Analyse
Note de Cours : de l'éxigence à la spécification

Ressource R3.03 :

15 octobre 2025

Table des matières

Introduction : Pourquoi l'analyse est-elle cruciale ?

En développement logiciel, une erreur détectée lors de la phase d'analyse coûte beaucoup moins cher à corriger qu'une erreur trouvée une fois l'application déployée. La phase d'analyse est un investissement : elle vise à s'assurer que l'on construit **le bon produit** avant de se demander si on le construit **de la bonne manière**. Cette ressource vous donne les outils pour transformer un besoin flou en un cahier des charges clair et actionnable. Les principes que nous allons détailler ne sont pas de simples abstractions ; leur non-respect a mené à certains des échecs les plus coûteux de l'histoire de l'informatique, comme l'illustrent les exemples présentés en annexe.

1 L'analyse des Éxigences (Requirements Analysis)

1.1 Recueil des besoins : l'art de poser les bonnes questions

Recueillir un besoin, ce n'est pas seulement prendre des notes, c'est mener une enquête. Votre objectif est de découvrir les besoins réels, souvent cachés derrière les demandes initiales. Le rôle de l'analyste n'est pas celui d'un simple scribe qui transcrit passivement une commande, mais celui d'un enquêteur qui cherche activement à comprendre le contexte, les frustrations et les objectifs finaux du métier.

Il faut savoir faire la distinction fondamentale entre la *demande* (la solution envisagée ou formulée par le client) et le *besoin* réel (le problème fondamental qui doit être résolu). Un client, en pensant déjà à une solution, pourrait demander « Je veux un bouton pour exporter ces données en PDF ». L'enquête de l'analyste doit chercher le « pourquoi » derrière cette demande. Le besoin réel pourrait être : « Je dois pouvoir partager un rapport synthétique et non-modifiable avec ma direction chaque semaine ». En comprenant cela, une solution bien plus pertinente, peut-être entièrement automatisée, pourrait émerger.

Cette démarche est cruciale car la première demande formulée par un client n'est souvent que le symptôme du problème, et non sa cause profonde. Votre mission est de creuser sous la

surface pour construire une solution qui guérit la maladie, et non une qui ne fait que masquer les symptômes.

À faire : Bonne pratique

Posez des questions ouvertes qui encouragent la discussion et la découverte.

- « Pourriez-vous me décrire une journée type de travail avec le système actuel ? »
- « Qu'est-ce qui vous frustre le plus aujourd'hui dans la gestion de vos commandes ? »
- « Si vous aviez une baguette magique, que ferait l'application pour vous ? »

Pourquoi ? Ces questions révèlent le contexte, la douleur de l'utilisateur et la vraie valeur attendue.

À ne pas faire : Mauvaise pratique

Se contenter de questions fermées ou qui suggèrent la solution.

- « Voulez-vous un bouton bleu ici ? »
- « Donc il vous faut juste un tableau avec les noms et les dates ? »
- « Le système doit-il utiliser une base de données SQL ? »

Pourquoi ? Ces questions limitent la conversation, ferment la porte à des besoins non anticipés et orientent prématurément vers des détails techniques.

Exercice 1 : L'analyste-enquêteur

*Cet exercice a pour but d'évaluer la capacité à distinguer la **demande** (la solution) du **besoin** (le problème) et à formuler les questions d'investigation adéquates.*

Contexte :

Vous êtes analyste pour une entreprise qui développe un logiciel de gestion pour une chaîne de restaurants. Lors d'une réunion, le manager d'un des restaurants vous fait la demande suivante :

> « Je veux un tableau de bord avec un gros compteur rouge qui affiche en temps réel le nombre de tables qui sont occupées depuis plus de 90 minutes. »

Instructions :

1. **Identifier la demande** : Quelle est la demande explicite et orientée "solution" formulée par le manager ?
2. **Identifier le besoin potentiel** : D'après vous, quel pourrait être le problème métier (le besoin réel) qui se cache derrière cette demande ? Proposez au moins deux hypothèses.
3. **Mener l'enquête** : En vous basant sur la section "À faire : Bonne pratique", formulez trois questions ouvertes que vous poseriez au manager pour découvrir son besoin réel et le contexte de sa frustration.

Correction suggérée

1. **La demande** : La demande est la solution spécifique proposée par le client : un tableau de bord affichant un compteur rouge pour les tables occupées depuis plus de 90 minutes.
2. **Les besoins potentiels (hypothèses)** :
   - **Hypothèse A (Rotation)** : Le restaurant perd de l'argent car les clients restent trop longtemps pendant les heures de pointe, empêchant d'accueillir de nouveaux clients. Le besoin est d'optimiser la rotation des tables.
   - **Hypothèse B (Service client)** : Les clients qui restent longtemps sont peut-être des clients oubliés par les serveurs. Le besoin est d'améliorer la qualité du service en s'assurant qu'aucune table n'est délaissée.
   - **Hypothèse C (Gestion des ressources)** : Le manager a du mal à savoir quand il peut libérer des serveurs en fin de service et a besoin d'un indicateur visuel pour anticiper la fin des repas.
3. **Questions d'investigation** :
   - « Pourriez-vous me décrire ce qui se passe aujourd'hui lorsqu'une table est occupée pendant une longue durée ? Quel impact cela a-t-il sur votre service ? »
   - « Que cherchez-vous à accomplir une fois que vous avez l'information qu'une table est occupée depuis plus de 90 minutes ? Quelle action cela déclenche-t-il ? »
   - « Racontez-moi la dernière fois où une situation liée à une table occupée trop longtemps a posé un vrai problème. Qu'est-ce qui s'est passé ? »

Exercice 2 : Transformer les mauvaises questions

Cet exercice a pour but de s'entraîner à reconnaître les questions fermées ou suggestives et à les transformer en questions ouvertes favorisant la découverte.

Contexte :

Un analyste débutant mène une réunion pour comprendre les besoins d'une équipe de support client qui souhaite améliorer son outil de gestion de tickets.

Instructions :
Pour chaque question posée par l'analyste ci-dessous, identifiez si elle correspond à une "Bonne pratique" ou une "Mauvaise pratique" selon les principes du cours. Si c'est une "Mauvaise pratique", reformulez-la en une question ouverte et efficace.

