Cours 04 - C++ pour les mathématiques appliquées

Analyse et Conception Objet

Comment bien concevoir un logiciel avant de coder
La dernière fois…

Ce que nous avons vu :

- Constructeur(s) et destructeur
- Surcharge d'opérateurs
- Règle de 3 et de 5

Points clés à retenir :

- Différence entre opérateur `=` et constructeur
- Quand sont appelés les destructeurs
- `const` virologie (propagation du const)

… aujourd’hui

Programme de la séance

Un **changement de perspective** :

Avant (Cours 1-3)

- Syntaxe C++
- Classes et objets
- Opérateurs
- Code, code, code !

Maintenant (Cours 4)

- Analyse des besoins
- Conception avant codage
- Diagrammes UML
- Communication

Pourquoi ? Parce qu'un bon logiciel commence par une bonne conception, pas par du code !

Introduction

Le problème du développement logiciel

Constat dans l'industrie

Statistiques alarmantes

- 30% des projets sont abandonnés
- 50% dépassent budget et délais
- 70% ne respectent pas le cahier des charges

Coûts

- Corrections de bugs : 40-80% du budget
- Maintenance : 60-90% du coût total
- Retard moyen : 6-12 mois

Causes principales

1. **Besoins mal compris** (30%)
2. Architecture inadaptée (25%)
3. Code non maintenable (20%)
4. Communication défaillante (15%)
5. Autres (10%)

1/3 des problèmes viennent de besoins mal exprimés ou mal compris !

Exemple d’échec : le cas classique

Scénario réaliste

Semaine 1 : Le client dit "Je veux un système de gestion"

Semaine 2 : Le développeur code une base de données

Semaine 4 : Le client : "Mais je voulais aussi des statistiques !"

Semaine 6 : Refonte complète, ajout de graphiques

Semaine 8 : Le client : "Ah et il faut gérer plusieurs utilisateurs"

Semaine 10 : Architecture incompatible, tout recommencer

Semaine 16 : Projet abandonné ou très en retard

Problème : On a codé avant de comprendre et concevoir.
Solution : Analyse et Conception Objet de Logiciels (ACOL)

La solution : ACOL

Analyse et Conception Objet de Logiciels

Phase d'ANALYSE

- Comprendre les besoins
- Identifier les acteurs
- Définir les fonctionnalités
- Créer un glossaire
- Lever les ambiguïtés

"QUOI faire ?"

Phase de CONCEPTION

- Définir l'architecture
- Choisir les patterns
- Concevoir les classes
- Définir les interfaces
- Optimiser la structure

"COMMENT le faire ?"

Phase d'IMPLÉMENTATION

- Écrire le code
- Tests unitaires
- Intégration
- Documentation
- Déploiement

"Réaliser"

Ce cours se concentre sur l'**ANALYSE** avec UML

UML - Unified Modeling Language

Un langage visuel pour modéliser

Qu’est-ce que UML ?

- Langage de modélisation **standard**
- Diagrammes **visuels**
- Compréhensible par tous (clients, développeurs)
- Utilisé dans **l’industrie**

Avantages

- Communication claire
- Documentation vivante
- Détection précoce d’erreurs
- Support pour l’implémentation

Types de diagrammes UML

Statiques (structure)

- Diagramme de classes
- Diagramme d’objets
- Diagramme de composants

Dynamiques (comportement)

- Cas d’utilisation
- Séquence
- États-transitions
- Activités

Exemple pédagogique : Bibliothèque universitaire

Notre fil conducteur pour ce cours

Contexte

- Bibliothèque d'une université
- Gère livres, revues, thèses
- Emprunt par étudiants et enseignants
- Système informatisé

Pourquoi cet exemple ?

- Simple à comprendre
- Tout le monde connaît
- Acteurs évidents
- Cas d'usage clairs

Besoins de base

- Emprunter un document
- Retourner un document
- Rechercher un document
- Gérer le catalogue
- Gérer les utilisateurs
- Calculer les pénalités de retard

Contraintes

- Nombre max d'emprunts par personne
- Durée d'emprunt limitée
- Certains documents non empruntables

Phase d’analyse

Comprendre avant de concevoir

Phase d’analyse - Objectifs

Que cherche-t-on à accomplir ?

