Utilisation

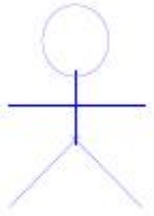
☞ Exemple



UML

DCU – Diagramme de Cas d'Utilisation

- Acteur
  - Toute entité externe au logiciel et qui interagit directement avec lui (qui l'utilise)
  - Représente un rôle joué par plusieurs personnes (ou systèmes)
  - La même personne/entité physique peut être représentée par plusieurs acteurs en fonction des rôles qu'elle joue
  - Un acteur n'est pas nécessairement une personne physique : il peut être un service administratif, une société, un système informatique, …
- Représentation de l'acteur

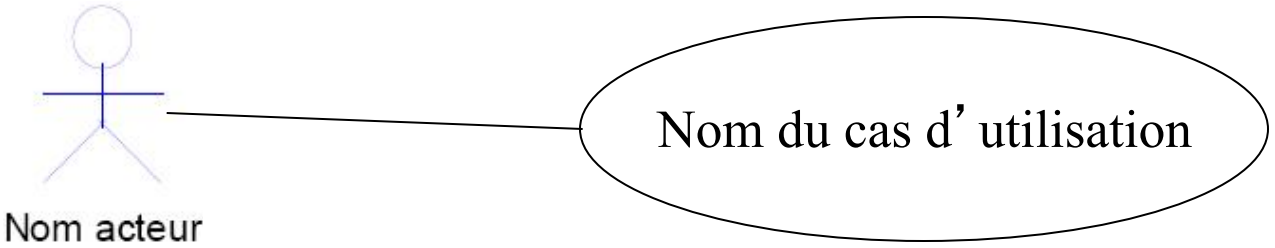


Nom acteur

UML

DCU – Diagramme de Cas d'Utilisation

- Cas d'utilisation
  - Comportement du logiciel du point de vue des Acteurs
  - Fonction métier (besoin fonctionnel) requise par au moins un acteur
  - Une fonction métier déclenchée par un acteur (directement ou indirectement)
  - Il modélise à la fois des activités (fonctions) et des communications
- Représentation d'un cas d'utilisation



UML

DCU – Diagramme de Cas d'Utilisation

- Deux catégories d'acteurs pour un CU
  - Acteur principal : toute entité qui utilisent directement un cas d'utilisation
  - Acteur secondaire : une entité qui bénéficie d'un résultat produit par le CU ou celle sollicitée par le logiciel
- Exemple
  - Bibliothécaire : voudrait enregistrer les emprunts pour les adhérents
  - Adhérent : informé de la date limite de remise

UML

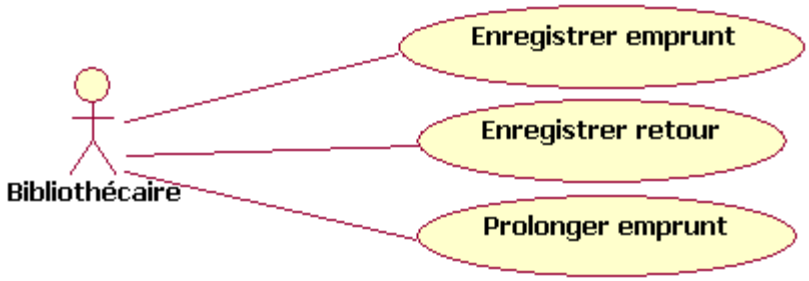
DCU – Identifier les acteurs

- Qui utilisera directement les fonctionnalités du logiciel ?
- Qui a besoin du support du logiciel pour effectuer ses tâches quotidiennes ?
- Avec quels autres systèmes le logiciel interagit-il ?
- Qui a un intérêt pour les résultats produits ?
- …