1. « Donc, pour clore un ticket, il vous faut juste un bouton "Terminé" ? »
2. « Qu'est-ce qui vous prend le plus de temps aujourd'hui quand vous traitez une demande client ? »
3. « On devrait trier les tickets par ordre alphabétique ou par date ? »
4. « Pourriez-vous me guider à travers les étapes que vous suivez lorsqu'un nouveau ticket arrive, du début à la fin ? »
5. « Vous voulez que le système envoie une notification par email au client à chaque mise à jour du ticket ? »

Correction suggérée

1. **Question :** « Donc, pour clore un ticket, il vous faut juste un bouton "Terminé" ? »
   - **Type :** Mauvaise pratique. C'est une question fermée qui suggère une solution.
   - **Reformulation :** « Parlez-moi du cycle de vie d'un ticket. Quelles sont les différentes étapes avant qu'un ticket puisse être considéré comme résolu ? »
2. **Question :** « Qu'est-ce qui vous prend le plus de temps aujourd'hui quand vous traitez une demande client ? »
   - **Type :** Bonne pratique. C'est une question ouverte qui invite à décrire les frustrations.
3. **Question :** « On devrait trier les tickets par ordre alphabétique ou par date ? »
   - **Type :** Mauvaise pratique. Elle propose un choix limité et ferme la porte à d'autres critères plus utiles.
   - **Reformulation :** « Quand vous regardez votre liste de tickets, comment décidez-vous lequel traiter en premier ? Quelles informations sont les plus importantes pour vous à ce moment-là ? »
4. **Question :** « Pourriez-vous me guider à travers les étapes que vous suivez lorsqu'un nouveau ticket arrive, du début à la fin ? »
   - **Type :** Bonne pratique. Elle invite à décrire un processus complet de manière ouverte.
5. **Question :** « Vous voulez que le système envoie une notification par email au client à chaque mise à jour du ticket ? »
   - **Type :** Mauvaise pratique. C'est une question fermée qui suggère une solution technique et une fréquence.
   - **Reformulation :** « À quels moments est-il important de communiquer avec le client au sujet de son ticket ? Quelles informations sont essentielles à lui transmettre et par quel moyen ? »

1.2 Exigences Fonctionnelles (EF) vs Non-Fonctionnelles (ENF)

Toute exigence, pour être utile, doit posséder trois qualités fondamentales : elle doit être **non-ambiguë**, **complète**, et surtout, **vérifiable** (ou testable).

- Une exigence ambiguë est une porte ouverte à l'interprétation. Le développeur et le client comprendront deux choses différentes, ce qui mènera inévitablement à un produit qui ne correspond pas aux attentes réelles.

— Une exigence incomplète force l'équipe de développement à faire des suppositions, comblant les « trous » avec sa propre logique. Ces suppositions sont une source majeure de défaillances de projet.
— Enfin, une exigence non vérifiable est un contrat sans valeur. Si l'on ne peut pas mesurer ou tester objectivement si l'exigence est satisfaite, alors on ne peut jamais prouver que le travail est terminé et correct. Une exigence non-testable n'est pas une spécification technique, c'est un simple vœu.

C'est sur ce critère de vérifiabilité que la distinction entre les types d'exigences devient particulièrement importante. Pour organiser et spécifier correctement le travail à réaliser, nous séparons les exigences en deux grandes familles qui répondent à deux questions distinctes :

Les Exigences Fonctionnelles (EF) décrivent *ce que le système doit faire.* Elles définissent les fonctionnalités, les comportements et les actions spécifiques du logiciel. (Exemple : « L'utilisateur doit pouvoir se connecter avec un email et un mot de passe »).

Les Exigences Non-Fonctionnelles (ENF) décrivent *comment le système doit se comporter.* Elles définissent les qualités, les contraintes et les caractéristiques du système, comme la performance, la sécurité, ou la fiabilité. (Exemple : « Le temps de connexion doit être inférieur à 2 secondes »).

1.2.1 Exemples d'Exigences Fonctionnelles (EF)

Les exigences fonctionnelles décrivent une action, une entrée ou une sortie du système. Elles sont souvent formulées avec un verbe d'action.

— **Gestion des utilisateurs :**
  — *Le système doit permettre à un administrateur de créer, modifier, et désactiver des comptes utilisateurs.*
  — *L'utilisateur doit pouvoir réinitialiser son mot de passe via un lien envoyé par email.*
— **Fonctionnalités métier (site e-commerce) :**
  — *Le système doit appliquer une réduction de 10% au panier si celui-ci contient plus de 3 articles identiques.*
  — *Le système doit générer une facture au format PDF et l'associer à l'historique de commande du client.*
  — *Le système doit autoriser le paiement via Carte Bancaire et PayPal.*
— **Reporting et Administration :**
  — *Le système doit fournir à l'administrateur un tableau de bord listant les 10 produits les plus vendus sur les 30 derniers jours.*
  — *Le système doit permettre l'export des transactions d'une période donnée au format CSV.*

1.2.2 Exemples d'Exigences Non-Fonctionnelles (ENF)

Les exigences non-fonctionnelles décrivent la *qualité* avec laquelle une action est effectuée ou une contrainte que le système doit respecter. Elles sont souvent quantifiables.

— **Performance et Rapidité :**
  — *Toutes les pages du site doivent se charger en moins de 3 secondes avec une connexion ADSL à 8 Mbps.*
  — *Le système doit pouvoir traiter 100 transactions par minute sans perte de performance.*
  — *La génération du rapport de ventes annuel ne doit pas excéder 5 minutes.*
— **Sécurité :**
  — *Tous les mots de passe des utilisateurs doivent être stockés en base de données en utilisant l'algorithme de hachage Argon2id.*
  — *Le système doit automatiquement déconnecter un utilisateur après 20 minutes d'inactivité.*

    - *Le système doit interdire l'accès aux pages d'administration depuis une adresse IP non déclarée.*
- **Fiabilité et Disponibilité (Reliability) :**
    - *Le système doit être disponible 99.8% du temps, hors plages de maintenance planifiées (les mardis de 2h à 4h du matin).*
    - *En cas d'échec de la communication avec le service de paiement, la transaction doit être mise en attente et re-tentée automatiquement 3 fois à 5 minutes d'intervalle.*
- **Utilisabilité et Accessibilité (Usability) :**
    - *Le parcours d'achat, de l'ajout au panier à la confirmation de paiement, ne doit pas nécessiter plus de 5 clics.*
    - *L'ensemble des fonctionnalités doit être accessible via la navigation au clavier.*
    - *Le site doit respecter le référentiel d'accessibilité WCAG 2.1 au niveau AA.*
- **Maintenabilité :**
    - *Tous les développements devront respecter la convention de nommage CamelCase et suivre le standard de code PSR-12.*
    - *Toute fonction métier critique doit être couverte par des tests unitaires avec une couverture de code d'au moins 80%.*

1.2.3 Quand une exigence fonctionnelle devient non-fonctionnelle (et inversement)

La frontière entre ces deux catégories, bien que claire en théorie, dépend fortement du contexte et de la finalité première du système étudié. Une même action peut être considérée comme fonctionnelle dans un cas, et non-fonctionnelle dans un autre.