Comprendre

- Les besoins du client
- Le domaine d’application
- L’environnement du système
- Les contraintes

Clarifier

- Lever les ambiguïtés
- Définir les termes métier
- Identifier les limites du système
- Prioriser les fonctionnalités

Produire

- Cahier des charges précis
- Glossaire des termes
- Diagrammes de cas d’utilisation
- Diagrammes de séquence
- Ébauche du modèle objet

Valider

- Avec le client
- Avec les utilisateurs
- Avec l’équipe technique

Expression des besoins

Problèmes du langage naturel

Cahier des charges initial

> "Le système doit permettre de gérer les livres. Les utilisateurs peuvent emprunter des ouvrages. Il faut aussi gérer les retards."

Problèmes :

- "Gérer" : que signifie exactement ?
- "Utilisateurs" : qui ? étudiants ? personnel ?
- "Emprunter" : combien ? combien de temps ?
- "Retards" : quelle pénalité ?

Après analyse

- **Gérer** = ajouter, supprimer, modifier, rechercher
- **Utilisateurs** = étudiants (3 livres max) + enseignants (5 livres max)
- **Durée** = 2 semaines (étudiants), 4 semaines (enseignants)
- **Retards** = 0.50€/jour pour tous

Glossaire créé**Ambiguïtés levées**

Types de besoins

Besoins fonctionnels vs non-fonctionnels

Fonctionnels

- Ce que le système **fait**
- Emprunter un livre
- Rechercher un document
- Calculer une pénalité

Non-fonctionnels

- **Comment** le système le fait
- Performance (< 2s)
- Fiabilité (99.9%)
- Sécurité (auth)
- Utilisabilité (ergonomie)

Piège classique : Se concentrer uniquement sur les besoins fonctionnels et négliger les non-fonctionnels.

Diagramme des cas d’utilisation

Qui fait quoi ?

Cas d’utilisation - Définition

Concepts de base

Acteur

- Quelqu’un ou quelque chose qui **interagit** avec le système
- Externe au système
- Peut être :
  - Une personne (Étudiant, Bibliothécaire)
  - Un système externe (Base de données)
  - Un matériel (Scanner de code-barres)

Types d’acteurs

- **Principal** : utilise les fonctionnalités principales
- **Secondaire** : administration, maintenance

Cas d’utilisation

- Une **fonctionnalité** du système
- Point de vue de l’acteur
- A une **valeur métier**
- Peut réussir ou échouer

Caractéristiques

- Nom : verbe à l’infinitif
- Description textuelle
- Acteur(s) impliqué(s)
- Pré-conditions
- Post-conditions

Exemple : Bibliothèque - Acteurs

Identification des acteurs

Acteurs principaux

Étudiant

- Emprunte des livres
- Consulte le catalogue
- Réserve des documents

Enseignant

- Mêmes droits qu'étudiant
- Limites différentes
- Peut suggérer acquisitions

Acteurs secondaires

Bibliothécaire

- Gère le catalogue
- Gère les emprunts/retours
- Paramètre le système

Administrateur

- Gère les utilisateurs
- Statistiques
- Sauvegarde

Systèmes externes

Système universitaire

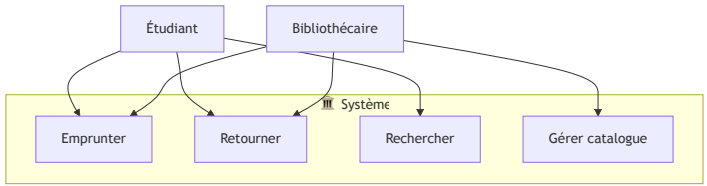
- Authentification
- Données étudiants

Fournisseurs

- Catalogues
- Commandes

Diagramme de cas d’utilisation - Bibliothèque

Vue d’ensemble



Acteurs principaux

- **Étudiant** : emprunte, retourne, recherche
- **Enseignant** : mêmes droits qu’étudiant
- **Bibliothécaire** : gère le catalogue et les emprunts

Cas d’utilisation

- Emprunter document
- Retourner document
- Rechercher document
- Gérer catalogue

Description d’un cas d’utilisation

Cas d’utilisation : Emprunter un document

Informations générales

- **Nom** : Emprunter un document
- **Acteur principal** : Étudiant / Enseignant
- **Acteur secondaire** : Système universitaire
- **Objectif** : Obtenir un document pour une période déterminée

Pré-conditions

- L’utilisateur est authentifié
- L’utilisateur n’a pas dépassé son quota
- Le document est disponible
- L’utilisateur n’a pas de retard > 30 jours

Scénario nominal

1. L’utilisateur présente sa carte
2. Le système vérifie l’identité
3. L’utilisateur scanne le code-barres du document
4. Le système vérifie la disponibilité
5. Le système enregistre l’emprunt
6. Le système affiche la date de retour
7. Le système imprime un reçu