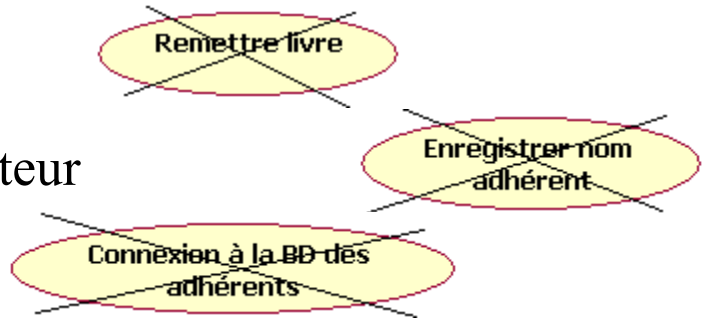
UML

DCU – Cas d'utilisation

- Rappel : un modèle (DCU) est un schéma subjectif !
- Les CU peuvent être conçus à différents niveaux de détail !



- Quelques caractéristiques des CU:
  - Informatisable
  - Une finalité pour l'utilisateur
  - Service/fonction métier



DCU

Fiches descriptives de Cas d'Utilisation

UML

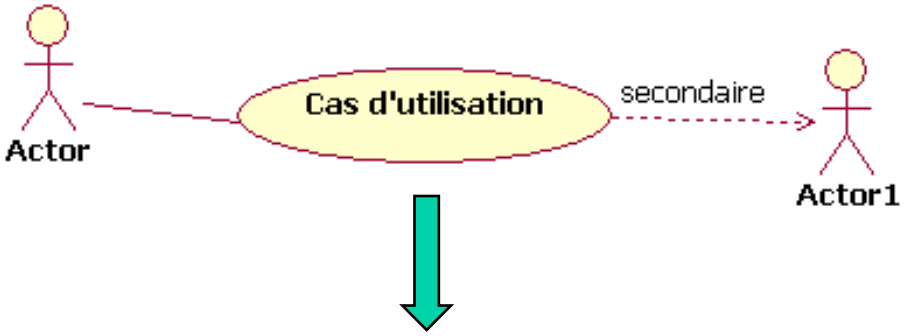
DCU – Fiche descriptive des CU

- Limites du DCU
  - Le DCU n'est qu'un sommaire des besoins fonctionnels
  - Nom des CU peu explicatif de la fonction ➔ fonctions pas assez précises
- D'où la nécessité de détailler les CU par une description textuelle
- Objectifs des fiches descriptives des CU
  - Décrire comment se déroule le CU/fonction sur le plan métier
  - Donner les scénarios principaux des CU
  - Définir les informations échangées entre l'application et les utilisateurs
  - Préparer les cas de tests

UML

DCU – Fiche descriptive des CU

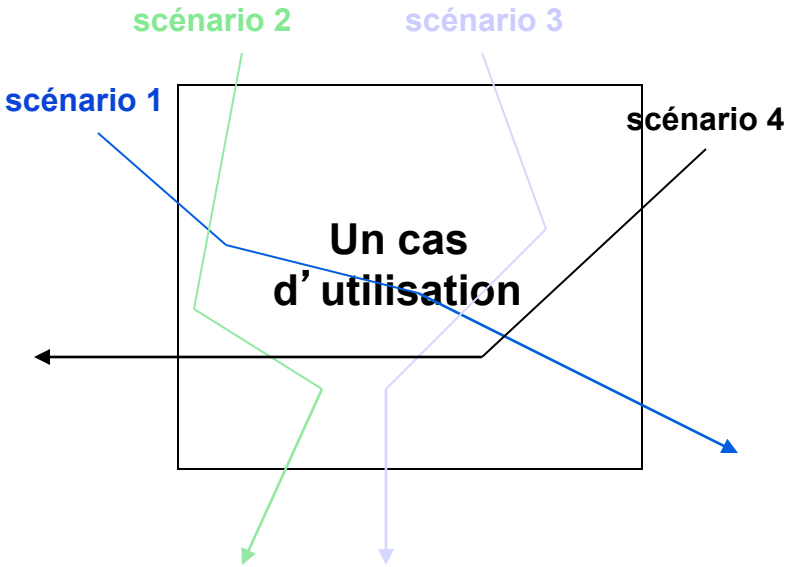
☞ Contenu d'une fiche descriptive d'un CU