Prenons un exemple concret : « le système doit analyser un fichier pour détecter la présence de virus ».

- **Contexte 1 : Un logiciel antivirus.** Dans ce cas, l'analyse antivirus est la raison d'être du produit. C'est l'action principale que l'utilisateur veut consciemment déclencher. L'exigence « le système doit scanner les fichiers sélectionnés pour y trouver des logiciels malveillants » est donc une **exigence fonctionnelle** fondamentale.
- **Contexte 2 : Une application web de partage de photos.** L'utilisateur veut téléverser sa photo de profil. L'objectif fonctionnel est d'afficher cette photo sur son profil. Pour des raisons de sécurité, le système doit analyser en arrière-plan le fichier téléversé pour s'assurer qu'il ne s'agit pas d'un cheval de Troie. Ici, « analyser le fichier pour détecter des virus » n'est pas ce que l'utilisateur demande ; c'est une contrainte de qualité, une mesure de sécurité qui définit *comment* le système doit se comporter lors du téléversement. C'est donc une **exigence non-fonctionnelle** (de sécurité).

Ainsi, la question clé pour classifier une exigence est la suivante : « Est-ce que cette action représente une finalité pour l'utilisateur, ou est-ce une contrainte de qualité qui encadre l'exécution d'une autre finalité ? ». La réponse à cette question dépendra toujours du cas d'usage spécifique du système.

Si les exigences fonctionnelles sont souvent plus directes à formuler de manière testable, les exigences non-fonctionnelles représentent un défi bien plus grand. Comment « vérifier » la performance, la robustesse ou la maintenabilité ? C'est ce que nous allons explorer.

Résumé : Exigence Fonctionnelle (EF) - ce que le système FAIT.

À faire : Bonne pratique

Être précis et décrire une action ou une capacité observable.

- « Le système **doit permettre** à un administrateur de **supprimer** le compte d'un utilisateur. »

— « Lorsque la commande est validée, le système **doit envoyer** un email de confirmation au client contenant le récapitulatif et le numéro de commande. »

À ne pas faire : Mauvaise pratique

Rester vague ou décrire des intentions.

— « Le système doit bien gérer les utilisateurs. » *(Que signifie "bien gérer" ?)*
— « Le système doit s'occuper des commandes. » *(Quelles actions ? À quel moment ?)*

Résumé : Exigence Non-Fonctionnelle (ENF) - COMMENT le système le fait.

À faire : Bonne pratique

Utiliser des métriques quantifiables et mesurables.

Performance : « La page d'accueil **doit se charger en moins de 2 secondes** avec une connexion ADSL de 8 Mbit/s. »

Sécurité : « Les mots de passe des utilisateurs **doivent être stockés** en utilisant l'algorithme de hachage **Bcrypt** avec un coût de 12. »

Sociétale (`AC22.04`) : « L'application **doit respecter** le niveau **AA** des directives d'accessibilité du **RGAA 4.1**. »

À ne pas faire : Mauvaise pratique

Utiliser des adjectifs subjectifs.

Performance : « Le site doit être rapide. » *(Rapide pour qui ? Dans quelles conditions ?)*

Sécurité : « Le système doit être sécurisé. » *(Contre quelles menaces ? Quel niveau de sécurité ?)*

Utilisabilité : « L'interface doit être intuitive. » *(Intuitif pour un expert ou un novice ?)*

Exercices de classification : EF vs ENF

Instructions : *Pour chaque exigence suivante, déterminez s'il s'agit d'une Exigence Fonctionnelle (EF) ou d'une Exigence Non-Fonctionnelle (ENF). Justifiez votre réponse en une phrase clef.*

Exercice 1

Exigence : « L'utilisateur doit pouvoir rechercher un produit en utilisant son nom ou son numéro de référence. »

Correction suggérée

Type : Exigence Fonctionnelle (EF).
Justification : Elle décrit **ce que** le système doit faire (une fonctionnalité de recherche), une action spécifique de l'utilisateur.

Exercice 2

Exigence : « La page de résultats de recherche doit s'afficher en moins de 1,5 seconde. »

Correction suggérée

Type : Exigence Non-Fonctionnelle (ENF).
Justification : Elle décrit **comment** le système doit se comporter. C'est une contrainte de performance mesurable (la vitesse).

Exercice 3

Exigence : « Si le panier d'un client dépasse 100€, une réduction de 5% doit être appliquée automatiquement. »

Correction suggérée

Type : Exigence Fonctionnelle (EF).
Justification : Elle définit une règle métier et une action du système (calculer et appliquer une réduction), c'est une fonctionnalité.

Exercice 4

Exigence : « La session d'un utilisateur doit expirer automatiquement après 15 minutes d'inactivité. »

Correction suggérée

Type : Exigence Non-Fonctionnelle (ENF).
Justification : Elle définit une contrainte de sécurité sur le comportement global du système, pas une action initiée par l'utilisateur.

Exercice 5

Exigence : « Le système doit envoyer un email de confirmation à l'utilisateur après la validation de sa commande. »

Correction suggérée

Type : Exigence Fonctionnelle (EF).
Justification : Elle décrit une sortie spécifique du système (générer et envoyer un email) en réponse à une action.

Exercice 6

Exigence : « Les contrastes de couleurs du site web doivent être conformes au standard WCAG 2.1 niveau AA. »

Correction suggérée

Type : Exigence Non-Fonctionnelle (ENF).
Justification : Elle décrit une contrainte d'accessibilité et d'utilisabilité (comment le site est présenté), et non une fonctionnalité.