Post-conditions

- L’emprunt est enregistré
- Le document est marqué "emprunté"
- Le quota de l’utilisateur est décrémenté

Description structurée

Exemple : structuration avec numérotation

Scénario principal

1. Présenter carte
2. Scanner carte
3. Vérifier profil
4. Scanner document
5. Enregistrer emprunt
6. Imprimer reçu

Avantages

- Clair et séquentiel
- Facile à tester
- Base pour les diagrammes

Variantes et exceptions

```
3. Vérifier profil
   3a. Si retards → bloquer
   3b. Si quota atteint → refuser

4. Scanner document
   4a. Si indisponible → proposer réservation
   4b. Si non empruntable → refuser
```

Principe : variantes = lettres (a, b, c…)

Relations entre cas d'utilisation

Structuration avec `<<include>>` et `<<extend>>`

`<<include>>` : inclusion obligatoire

```
Emprunter document
    <<include>> Authentifier utilisateur
    <<include>> Vérifier disponibilité
    <<include>> Enregistrer transaction
```

Chaque emprunt **doit** passer par ces étapes.

`<<extend>>` : extension optionnelle

```
Emprunter document
    <<extend>> Réserver si indisponible
    <<extend>> Suggérer documents similaires
```

Ces fonctionnalités sont **optionnelles**.

Diagrammes de séquence

Quand et dans quel ordre ?

Diagramme de séquence - Principe

Montrer les interactions dans le temps

Objectif

- Illustrer un **scénario** d'un cas d'utilisation
- Montrer l'**ordre chronologique** des interactions
- Identifier les **messages** échangés

Éléments

- **Acteurs** : humains ou systèmes
- **Ligne de vie** : existence dans le temps
- **Messages** : communications
- **Activation** : traitement en cours

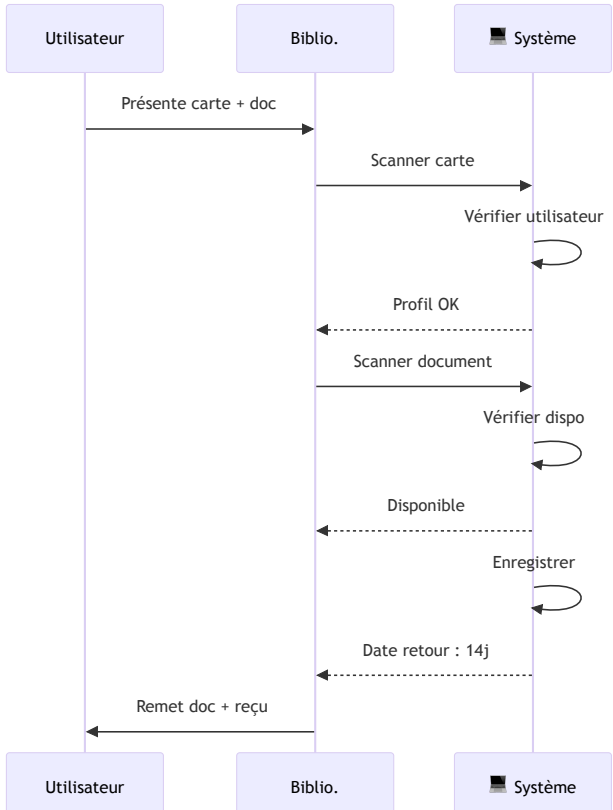
Notation

- Rectangle : acteur ou objet
- Ligne verticale pointillée : ligne de vie
- Flèche pleine → : appel synchrone
- Flèche pointillée ← : retour
- Boîte sur ligne de vie : activation

Utilisé pour

- Documenter un scénario précis
- Valider la logique avec le client
- Identifier les services nécessaires

Exemple : Emprunter un document (succès)