Pré-conditions

Scénarios

Post-conditions



UML

DCU – Fiche descriptive des CU

☞ Contenu d'une fiche descriptive d'un CU

1. Identification du CU

Nom du CU, date de création, résumé , version, acteurs impliqués

2. Séquences : scénarios possibles du CU

2.1 Pré-conditions : conditions devant être satisfaites pour que le CU puisse se dérouler

2.2 **Enchaînements** : scénarios du CU

Nominal (obligatoire) : scénario le plus probable (favorable)

Alternatifs (optionnel) : scénarios alternatifs au scénario nominal

Exceptionnels (optionnel) : traitement des exceptions (erreurs)

2.3 Post-conditions : conditions devant être satisfaites à la fin du CU et ce quelque soit le scénario

UML

Fiche descriptive – Exemple

☞ Fiche descriptive du CU « *Valider la création d'un compte bancaire*»

Nom du cas : Valider la création d'un compte bancaire

Acteur principal : Directeur d'agence

Date : 15/01/2014 **Version** : 1.0

Pré-conditions : le directeur est authentifié

Enchaînement nominal

Directeur d'agence	CU – Valider des comptes
1. Demander à valider des comptes	2. Afficher la liste des comptes en attente de validation
3. Choisir un compte	4. Afficher les informations du compte choisi (propriétaire, justif. de ressources, date de création)
5. Valider le compte	6. Enregistrer la validation + date de validation
	7. Retour au point 2

Post-conditions : Les comptes validés (ou annulés) auparavant ne changent pas d'état

UML

Fiche descriptive – Exemple

Enchaînement alternatif 1 – Non validation d'un compte

Le cas continue au point 5

5. Annuler la création du compte et saisir le motif

6. Enregistrer la non validation du compte et le motif

Enchaînement alternatif 2 – Aucun compte en attente de validation

Le cas continue au point 2

2. Afficher le message « Pas de comptes à valider »

Enchaînement d'exception – Motif de non validation de compte non renseigné

Le cas continue au point 6 du cas alternatif 1

6. Afficher le message « Motif de non validation obligatoire ! »

7. Retour au point 4

UML

Fiche descriptive – Exemple

☞ Un deuxième scénario peu pertinent !

Directeur d'agence	CU – Valider des comptes
1. Demander à valider des comptes	2. Demander à saisir le numéro du compte à valider
3. Saisir un numéro de compte	4. Afficher les informations du compte (propriétaire, justif. de ressources, date de création)
5. Valider le compte	6. Enregistrer la validation + date de validation
	7. Retour au point 2

☞ Scénario alternatif : « Le compte saisi n'existe pas ou a déjà été validé »

4. Afficher le message « Compte inconnu ou déjà validé ! »

5. Retour au point 2

UML

Fiche descriptive – Exemple

☞ Un scénario peu pertinent !

Directeur d'agence	CU – Valider des comptes
1. Demander à valider des comptes	2. Afficher la liste des (tous !) comptes bancaires
3. Choisir un compte	4. Afficher les informations du compte (propriétaire, justif. de ressources, date de création)
5. Valider le compte	6. Enregistrer la validation + date de validation
	7. Retour au point 2

☞ Scénario alternatif : « Le compte choisi a déjà été validé »

4. Afficher le message « Compte déjà validé ! »

5. Retour au point 2

UML

DCU – Fiche descriptive des CU

- Avantages
  - Décrire les CU : meilleure compréhension des fonctions du logiciel
  - Faciliter la conception des diagrammes UML (notamment les diagrammes de séquence et de collaboration)
  - Préparer les cas de tests
  - Préparer la documentation finale du logiciel

DCU

Relations entre Cas d'Utilisation

UML

DCU – Relations dans le DCU

- Modulariser pour mieux **maintenir**
- Modulariser pour **réutiliser**
  - Constat : comportements (traitements) communs à plusieurs CU
  - Conséquence : redondance du travail de développement
  - Modulariser pour éviter les développements redondants
- Types de relations dans un DCU
  - 3 types de relation en CU : Inclusion, Extension et Héritage
  - 1 type de relation entre acteurs : Héritage