Exercice 7

Exigence : « L'administrateur doit pouvoir accéder à un tableau de bord pour visualiser la liste des utilisateurs inscrits. »

Correction suggérée

Type : Exigence Fonctionnelle (EF).
Justification : Elle décrit une fonctionnalité spécifique ("voir une liste") disponible pour un rôle utilisateur particulier (administrateur).

Exercice 8

Exigence : « L'application doit être disponible 99,9% du temps en dehors des fenêtres de maintenance prévues. »

Correction suggérée

Type : Exigence Non-Fonctionnelle (ENF).
Justification : Elle définit une qualité de service mesurable, à savoir la fiabilité et la disponibilité du système.

Exercice 9

Exigence : « Toutes les nouvelles contributions au code source doivent être accompagnées de tests unitaires atteignant une couverture de 85%.»

Correction suggérée

Type : Exigence Non-Fonctionnelle (ENF).
Justification : Elle définit une contrainte sur le processus de développement pour garantir la qualité du code, c'est une exigence de maintenabilité.

Exercice 10

Exigence : « L'utilisateur doit pouvoir exporter l'historique de ses factures dans un fichier au format CSV. »

Correction suggérée

Type : Exigence Fonctionnelle (EF).
Justification : Elle décrit une action spécifique (exporter des données) que l'utilisateur peut déclencher via le système.

2 Formalisation des Exigences

Une fois les besoins recueillis et les exigences identifiées, il faut les formaliser. Autrement dit, il faut les écrire d'une manière qui soit à la fois claire pour le client (qui doit les valider) et directement exploitable par l'équipe de développement (qui doit les réaliser).

Pendant des décennies, la méthode classique consistait à rédiger des cahiers des charges de centaines de pages, décrivant le système dans ses moindres détails techniques. Ces documents massifs souffraient de défauts majeurs : ils étaient longs à rédiger, rapidement obsolètes, rarement lus en entier, et surtout, ils noyaient la finalité du projet sous une avalanche de détails techniques. L'équipe passait plus de temps à interpréter le document qu'à répondre au besoin réel.

Face à cet échec, les méthodes Agiles ont popularisé une approche radicalement différente, centrée sur l'humain et la valeur : les **User Stories**.

2.1 Les User Stories : la voix de l'utilisateur

Une User Story (ou « récit utilisateur ») est une description simple, courte et rédigée en langage naturel d'une fonctionnalité souhaitée, racontée du point de vue de la personne qui va

l'utiliser. Son but n'est pas de documenter exhaustivement une exigence, mais d'amorcer une conversation. C'est une promesse de dialogue entre le porteur du besoin et ceux qui vont le construire.

La force de la User Story est sa structure qui oblige à penser à la **valeur ajoutée**. Elle force à répondre à trois questions fondamentales sur une fonctionnalité : QUI la veut ? QUE/QUOI veut-il faire ? et POURQUOI le veut-il ?

Ce formalisme, popularisé par Rachel Davies https://rachelcdavies.github.io, suit un modèle simple mais puissant :

```
En tant que <rôle>, je veux <action> afin de <bénéfice>.
```

Décortiquons cette structure :

`En tant que <rôle>` Il s'agit du **QUI**. Ce n'est pas un « utilisateur » générique, mais un acteur précis du système (ex : « un client enregistré », « un administrateur du site », « un gestionnaire de stock »). Définir le rôle force à adopter le point de vue de cet acteur et à créer de l'empathie [techdocs.broadcom.com].

`je veux <action>` C'est le **QUOI**, l'exigence fonctionnelle elle-même. C'est le comportement attendu du système, l'objectif à atteindre (ex : « consulter mon historique de commandes », « exporter les statistiques en CSV »).

`afin de <bénéfice>` C'est le cœur de la User Story, le **POURQUOI**. Cette partie exprime la valeur, le gain ou le problème que l'on cherche à résoudre pour le rôle. C'est la justification métier de la fonctionnalité (ex : « suivre l'état de mes livraisons », « intégrer les données dans mon logiciel de comptabilité »). C'est ce qui permet à l'équipe de développement de comprendre l'intention et, si nécessaire, de proposer une solution technique encore meilleure pour atteindre ce but.

Une User Story n'est donc pas un document de spécification complet. C'est une invitation à discuter des détails, à définir les règles de gestion et les critères d'acceptation qui permettront, à la fin, de valider que le bénéfice attendu est bien atteint.

À faire : Bonne pratique

Mettre l'accent sur le bénéfice métier final pour l'utilisateur.

— « En tant que *client régulier*, je veux *sauvegarder mon adresse de livraison* afin de *ne pas avoir à la resaisir à chaque commande.* »

Pourquoi ? On comprend immédiatement le gain de temps et le confort pour l'utilisateur. L'équipe de développement peut même proposer une meilleure solution (ex : gérer plusieurs adresses).

À ne pas faire : Mauvaise pratique

Décrire une interaction avec l'interface ou une tâche technique.

— « En tant qu'utilisateur, je veux un champ de texte pour mon adresse. »

— « En tant que développeur, je veux une table 'Address' dans la base de données. »

Pourquoi ? La première est une description de l'interface, pas un besoin. La seconde est une tâche technique, pas une User Story. Aucune des deux n'explique la finalité métier.

Exercices sur les User Stories

Une user story est une description simple d'une fonctionnalité, racontée du point de vue de l'utilisateur. Le format le plus courant est :

En tant que <rôle>, je veux <objectif/action> afin de <bénéfice/résultat>.

Exercice 1

Contexte : Vous travaillez sur une application de partage de recettes de cuisine.
Tâche : Écrivez une user story pour un utilisateur qui souhaite enregistrer ses recettes préférées pour les retrouver plus tard.

Correction suggérée

En tant que utilisateur connecté, je veux pouvoir sauvegarder une recette dans une liste de "favoris" afin de la retrouver facilement sans avoir à la rechercher à nouveau..
Pourquoi c'est une bonne story :
- **Rôle :** "Utilisateur connecté" (précise qu'il faut être logué).
- **Objectif :** "Sauvegarder une recette...".
- **Bénéfice :** "La retrouver facilement...".

Exercice 2

Contexte : Un chef de projet vous présente la story suivante pour un site e-commerce :

> *« En tant que développeur, je veux utiliser une base de données PostgreSQL afin d'améliorer la structure. »*

Tâche : Identifiez au moins deux raisons pour lesquelles ceci n'est pas une bonne user story.