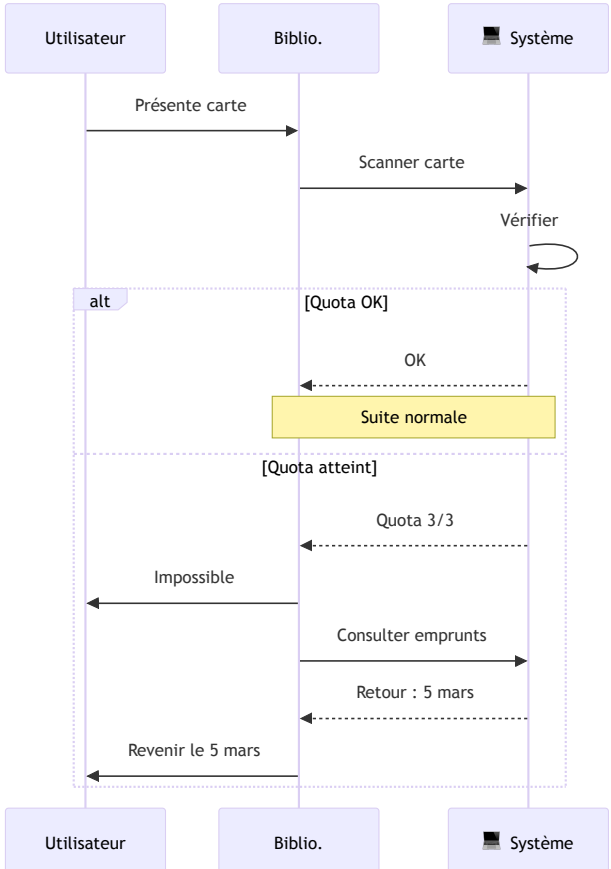
Scénario nominal

1. L'utilisateur présente sa carte et le document
2. Le système vérifie l'identité
3. Le système vérifie le quota
4. Le système vérifie la disponibilité
5. Le système enregistre l'emprunt
6. Le système calcule la date de retour
7. Remise du document et du reçu

Points clés

- Vérifications en cascade
- Base de données cachée dans "Système"
- Durée : 14 jours (étudiant)

Exemple : Emprunter un document (quota dépassé)



Scénario alternatif

Branche : Quota OK

- Suite du scénario nominal

Branche : Quota dépassé

- Message d'erreur
- Consultation des emprunts en cours
- Information sur la prochaine date de retour
- Refus de l'emprunt

Fragment `alt`

- Alternative entre 2 branches
- Utilisé pour les conditions
- `else` = branche par défaut

Fragments dans les diagrammes de séquence

Structures de contrôle

`alt` **: Alternative**

```
alt Si condition A
    Actions A
else Si condition B
    Actions B
else
    Actions par défaut
end
```

`opt` **: Optionnel**

```
opt Si condition
    Actions optionnelles
end
```

`loop` **: Boucle**

```
loop Pour chaque document
    Scanner document
    Enregistrer
end
```

`par` **: Parallèle**

```
par Actions simultanées
    Mise à jour BD
and
    Envoi email
end
```

Diagrammes états-transitions

Quels états possibles ?

Diagramme états-transitions - Principe

Modéliser le cycle de vie d'un objet

Objectif

- Montrer les **états** possibles d'un objet
- Montrer les **transitions** entre états
- Identifier les **événements** déclencheurs

Éléments

- **État** : situation stable
- **Transition** : passage d'un état à un autre
- **Événement** : cause de la transition
- **Action** : ce qui se passe pendant la transition

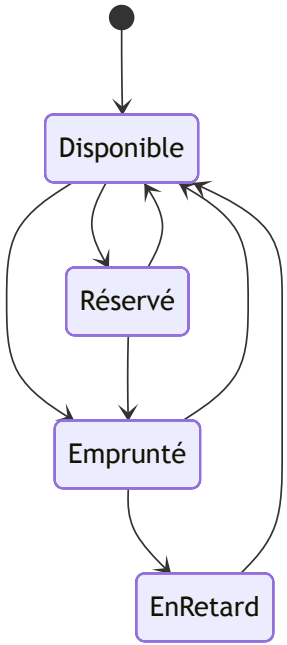
Utilisé pour

- Objets avec comportement complexe
- Protocoles de communication
- Interfaces utilisateur
- Workflow métier

Exemples typiques

- Document (Brouillon → Validé → Archivé)
- Commande (Créée → Payée → Expédiée)
- Connexion (Déconnecté → Connecté)

Exemple : États d’un document



Cycle de vie

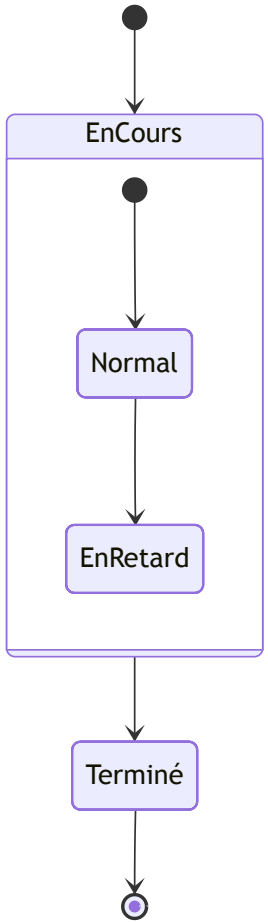
États possibles

- **Disponible** : peut être emprunté
- **Emprunté** : en cours d’emprunt
- **EnRetard** : délai dépassé
- **Réservé** : réservé par un utilisateur

