De l'Analyse à la Conception

La Modélisation Objet avec le Diagramme de Classes du Domaine

Processus de Modélisation

14 octobre 2025

Agenda de la session

1. La transition fondamentale : du problème à la solution
2. Le Diagramme de Classes du Domaine : notre dictionnaire visuel
3. Ce qu'il contient et ce qu'il exclut
4. Les 3 objectifs clés de ce diagramme
5. Méthodologie : comment le construire ?
6. Cas pratique : modélisation de "InnovateBox"

La Transition Fondamentale

Où en sommes-nous ? L'analyse

Nous avons capturé le besoin

- **User stories :** le "pourquoi" et le "pour qui"
- **Cas d'utilisation :** le "comment" du point de vue de l'utilisateur

Nous avons défini CE QUE le système doit faire.

Nous sommes dans le **monde de l'utilisateur**, avec son langage.

L'étape suivante : la conception

Il est temps d'opérer une transition fondamentale :

Passer du domaine du problème (les besoins métier)

↓

Au domaine de la solution (la structure du logiciel)

Pour construire ce pont, nous avons besoin d'un plan : le **Diagramme de Classes**.

Le Piège Majeur : Penser "Code" Trop Tôt

Attention : danger !
Penser immédiatement en termes de bases de données, de code, d'interfaces techniques est prématuré.

Avant de se demander *comment* coder une `Facture`...

...nous devons nous accorder sur ce qu'EST une `Facture` dans le contexte du métier.

Le Diagramme de Classes du Domaine

Le Diagramme de Classes du Domaine

Notre premier objectif
Non pas concevoir le logiciel, mais **modéliser le domaine métier lui-même**.

C'est le diagramme le plus important de la modélisation orientée objet.

Considérez-le comme un dictionnaire visuel, validé par le client, qui définit les concepts clés de son activité.

Ce qu'il contient (1/3) : Les Classes Conceptuelles

Ce sont les "choses", les concepts importants du domaine métier.

Comment les trouver ? En identifiant les noms dans les User Stories et les Cas d'Utilisation.

Exemples
`Client`, `Produit`, `Commande`, `Facture`

Ce qu'il contient (2/3) : Les Attributs

Ce sont les propriétés pertinentes qui décrivent les concepts.

On ne liste que les attributs qui ont un sens pour le métier à ce stade.

Exemples

- Un `Produit` a un `nom` et un `prix`.
- Un `Client` a une `adresse_de_livraison`.

Ce qu'il contient (3/3) : Les Associations

Ce sont les relations et les verbes d'action qui lient les concepts entre eux.

Comment les trouver ? En identifiant les verbes qui connectent les noms.

Exemples

- Un `Client` *passe* une ou plusieurs `Commandes`.
- Une `Commande` *contient* plusieurs `Produits`.

Les associations sont cruciales car elles formalisent les **règles de gestion**.

Ce qu'il exclut volontairement (1/3) : Les Méthodes

On ne se préoccupe pas encore des actions ou des opérations.

Logique
Les méthodes (`calculerTotal()`, `sauvegarderEnBase()`) relèvent de la conception, du COMMENT.

Pour l'instant, on se concentre sur le QUOI.

Ce qu'il exclut volontairement (2/3) : Les Types Techniques

On n'indique pas la représentation technique des données.

Exemple
On écrit `prix`, pas `double`, `float` ou `Decimal`.

Ce choix sera fait plus tard, durant la phase de conception technique détaillée.

Ce qu'il exclut volontairement (3/3) : Les Classes Techniques

On ne modélise pas les classes qui sont des artefacts de la solution logicielle.

Exemples de classes à exclure à ce stade
`DatabaseManager`, `PDFController`, `InterfaceUtilisateur`

Ces classes n'existent pas dans le domaine métier du client.

Les Objectifs

Objectifs

1. Valider la Compréhension : Fournir un support de discussion avec le client pour s'assurer que notre compréhension de son activité est correcte et complète.
2. Créer un Langage Commun : Établir un vocabulaire stable et non-ambigu qui sera utilisé par les analystes, les développeurs et le client.
3. Bâtir les Fondations : Servir de base solide et stable pour la phase de conception technique. Ce diagramme conceptuel sera ensuite enrichi pour devenir un diagramme de classes de conception, prêt à être implémenté.