Correction suggérée

Cette story est mauvaise car :

1. **Le rôle n'est pas un utilisateur final.** 'En tant que développeur' ne représente pas un client.
2. **Elle décrit le "comment" et non le "quoi".** L'utilisation d'une technologie spécifique est un détail d'implémentation.
3. **Le bénéfice est flou pour l'utilisateur.** "Améliorer la structure" n'est pas un bénéfice tangible pour celui qui utilise le site.

Qu'est-ce qu'une Epic ? Dans le contexte des méthodes Agiles, une **Epic** est une User Story de très grande taille, qui ne peut pas être réalisée par une équipe de développement au cours d'une seule itération (sprint). Elle représente une fonctionnalité majeure ou un besoin métier complexe et de haut niveau.

L'objectif principal d'une Epic n'est pas d'être développée telle quelle, mais de servir de **conteneur** pour des exigences plus larges qui seront décomposées par la suite.

Epic

Une Epic est une "promesse de conversation" sur une grande fonctionnalité. Son but est d'être découpée en plusieurs **User Stories** plus petites et gérables, qui pourront alors être estimées, priorisées et développées.

Analogie courante : Si les User Stories sont les *scènes* d'un film, alors l'Epic en est l'*acte* complet.

Cette décomposition permet à l'équipe de livrer de la valeur de manière incrémentale, sans devoir attendre que l'ensemble de la fonctionnalité "Epic" soit spécifiée dans ses moindres détails.

Exercice 3

Contexte : Pour une plateforme d'e-learning, vous avez l'exigence très large (une "Epic") : « Gérer son profil utilisateur ».
Tâche : Décomposez cette Epic en au moins trois user stories plus petites et spécifiques.

Correction suggérée

- *En tant que utilisateur, je veux modifier mon mot de passe afin de sécuriser mon compte..*
- *En tant que utilisateur, je veux mettre à jour mes informations personnelles afin de que mes futures inscriptions soient correctes..*
- *En tant que utilisateur, je veux ajouter ou changer ma photo de profil afin de personnaliser mon compte..*

Exercice 4

Contexte : Vous avez la user story suivante pour un forum en ligne :

> *En tant que membre, je veux pouvoir voter pour un commentaire afin de montrer qu'il est utile..*

Tâche : Écrivez trois critères d'acceptation pour cette user story.

Correction suggérée

Critères d'acceptation :

1. **Scénario "Vote positif" :** Étant donné que je suis connecté, lorsque je clique sur la flèche vers le haut, le compteur de votes s'incrémente de 1.
2. **Scénario "Un seul vote" :** Étant donné que j'ai déjà voté, lorsque je clique à nouveau sur la flèche vers le haut, mon vote est annulé et le compteur diminue de 1.
3. **Scénario "Non-membre" :** Étant donné que je ne suis pas connecté, je ne peux pas cliquer sur la flèche de vote.

Exercice 5

Contexte : Un client demande : « Je veux pouvoir mettre des couleurs sur mes tâches. »
Tâche : Complétez la user story en proposant deux bénéfices ('afin de...') potentiels.

Correction suggérée

- **Option 1 (Priorisation) :** *En tant que utilisateur, je veux mettre des couleurs différentes sur mes tâches afin de je puisse rapidement identifier les tâches les plus urgentes..*
- **Option 2 (Catégorisation) :** *En tant que utilisateur, je veux mettre des couleurs différentes sur mes tâches afin de les regrouper visuellement par projet..*

Exercice 6

Contexte : Story proposée pour un site e-commerce :

> *En tant que client, je veux chercher un produit, l'ajouter au panier et payer par carte de crédit afin de acheter un article..*

Tâche : Expliquez pourquoi cette story est trop grosse ("Big/Small") et comment la découper.

Correction suggérée

Cette story est une Epic car elle couvre plusieurs étapes distinctes. Elle doit être découpée :

1. *En tant que client, je veux chercher un produit par son nom afin de trouver ce que je cherche..*
2. *En tant que client, je veux ajouter un produit à mon panier depuis sa page afin de le préparer pour l'achat..*
3. *En tant que client, je veux payer ma commande par carte de crédit afin de finaliser mon achat..*

Exercice 7

Contexte : Une plateforme d'e-learning veut une fonctionnalité de messagerie.
Tâche : Rédigez deux stories distinctes pour envoyer un message : une pour un **professeur**, l'autre pour un **proviseur**.

Correction suggérée

- **Professeur :** *En tant que professeur, je veux envoyer un message à tous les élèves de ma classe afin de leur communiquer des informations sur le cours..*
- **Proviseur :** *En tant que proviseur, je veux envoyer un message à tous les élèves d'une classe afin de diffuser des annonces générales de l'école..*

Exercice 8

Contexte : Story proposée pour un logiciel photo : « En tant qu'utilisateur, je veux un filtre puissant pour améliorer mes photos. »
Tâche : Expliquez pourquoi elle est ambiguë et proposez une question pour la clarifier.

Correction suggérée

Problème : Les mots "puissant" et "améliorer" ne sont pas vérifiables et sont sujets à interprétation.
Question à poser : « Pouvez-vous me décrire ce que "améliorer" signifie pour vous ? Cherchez-vous à ajuster les couleurs, la netteté, ou autre chose ? »

Exercice 9

Contexte : Story technique : « Mettre à jour l'API de paiement vers la v3 pour que le code soit plus maintenable. »
Tâche : Expliquez son problème et reformulez-la en orientant la valeur vers l'utilisateur.