Transitions principales

- Emprunter → Emprunté
- Retourner → Disponible
- Délai dépassé → EnRetard
- Retourner + pénalité → Disponible

Exemple : États d'un emprunt



Cycle de vie

États composites

- **EnCours** contient :
  - Normal (dans les délais)
  - EnRetard (délai dépassé)

Évolution temporelle

- J-3 : rappel email automatique
- J : passage en EnRetard
- Retour : passage en Terminé
- Calcul pénalité si retard

Actions automatiques

- Email de rappel
- Calcul pénalité quotidien
- Blocage compte si > 30j

Diagramme de classes d’analyse

Quels objets du domaine ?

Diagramme de classes d'analyse

Modèle objet du domaine métier

Objectif

- Identifier les **concepts** du domaine
- Modéliser les **relations** entre concepts
- Définir les **attributs** essentiels

Important

- C'est un modèle **conceptuel**, pas technique
- Pas encore de code
- Pas de méthodes (ou très peu)
- Terminologie du domaine métier

Technique d'identification

1. Analyser le texte

- **Noms** → classes candidates
- **Adjectifs** → attributs
- **Verbes** → relations ou opérations

2. Filtrer

- Éliminer les synonymes
- Regrouper les concepts similaires
- Garder ce qui a un rôle métier

Exemple : Bibliothèque - Classes principales

Classes conceptuelles identifiées

Personnes

- Utilisateur
  - Étudiant
  - Enseignant
- Bibliothécaire
- Auteur

Documents

- Document (abstrait)
  - Livre
  - Revue
  - Thèse
  - DVD
- Exemplaire

Gestion

- Emprunt
- Réservation
- Pénalité
- Catalogue

Justification

- **Utilisateur** : acteur principal, a des attributs métier (quota, retards)
- **Document** : objet métier central, plusieurs types
- **Exemplaire** : un livre peut avoir plusieurs copies physiques
- **Emprunt** : transaction métier, a un cycle de vie
- **Pénalité** : règle métier importante

Relations entre classes

Types de relations

Association

- Relation simple entre classes
- Exemple : Utilisateur – Emprunt

Agrégation (◇)

- "A un" (existence indépendante)
- Exemple : Bibliothèque ◇– Document

Composition (◆)

- "Est composé de" (existence dépendante)
- Exemple : Document ◆– Page

Héritage (→)

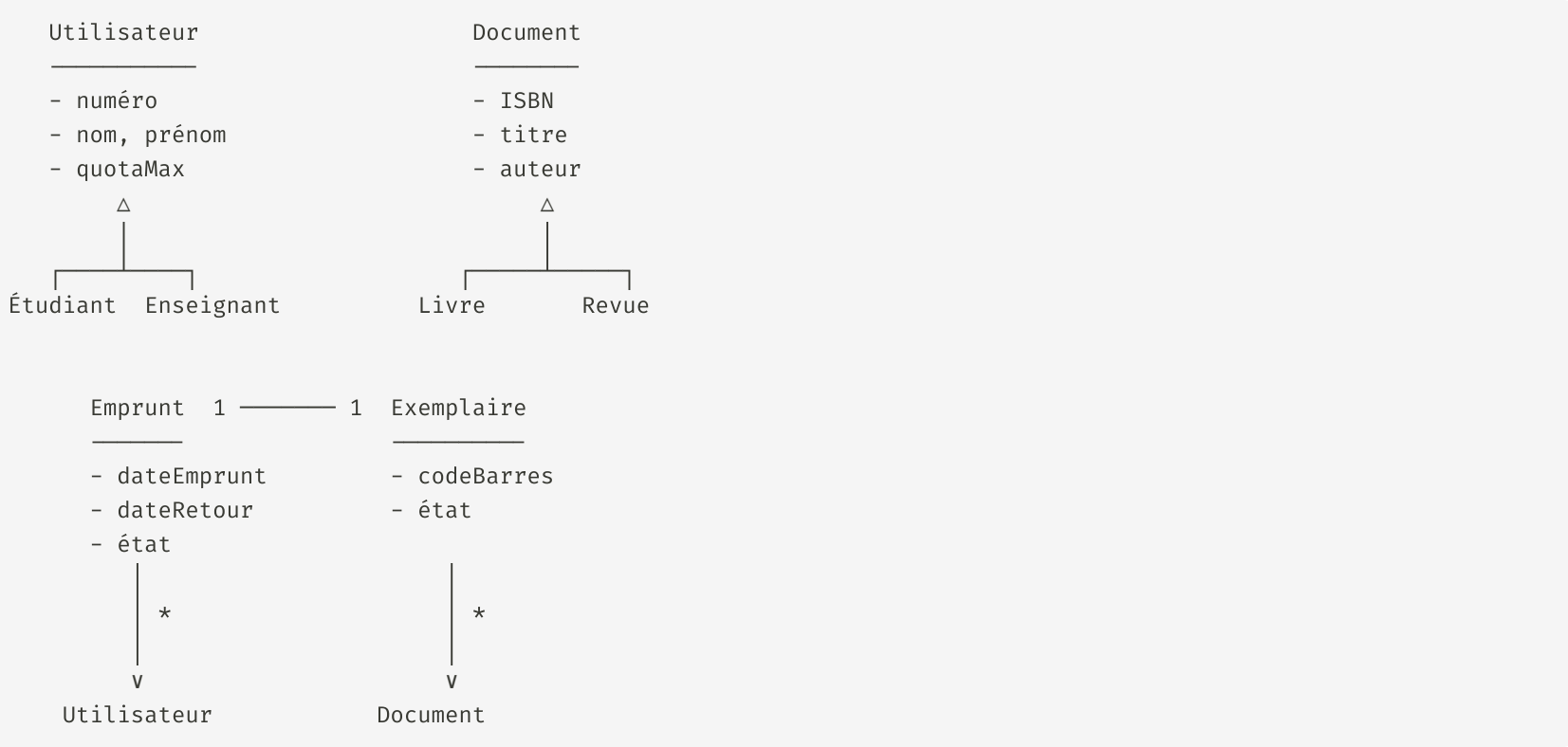
- "Est un" (spécialisation)
- Exemple : Livre → Document