UML

DCU – Relations dans le DCU

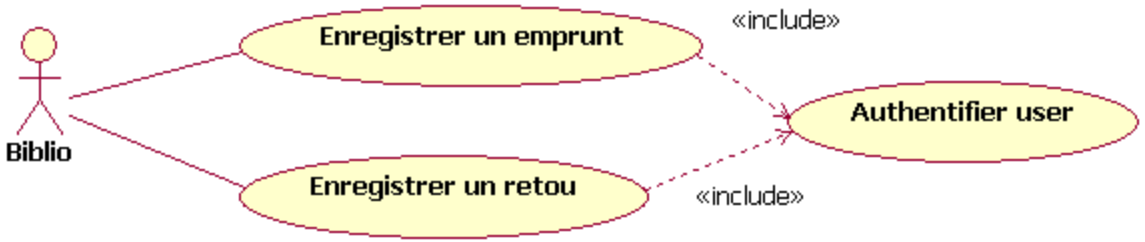
☞ **Inclusion de CU**

– Sémantique : CU1 *inclut* CU2 ssi CU1 fait appel à CU2 dans tous les scénarios possibles

– CU1 ne peut se terminer avec succès si CU2 ne s'est pas terminé avec succès !

– Représentation graphique

– Exemple : la bibliothécaire doit être authentifiée pour enregistrer un emprunt ou un retour



UML

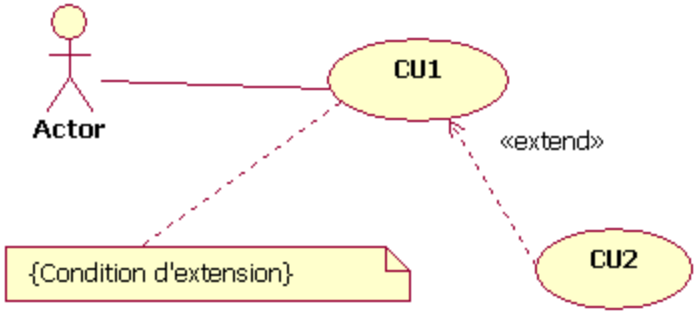
DCU – Relations dans le DCU

☞ **Extension de CU**

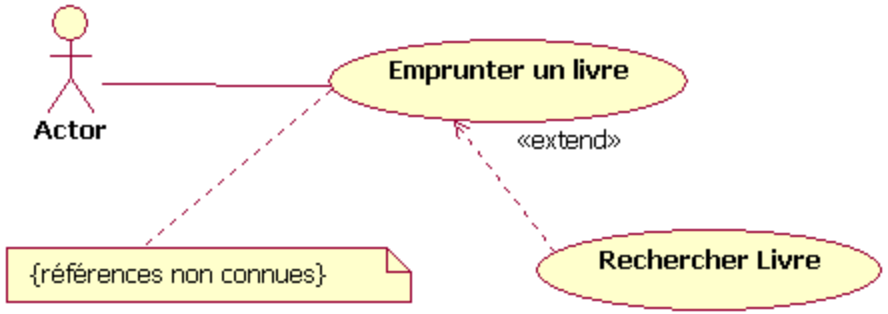
– Sémantique : CU1 *étend* CU2 ssi CU1 fait appel à CU2 dans certains scénarios

– L'extension est conditionnelle

– Représentation graphique



– Exemple : lorsqu'on emprunte un livre, on peut le rechercher si on ne connait pas ses références exactes