Correction suggérée

Problème : La "maintenabilité" n'est pas une valeur directe pour l'utilisateur. C'est une tâche technique (enabler).
Reformulation (si la v3 le permet) : *En tant que client, je veux payer avec Apple Pay afin de avoir plus d'options et payer plus rapidement..*

Les Critères INVEST pour une User Story L'acronyme **INVEST** est un guide mnémotechnique, introduit par Bill Wake, pour s'assurer qu'une user story est bien formulée et prête pour le développement. Chaque lettre représente une qualité essentielle :

- **I – Independent (Indépendante)** La story doit être autonome et pouvoir être réalisée indépendamment des autres. Cela évite les dépendances qui bloquent le travail et permet de la prioriser librement.
- **N – Negotiable (Négociable)** La story n'est pas un contrat rigide, mais le point de départ d'une conversation entre l'équipe et les parties prenantes sur les détails de l'implémentation.
- **V – Valuable (Ayant de la valeur)** Elle doit apporter une valeur claire et tangible à l'utilisateur final ou au client. Si une story n'a pas de valeur, il faut se demander pourquoi on la réalise.
- **E – Estimable (Estimable)** L'équipe de développement doit être capable d'évaluer l'effort nécessaire à sa réalisation. Une story trop vague ou trop grosse ne peut pas être estimée.
- **S – Small (Petite / de bonne taille)** Elle doit être assez petite pour être réalisable en une seule itération (un sprint) par l'équipe. Les grosses stories (Epics) doivent être découpées.
- **T – Testable (Testable)** La story doit être vérifiable. Il doit exister des critères d'acceptation clairs et objectifs pour prouver qu'elle est terminée et fonctionne correctement.

Exercice 10

Contexte : Pour une application bancaire : « En tant que client, je veux un écran de sécurité complet. »
Tâche : Évaluez cette story avec les critères **INVEST** pour expliquer pourquoi elle n'est pas prête.

Correction suggérée

- **Independent :** Non, dépend d'autres fonctions.
- **Negotiable :** Non, trop vague pour négocier.
- **Valuable :** Oui, mais la valeur est mal définie.
- **Estimable :** Non, l'effort est impossible à chiffrer.
- **Small :** Non, c'est une Epic.
- **Testable :** Non, "complet" n'est pas un critère testable.

Conclusion : C'est une Epic à décomposer en stories plus petites (ex : changer son PIN, activer la biométrie...).

2.2 Les Cas d'Utilisation : le scénario de l'interaction

Si la User Story est la voix de l'utilisateur exprimant un besoin et une valeur, le Cas d'Utilisation (Use Case) est le script détaillé de la conversation entre cet utilisateur et le système. C'est une technique de spécification plus formelle, issue du standard UML (Unified Modeling Language), qui vise à décrire de manière exhaustive toutes les interactions possibles pour atteindre un objectif.

2.2.1 Rappel sur la démarche UML

Dans la démarche UML, tout part d'une vue d'ensemble du système.

1. **Identifier les Acteurs et les Cas d'Utilisation :** On commence par identifier les « acteurs » (les rôles, humains ou autres systèmes, qui interagissent avec notre logiciel) et les objectifs principaux qu'ils veulent atteindre. Chaque objectif devient un cas d'utilisation.
2. **Créer le Diagramme de Cas d'Utilisation :** On représente ensuite ces éléments sur un **diagramme de cas d'utilisation**. Ce diagramme est une carte de haut niveau des fonctionnalités du système. Il montre *qui* peut faire *quoi*, mais pas *comment*. Seul, ce diagramme est insuffisant. C'est une simple table des matières.
3. **Rédiger les Spécifications Détaillées :** C'est ici qu'intervient la véritable analyse. Pour chaque bulle (cas d'utilisation) du diagramme, on rédige un document textuel qui décrit pas à pas le déroulement des interactions. C'est cette **spécification textuelle** qui donne toute sa puissance à la méthode.
4. **Réaliser les diagrammes de conception :** Forte de cette description textuelle précise, l'équipe technique peut alors modéliser les interactions plus finement à l'aide de *diagrammes de séquence* et concevoir la structure du code avec des *diagrammes de classes*.

Le Cas d'Utilisation est donc une passerelle fondamentale entre le besoin fonctionnel de haut niveau et la conception technique détaillée.

2.2.2 La structure d'un Cas d'Utilisation

La force du Cas d'Utilisation réside dans sa description textuelle qui détaille non seulement le *« happy path »* (le scénario nominal où tout se passe comme prévu), mais aussi et surtout tous les cas alternatifs et les scénarios d'erreurs possibles. Oublier ces scénarios d'exception est une erreur classique qui mène à des applications fragiles et imprévisibles.

Un cas d'utilisation formel contient plusieurs sections (Nom, Acteurs, Préconditions, etc.), mais son cœur est la description de ses flux d'événements.

Prenons l'exemple simplifié du cas d'utilisation « Payer sa commande ».

Cas d'Utilisation : Payer sa commande

Acteur Primaire : Le Client.
Précondition : Le client a validé son panier et se trouve sur la page de résumé de commande.

Scénario Nominal (ou "Happy Path")

1. Le Client clique sur le bouton « Procéder au paiement ».
2. Le Système affiche un formulaire sécurisé pour la saisie des informations de carte bancaire.
3. Le Client saisit les informations de sa carte (numéro, date d'expiration, CVV) et valide.
4. Le Système transmet une demande d'autorisation au système externe de paiement.
5. Le système externe de paiement **autorise** la transaction.
6. Le Système enregistre la commande avec le statut « Payée », déclenche l'envoi d'un email de confirmation au Client et affiche une page de succès avec le numéro de commande.

Scénarios Alternatifs et d'Exception

5a. *Le système externe de paiement* ***refuse*** *la transaction (fonds insuffisants, carte expirée...).*

6a. Le Système affiche un message d'erreur clair informant le Client du refus et l'invite à utiliser un autre moyen de paiement ou à vérifier ses informations.

7a. Le Système retourne le Client à l'étape 3. Le cas d'utilisation reprend son cours.

4b. *Le système externe de paiement est indisponible (erreur de communication réseau).*

5b. Le Système informe le Client de l'incident technique et lui propose de réessayer ultérieurement. La commande est conservée avec un statut « En attente de paiement ».

Postcondition de succès : La commande est marquée comme "Payée".
Postcondition d'échec : La commande reste dans son état initial ou passe en "Échec de paiement".

Cette approche structurée force l'analyste à anticiper les problèmes et à spécifier un comportement robuste, réduisant ainsi les zones d'ombre pour l'équipe de développement et garantissant une meilleure expérience utilisateur, même lorsque les choses tournent mal.

Exercices sur les Cas d'Utilisation (Use Cases)

Exercice 1

Titre : Identifier les Acteurs et les Cas d'Utilisation
Contexte : Vous êtes analyste pour un nouveau système de gestion de bibliothèque. Les membres peuvent rechercher des livres, vérifier leur disponibilité, et les emprunter. Les bibliothécaires, eux, peuvent enregistrer le retour des livres et ajouter de nouveaux livres au catalogue.
Tâche : Identifiez les acteurs (humains) et listez au moins quatre cas d'utilisation principaux pour ce système.