Multiplicités

- `1` : exactement 1
- `0..1` : 0 ou 1
- `*` ou `0..*` : 0 ou plusieurs
- `1..*` : 1 ou plusieurs
- `n..m` : entre n et m

Diagramme de classes - Bibliothèque

Modèle conceptuel simplifié



Relations clés

Multiplicités - Exemples

Lecture des cardinalités

Exemples de lecture

`Utilisateur 1 ── 0..* Emprunt` → Un utilisateur a 0+ emprunts

`Emprunt * ── 1 Exemplaire` → Un emprunt = 1 exemplaire

`Document 1 ── 1..* Exemplaire` → Un document a au moins 1 copie

Contraintes métier

À documenter séparément :

- Étudiant : max 3 emprunts
- Enseignant : max 5 emprunts
- Exemplaire abîmé : non empruntable
- Retard > 30j : compte bloqué

Attributs vs Classes

Quand créer une classe ?

Utiliser un ATTRIBUT si :

- Type primitif (int, string, date)
- Valeur simple
- Pas d'opération associée

Exemples

```
Utilisateur
————————————
- nom: string
- prénom: string
- dateNaissance: date
- actif: boolean
```

Utiliser une CLASSE si :

- Valeur composite
- Opérations associées
- Utilisé par plusieurs classes
- A un cycle de vie propre

Exemples

```
Utilisateur
————————————
- adresse: ???

attribut string ?
 classe Adresse
   - rue
   - ville
   - codePostal
```

Erreurs courantes à éviter

Pièges classiques

Classes trop techniques

```
Base de données
Connexion
Controller
Manager
```

Ce sont des choix d'implémentation, pas des concepts métier.

Classes redondantes

```
Livre ET Ouvrage
Client ET Utilisateur
```

Choisir un seul terme (glossaire).

Attributs qui sont des classes

```
Livre
   - auteur: string
```

Auteur devrait être une classe si on veut gérer plusieurs livres du même auteur.

** Bon équilibre**

- 5-15 classes pour un petit système
- Ni trop granulaire, ni trop gros
- Penser "domaine métier"

Exercice pratique

À vous de jouer !

Système de réservation de salles

Une université veut gérer la réservation de ses salles de cours.