UML

DCU – Relations dans le DCU

☞ **Héritage entre CU**

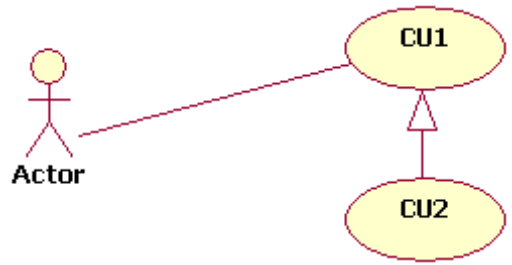
– Sémantique : CU2 *hérite de* CU1 signifie que :

CU2 est une spécialisation de CU1 (CU1 et une généralisation de CU2)

CU1 spécialise CU2 en y rajoutant des spécificités

– Traitements de même nature et substituables

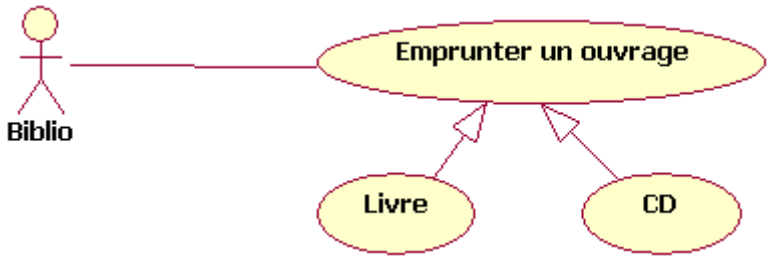
– Représentation graphique



– Exemple

On peut emprunter soit

des livres soit des CD



UML

DCU – Relations dans le DCU

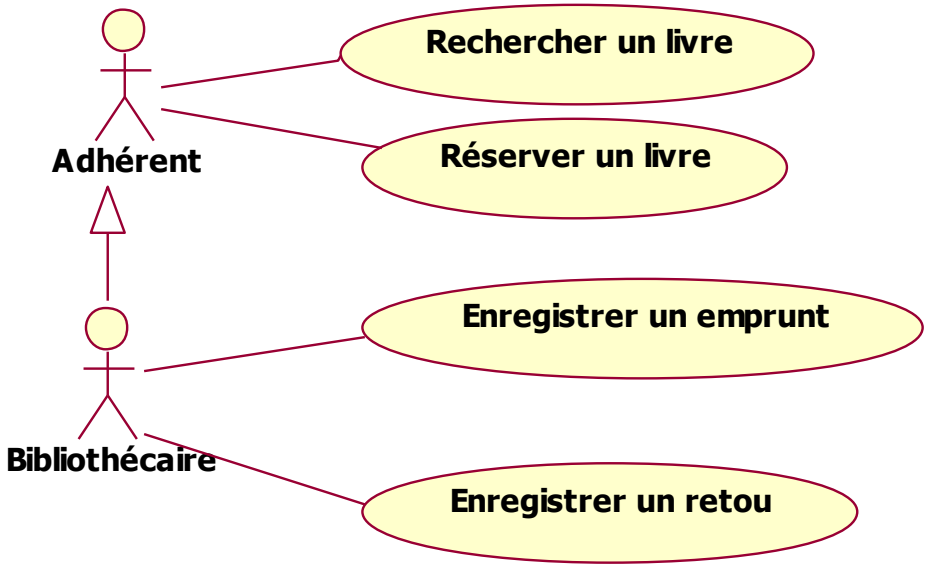
☞ **Héritage entre acteurs**

– Sémantique :

Acteur1 *hérite de* Acteur2 si Acteur1 utilise tous les CU de Acteur2

– Exemple

La bibliothécaire utilise (a besoin de) tous les CU de l'adhérent



UML

DCU – Résumé

- Le DCU exprime les attentes des utilisateurs
  - Fonctions métiers exprimées du point de vue de l'utilisateur
- Il doit être lisible
  - C'est un support de discussion permettant de valider les besoins fonctionnels
- Il peut inclure des relations entre CUs
  - Ne pas trop détailler les CU en voulant utiliser systématiquement ces relations
  - Attention ! Ne pas confondre les relations d'héritage et d'extension
- Le DCU est indépendant du modèle *Objet*
  - Peut être utilisé dans n'importe quel processus de développement