Correction suggérée

- **Acteurs :**
  - Le Membre
  - Le Bibliothécaire
- **Cas d'Utilisation possibles :**
  - Rechercher un livre
  - Emprunter un livre
  - Enregistrer le retour d'un livre
  - Ajouter un nouveau livre au catalogue

Exercice 2

Titre : Rédiger le Scénario Nominal ("Happy Path")
Contexte : En se basant sur la démarche UML, vous devez détailler un cas d'utilisation.
Tâche : Rédigez le scénario nominal (flux principal où tout se passe bien) pour le cas d'utilisation **« Se connecter au système »**. L'acteur primaire est l'« Utilisateur ». La précondition est que l'utilisateur se trouve sur la page d'accueil.

Correction suggérée

Cas d'Utilisation : Se connecter au système
Acteur Primaire : L'Utilisateur
Précondition : L'utilisateur est sur la page d'accueil du site.
Scénario Nominal :

1. L'Utilisateur clique sur le bouton « Se connecter ».
2. Le Système affiche un formulaire avec les champs "Email" et "Mot de passe".
3. L'Utilisateur saisit son email et son mot de passe corrects, puis valide le formulaire.
4. Le Système vérifie les informations en base de données.
5. Les informations sont valides. Le Système ouvre une session pour l'Utilisateur.
6. Le Système redirige l'Utilisateur vers son tableau de bord personnel et affiche un message de bienvenue.

Postcondition de succès : L'Utilisateur est authentifié et a une session active.

Exercice 3

Titre : Anticiper les Scénarios d'Exception
Contexte : Un cas d'utilisation n'est complet que s'il anticipe ce qui peut mal se passer.
Tâche : En reprenant le scénario nominal de l'exercice 2 (« Se connecter au système »), identifiez deux scénarios alternatifs ou d'exception possibles. Pour chaque scénario, décrivez brièvement la séquence d'actions (par ex : 4a, 5a...).

Correction suggérée

Scénarios Alternatifs et d'Exception pour « Se connecter au système »
Flux Alternatif A : Le mot de passe ou l'email est incorrect

4a. Le Système vérifie les informations en base de données et constate que la combinaison email/mot de passe est invalide.
5a. Le Système ré-affiche le formulaire de connexion (étape 2) avec un message d'erreur clair : « L'adresse email ou le mot de passe est incorrect. ». Le cas d'utilisation reprend à l'étape 3.

Flux d'Exception B : Compte utilisateur bloqué

4b. Le Système vérifie les informations et constate que le compte associé à l'email est bloqué (suite à trop de tentatives infructueuses, par exemple).
5b. Le Système affiche une page informant l'Utilisateur que son compte est temporairement verrouillé et lui propose un lien pour suivre la procédure de déblocage (ex : réinitialiser le mot de passe). Le cas d'utilisation se termine.

Exercice 4

Titre : Compléter un Cas d'Utilisation
Contexte : Vous recevez une ébauche de cas d'utilisation incomplète de la part d'un collègue.
Tâche : Complétez les sections manquantes (marquées par [À COMPLÉTER]) du cas d'utilisation suivant.

Cas d'Utilisation : Réinitialiser son mot de passe

Acteur Primaire : [À COMPLÉTER]

Précondition : [À COMPLÉTER]

Scénario Nominal :
1. L'Utilisateur clique sur le lien « Mot de passe oublié ? ».
2. Le Système affiche un formulaire demandant l'adresse email de l'Utilisateur.
3. L'Utilisateur saisit son adresse email et valide.
4. Le Système génère un lien de réinitialisation unique et sécurisé.
5. Le Système envoie ce lien à l'adresse email saisie.
6. Le Système affiche une page confirmant que l'email a été envoyé.

Scénarios Alternatifs : [À COMPLÉTER AVEC UN EXEMPLE]

Postcondition de succès : [À COMPLÉTER]

Postcondition d'échec : [À COMPLÉTER]

Correction suggérée

Cas d'Utilisation : Réinitialiser son mot de passe

Acteur Primaire : L'Utilisateur (non authentifié)

Précondition : L'Utilisateur se trouve sur la page de connexion.

Scénario Nominal :
1. L'Utilisateur clique sur le lien « Mot de passe oublié ? ».
2. Le Système affiche un formulaire demandant l'adresse email de l'Utilisateur.
3. L'Utilisateur saisit son adresse email et valide.
4. Le Système génère un lien de réinitialisation unique et sécurisé.
5. Le Système envoie ce lien à l'adresse email saisie.
6. Le Système affiche une page confirmant que l'email a été envoyé.

Scénarios Alternatifs : Flux Alternatif A : L'adresse email n'est pas trouvée dans le système

3a. L'Utilisateur saisit une adresse email qui n'est associée à aucun compte.

4a. Pour des raisons de sécurité (afin de ne pas confirmer l'existence ou non d'un utilisateur), le Système poursuit le flux en apparence, mais n'envoie aucun email.

5a. Le Système affiche la même page de confirmation que dans le scénario nominal (étape 6). Le cas d'utilisation se termine.

Postcondition de succès : Un email contenant un lien de réinitialisation a été envoyé à l'adresse fournie. L'ancien mot de passe est toujours actif.

Postcondition d'échec : Aucun email n'est envoyé et l'état du compte de l'utilisateur reste inchangé.

3 Un exemple complet des exigences aux cas d'usage en passant par les users stories

3.1 Le Besoin Initial : Les Exigences Métier

L'entreprise « Alpha Corp » souhaite améliorer l'engagement de ses collaborateurs et encourager l'innovation interne. Elle veut mettre en place une plateforme web simple, **InnovateBox**, où les employés peuvent soumettre des idées et où les managers peuvent les consulter.

Exigences de Haut Niveau

- **REQ-01 :** Les employés doivent pouvoir soumettre de nouvelles idées de manière anonyme ou non.
- **REQ-02 :** Les managers doivent pouvoir consulter la liste de toutes les idées soumises.
- **REQ-03 :** Le système doit envoyer une notification au manager lorsqu'une nouvelle idée est postée.
- **REQ-04 :** Toutes les communications avec le serveur doivent être sécurisées (HTTPS).