- Les enseignants peuvent réserver des salles pour leurs cours
- Les étudiants peuvent consulter les disponibilités
- Le personnel administratif gère les salles et les équipements
- Certaines salles ont des équipements spécifiques (vidéoprojecteur, ordinateurs)
- Les réservations doivent éviter les conflits horaires

Questions

1. Identifier les acteurs
2. Lister 5 cas d'utilisation principaux
3. Dessiner un diagramme de séquence pour "Réserver une salle"
4. Identifier les classes principales du domaine

Application à la construction d'une application

Comment analyser un problème de triangulation

Du simple au complexe

Appliquer ACOL à la triangulation 3D

Vous avez appris

- Identifier les acteurs
- Définir les cas d'utilisation
- Créer des diagrammes de séquence
- Modéliser le domaine avec des classes

Maintenant : appliquer au projet réel

Projet : Triangulation de nuages de points

Contexte simplifié :

- Scanner 3D → fichier de points
- Logiciel → reconstruit la surface
- Utilisateur scientifique → visualise/exporte

Défi : modéliser un système algorithmique

Analyse de la situation - Acteurs

Identification pour un système algorithmique

Acteurs principaux

Utilisateur scientifique

- Chercheur en vision 3D
- Ingénieur en rétro-ingénierie
- Étudiant en projet

Besoins

- Charger des données
- Lancer la reconstruction
- Ajuster les paramètres
- Visualiser les résultats
- Exporter le maillage

Acteurs secondaires

Système de fichiers

- Fournit les fichiers .pts
- Stocke les résultats

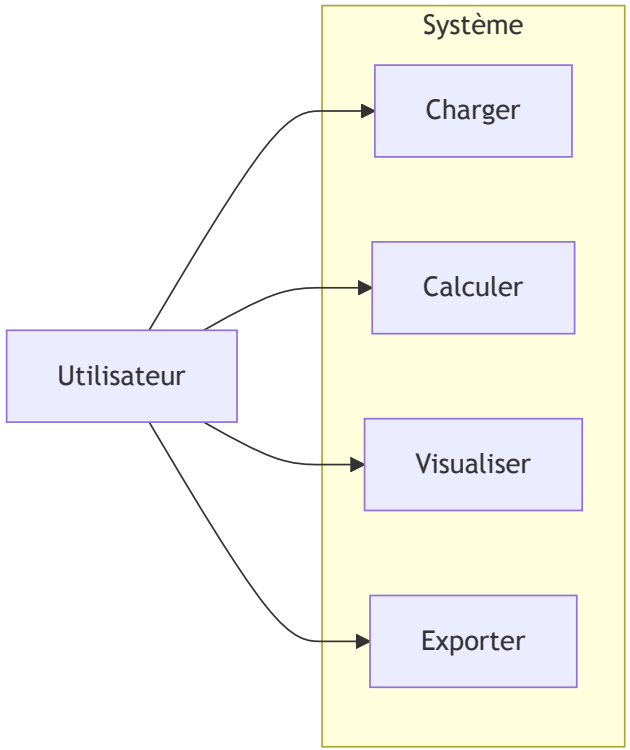
Développeur/Mainteneur

- Configure les algorithmes
- Optimise les performances
- Debug

Visualiseur 3D

- Affiche le nuage de points
- Affiche le maillage

Cas d’utilisation - Triangulation



Workflow principal

Cas d’usage métier

1. Charger nuage de points (.pts)
2. Calculer triangulation (avec config paramètres)
3. Visualiser résultat 3D
4. Exporter maillage (.obj)

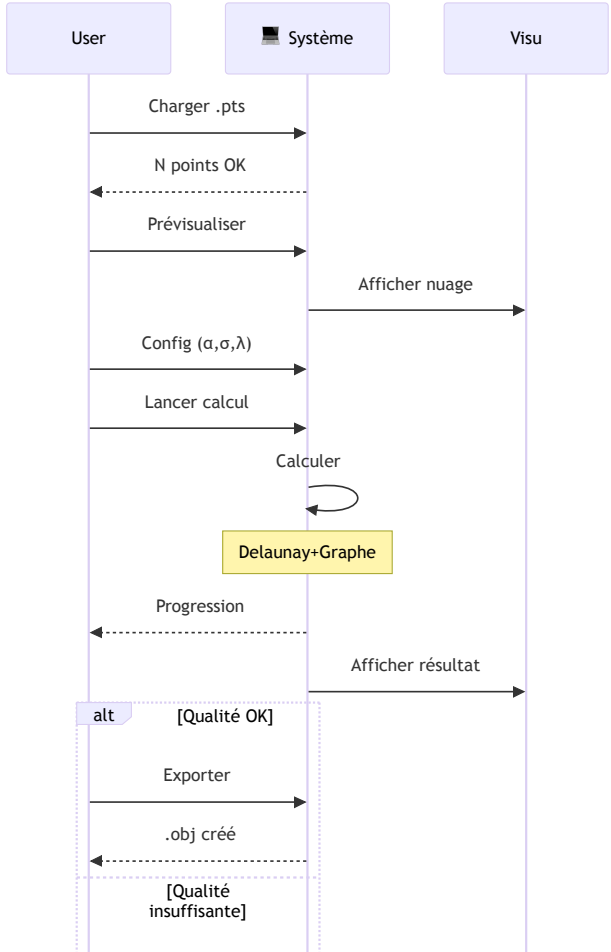
Acteur

- Utilisateur scientifique (chercheur, ingénieur)

Note importante

- Niveau **métier**, pas algorithmique
- Pas de détails sur Delaunay, graphes, etc.