Bonnes et Mauvaises Pratiques

Bonne Pratique

Modéliser les concepts du monde réel et leurs relations

- Les noms de classe sont des substantifs.
- Les attributs sont simples.
- Les associations représentent des verbes.

```
[Client] 1 - 0..* [Commande]
[Commande] * - 1..* [Produit]
```

Classes : `Client`, `Commande`, `Produit`.

Attributs : `nom`, `email`, `prix`.

Mauvaise Pratique

Inclure des détails d'implémentation technique

- Classes de frameworks (`...Controller`, `...Service`).
- Classes utilitaires (`DatabaseManager`).
- Types de collections (`ArrayList<Product>`).
- Méthodes (`saveToDatabase()`).

```
[ClientController] -> [ClientService] -> [ClientRepository]
```

Pourquoi ? C'est un diagramme de *conception* ou d'*architecture*, pas un diagramme de **domaine**. Il répond au "comment", pas au "quoi".

Méthodologie

Méthodologie en 3 Étapes Clés

La technique consiste à "lire entre les lignes" de nos spécifications.

1. **Identifier les classes candidates :**
   Cherchez les noms et groupes nominaux importants.

2. **Identifier les attributs :**
   Cherchez les propriétés qui décrivent ces noms.

3. **Identifier les associations :**
   Cherchez les verbes qui lient ces noms entre eux.

Le Travail de Raffinement

Une fois la première ébauche réalisée, il faut affiner :

- Éliminer les doublons et clarifier les concepts.
- Transformer des noms en attributs si ce ne sont que de simples propriétés.
- Définir les **multiplicités** (cardinalités) pour chaque association.

Exemple de question pour les multiplicités
Un `Employé` peut-il soumettre *plusieurs* `Idées` ?

→ Oui → `Employé 1 - 0..* Idée`

Cas Pratique : InnovateBox

Application Pratique : InnovateBox

Matériel de base (nos récits)

- *US1 :* En tant qu'**employé**, je veux soumettre une **idée** avec un **titre** et une **description**...
- *US2 :* En tant que **manager**, je veux consulter la **liste** des **idées**...
- *US3 :* En tant qu'**employé**, je veux **voter** pour une **idée**...
- *CU-Soumettre :* L'**Employé** saisit un **titre** et une **description**. Le système enregistre l'**idée**.

Étape 1 : Extraction des Noms (Classes Candidates)

En lisant les récits, nous surlignons les noms pertinents :

Employé, Idée, Titre, Description, Manager, Vote

Notre liste brute de classes potentielles est :

`Employé`, `Idée`, `Titre`, `Description`, `Manager`, `Vote`

Étape 2 : Raffinage (Concepts et Attributs)

Analysons cette liste :

- `Employé` et `Idée` : concepts centraux → **Bonnes classes**.

- `Titre` et `Description` : ce ne sont pas des concepts autonomes, mais des propriétés d'une 'Idée' → **Deviennent des attributs**.

Étape 2 : Raffinage (Héritage)

Analyse (suite) :

- `Manager` : est-il différent d'un `Employé` ?

Un Manager *est un* Employé avec des droits supplémentaires.
C'est un cas de **généralisation/spécialisation (héritage)**.

→ La classe `Manager` hérite de la classe `Employé`.

Étape 2 : Raffinage (Classe-Association)

Analyse (fin) :

- `Vote` : est-ce un simple compteur ?

Non, car le système doit mémoriser **QUI** a voté pour **QUOI**. Un vote est une relation entre un `Employé` et une `Idée`.

→ Candidate idéale pour une **classe-association**.

Résultat des étapes 1 et 2

Classes retenues

- `Employé`
- `Manager`
- `Idée`
- `Vote`

Attributs identifiés

- Pour `Idée` :
  - `titre`
  - `description`
- Pour `Employé` :
  - `nom` (implicite)

Étape 3 : Associations et Multiplicités (Auteur)

Nous analysons les verbes liant nos classes.

Un `Employé` *soumet* une `Idée`

- Une `Idée` est soumise par **exactement 1** `Employé`.
- Un `Employé` peut soumettre **0 ou plusieurs** `Idées`.

```
[Employé] 1 - 0..* [Idée]
```

Étape 3 : Associations et Multiplicités (Vote)

Un `Employé` *vote pour* une `Idée` via un `Vote`

- Un `Vote` est émis par **exactement 1** `Employé`.
- Un `Vote` concerne **exactement 1** `Idée`.
- Un `Employé` peut émettre **plusieurs** `Votes`.
- Une `Idée` peut recevoir **plusieurs** `Votes`.

```
[Employé] 1 - 0..* [Vote]      [Idée] 1 - 0..* [Vote]
```

Résultat : Le Diagramme de Domaine "InnovateBox"

Notre Dictionnaire Visuel est complet

- Une classe `Employé` avec l'attribut ‘nom‘.

Résultat : Le Diagramme de Domaine "InnovateBox"

Notre Dictionnaire Visuel est complet

- Une classe `Employé` avec l'attribut 'nom'.
- Une classe `Manager` qui hérite de `Employé`.

Résultat : Le Diagramme de Domaine "InnovateBox"

Notre Dictionnaire Visuel est complet

- Une classe `Employé` avec l'attribut ‘nom‘.
- Une classe `Manager` qui hérite de `Employé`.
- Une classe `Idée` avec les attributs ‘titre‘ et ‘description‘.

Résultat : Le Diagramme de Domaine "InnovateBox"

Notre Dictionnaire Visuel est complet

- Une classe `Employé` avec l'attribut ‘nom‘.
- Une classe `Manager` qui hérite de `Employé`.
- Une classe `Idée` avec les attributs ‘titre‘ et ‘description‘.
- Une classe `Vote` (avec ‘dateDeVote‘ par exemple).

Résultat : Le Diagramme de Domaine "InnovateBox"

Notre Dictionnaire Visuel est complet

- Une classe `Employé` avec l'attribut ‘nom‘.
- Une classe `Manager` qui hérite de `Employé`.
- Une classe `Idée` avec les attributs ‘titre‘ et ‘description‘.
- Une classe `Vote` (avec ‘dateDeVote‘ par exemple).
- Association "estAuteurDe" entre `Employé` (1) et `Idée` (0..*).

Résultat : Le Diagramme de Domaine "InnovateBox"

Notre Dictionnaire Visuel est complet

- Une classe `Employé` avec l'attribut ‘nom‘.
- Une classe `Manager` qui hérite de `Employé`.
- Une classe `Idée` avec les attributs ‘titre‘ et ‘description‘.
- Une classe `Vote` (avec ‘dateDeVote‘ par exemple).
- Association "estAuteurDe" entre `Employé` (1) et `Idée` (0..*).
- `Vote` est connecté à `Employé` (1) et à `Idée` (1).

Conclusion

Le Diagramme de Classes du Domaine est une étape **non-technique** mais **fondamentale**.

Il permet de :

- Valider la compréhension du métier.
- Unir les équipes autour d'un vocabulaire commun.
- Construire des fondations solides pour le développement.

<5-> C'est le pont réussi entre le **domaine du problème** et le **domaine de la solution**.

Conclusion

Le Diagramme de Classes du Domaine est une étape **non-technique** mais **fondamentale**.

Il permet de :

- Valider la compréhension du métier.
- Unir les équipes autour d'un vocabulaire commun.
- Construire des fondations solides pour le développement.

<5-> C'est le pont réussi entre le **domaine du problème** et le **domaine de la solution**.

Conclusion

Le Diagramme de Classes du Domaine est une étape **non-technique** mais **fondamentale**.

Il permet de :

- Valider la compréhension du métier.
- Unir les équipes autour d'un vocabulaire commun.
- Construire des fondations solides pour le développement.

<5-> C'est le pont réussi entre le **domaine du problème** et le **domaine de la solution**.

Conclusion

Le Diagramme de Classes du Domaine est une étape **non-technique** mais **fondamentale**.

Il permet de :

- Valider la compréhension du métier.
- Unir les équipes autour d'un vocabulaire commun.
- Construire des fondations solides pour le développement.

<5-> C'est le pont réussi entre le **domaine du problème** et le **domaine de la solution**.

Questions ?