Diagramme de séquence - Triangulation



Scénario complet

1. **Chargement** : Fichier .pts parsé
2. **Prévisualisation** : Nuage affiché en 3D
3. **Configuration** : Paramètres α, σ, λ
4. **Calcul** : Triangulation (peut prendre du temps)
5. **Visualisation** : Maillage 3D affiché
6. **Évaluation** : Statistiques de qualité
7. **Export** : Si OK → fichier .obj

Points clés

- Progression affichée pendant calcul
- Boucle itérative possible (réajustement)
- Vue **métier**, pas détails algo

Classes du domaine - Triangulation

Modèle conceptuel (analyse, pas conception)

```
NuageDePoints  1 ──── *  Point3D
──────────────              ───────
- source                    - x, y, z
- nbPoints
- bornes

  Maillage    1 ──── *  Triangle
  ────────              ────────
  - nbTriangles         - sommets[3]
  - qualité             - normale

Paramètres
──────────
- alpha, sigma, lambda
- nbPointsVue
```

Concepts clés : NuageDePoints (entrée), Maillage (sortie), Paramètres (configuration)

On modélise les **concepts métier**, pas les algorithmes (Delaunay, Graphe).

De l’analyse à la conception

Ce que vous ferez ensuite

Phase d’analyse

- Cas d’utilisation
- Séquences
- Modèle du domaine

Ce qui reste flou

- Comment calculer Delaunay ?
- Quelle structure de données ?
- Algorithme de flot max ?

Phase de conception (prochains cours)

- Architecture détaillée
- Classes techniques
  - `DelaunayTriangulation<n>`
  - `Graphe`, `Simplexe<n>`
- Structures de données optimisées
- Choix algorithmiques

Implémentation

- Code C++
- Templates
- Optimisations

Important : L'analyse reste au niveau métier. La conception entre dans les détails techniques.

Conclusion

À retenir

Concepts clés du cours

Pourquoi l'analyse ?

- 30% des échecs = besoins mal compris
- Analyser avant de coder
- Communiquer avec le client
- Éviter les malentendus coûteux

Outils UML

- **Cas d'utilisation** : qui fait quoi ?
- **Séquence** : dans quel ordre ?
- **États-transitions** : quels états ?
- **Classes** : quels concepts métier ?

Principes

- Rester au niveau **métier** (pas technique)
- Utiliser le **vocabulaire du domaine**
- **Valider** avec le client
- **Documenter** pour l'équipe

Livrables d'analyse

- Glossaire
- Diagrammes UML
- Descriptions textuelles
- Spécifications validées

Méthodologie en 4 étapes

Processus d'analyse simplifié

```
COMPRENDRE              CAS D'UTILISATION
   (1-2 jours)             (2-3 jours)

   • Lire CDC              • Identifier acteurs
   • Rencontrer client     • Lister cas d'usage
   • Créer glossaire       • Décrire scénarios
        ↓                       ↓

MODÉLISER                VALIDER
   (2-3 jours)             (1 jour)

   • Classes métier        • Présenter au client
   • Relations             • Corriger
   • Diagrammes            • Finaliser
```

Durée totale : ~1 semaine pour un projet de taille moyenne

Bonnes pratiques

Do’s and Don’ts

À faire

- Impliquer le client régulièrement
- Utiliser le vocabulaire métier
- Rester simple et clair
- Itérer (analyser → valider → corriger)
- Documenter les décisions
- Créer un glossaire
- Penser aux cas d’erreur

À éviter

- Coder avant d’analyser
- Utiliser du jargon technique en analyse
- Faire des diagrammes trop complexes
- Modéliser l’implémentation
- Ignorer les retours client
- Oublier les exceptions
- Supposer que "c’est évident"

La prochaine fois

Cours 5 : Conception objet et Design Patterns

De l'analyse à la conception

- Architecture logicielle
- Principes SOLID
- Découpage en couches
- Responsabilités des classes

Design Patterns essentiels

- Singleton
- Factory
- Strategy
- Observer
- Et autres patterns utiles

Préparation : Révisez les diagrammes de classes. Nous allons passer à la conception technique avec des patterns réutilisables.