Note : *REQ-01, REQ-02, REQ-03 sont des exigences fonctionnelles (EF). REQ-04 est une exigence non-fonctionnelle (ENF) de sécurité.*

3.2 Approche Agile : Les User Stories

Dans une approche Agile, l'analyste se concentre sur la valeur apportée aux utilisateurs. Il transforme les exigences en User Stories.

3.2.1 Création des User Stories

À partir de **REQ-01** et **REQ-02**, on peut dériver les stories suivantes :

User Story pour l'Employé (REQ-01)

En tant que employé, je veux soumettre une idée avec un titre et une description afin de partager une suggestion d'amélioration pour l'entreprise..

User Story pour le Manager (REQ-02)

En tant que manager, je veux consulter la liste de toutes les idées soumises, triées de la plus récente à la plus ancienne afin de avoir une vue d'ensemble des innovations proposées..

3.2.2 Détail d'une User Story avec ses Critères d'Acceptation

Pour que la story de l'employé soit "prête" au développement, on y ajoute des critères d'acceptation.

User Story : *En tant que employé, je veux soumettre une idée avec un titre et une description afin de partager une suggestion d'amélioration..*

Critères d'Acceptation :

1. **Scénario "Soumission réussie" :** Étant donné que je suis connecté en tant qu'employé, lorsque je remplis le titre et la description et que je clique sur "Soumettre", mon idée apparaît en haut de la liste des idées et je vois un message de succès.
2. **Scénario "Champs obligatoires" :** Étant donné que je suis sur le formulaire de soumission, si je clique sur "Soumettre" sans remplir le titre ou la description, un message d'erreur s'affiche sous le champ manquant et l'idée n'est pas envoyée.
3. **Scénario "Option d'anonymat" :** Étant donné que je suis sur le formulaire, lorsque je coche la case "Soumettre anonymement", mon nom n'est pas affiché à côté de l'idée dans la liste publique.

3.3 Approche Formelle : Les Cas d'Utilisation (UML)

Dans une approche plus formelle (souvent basée sur UML), l'analyse cherche à décrire exhaustivement les interactions.

3.3.1 Le Diagramme de Cas d'Utilisation

Le diagramme sert de "table des matières" des fonctionnalités du système.

Diagramme de Cas d'Utilisation pour "InnovateBox"

Acteurs :
- Employé
- Manager (est aussi un Employé)
- Système de Notification (acteur secondaire/externe)

Cas d'Utilisation :
- **CU-01 :** Soumettre une idée (concerne REQ-01)
- **CU-02 :** Consulter les idées (concerne REQ-02)
- **CU-03 :** Envoyer une notification (concerne REQ-03)

(Dans un vrai diagramme, le Manager hériterait de l'Employé et le cas "Consulter les idées" lui serait réservé, tandis que le cas "Envoyer une notification" serait déclenché par "Soumettre une idée" et interagirait avec le Système de Notification.)

3.3.2 La Description Textuelle Détaillée d'un Cas d'Utilisation

C'est ici que l'on détaille le "script" de l'interaction pour **CU-01**.

Cas d'Utilisation : CU-01 – Soumettre une idée

Acteur Primaire : Employé

Précondition : L'Employé est authentifié sur la plateforme InnovateBox.

Scénario Nominal (Happy Path)
1. L'Employé clique sur le bouton « Proposer une idée ».
2. Le Système affiche un formulaire contenant les champs "Titre", "Description" et une case à cocher "Soumettre anonymement".
3. L'Employé saisit un titre et une description. Il laisse la case "Soumettre anonymement" décochée.
4. L'Employé clique sur le bouton « Valider ».
5. Le Système valide les données (champs non vides).
6. Le Système enregistre l'idée en base de données, en l'associant au nom de l'Employé.
7. Le Système déclenche le cas d'utilisation CU-03 "Envoyer une notification".
8. Le Système affiche une page de confirmation avec le message « Votre idée a été soumise avec succès ! » et redirige vers la liste des idées.

Scénarios Alternatifs et d'Exception

5a. Données invalides : Si à l'étape 5, le titre ou la description est vide, le Système ré-affiche le formulaire (étape 2) avec un message d'erreur indiquant quels champs sont requis. Le cas d'utilisation reprend à l'étape 3.

3b. Soumission anonyme : À l'étape 3, l'Employé coche la case "Soumettre anonymement". À l'étape 6, le Système enregistre l'idée sans la lier à l'Employé. Le reste du flux est identique.

6c. Erreur de base de données : À l'étape 6, si l'enregistrement échoue pour une raison technique, le Système affiche une page d'erreur informant l'utilisateur : « Un problème technique est survenu. Veuillez réessayer plus tard. ». Le cas d'utilisation se termine.

Postcondition de succès : Une nouvelle idée est enregistrée dans le système et est visible par les managers.

Postcondition d'échec : Aucune idée n'est enregistrée. L'utilisateur a été informé de l'échec.

3.4 Synthèse & Comparaison

Pour la même fonctionnalité « Soumettre une idée », les deux artefacts produits sont différents mais complémentaires.

Caractéristique	User Story	Cas d'Utilisation
Objectif Principal	Expliquer **POURQUOI** la fonctionnalité est nécessaire (la valeur métier) et **POUR QUI**.	Décrire exhaustivement **COMMENT** l'utilisateur et le système interagissent pour accomplir une tâche.
Format	Phrase courte et informelle : *« En tant que..., je veux..., afin de... »*	Document textuel structuré et formel (préconditions, scénarios, postconditions).
Niveau de détail	Léger. Vise à être une "promesse de conversation". Les détails sont dans les critères d'acceptation et les discussions.	Élevé et exhaustif. Décrit le "happy path" ainsi que tous les scénarios alternatifs et d'erreurs prévisibles.
Force principale	Garde l'équipe concentrée sur la **valeur utilisateur** et la priorisation. Favorise la flexibilité.	Réduit l'ambiguïté pour les développeurs et testeurs. Force l'analyse à **anticiper les problèmes** et les cas limites.
Quand l'utiliser ?	Idéal pour le backlog de produit, la planification de sprint et la communication avec les parties prenantes (approche Agile).	Idéal pour spécifier des processus complexes, critiques ou réglementés où l'exhaustivité est primordiale (approche formelle).

Exercices d'Application sur "InnovateBox"

Les exercices suivants vous invitent à étendre la plateforme "InnovateBox" en spécifiant de nouvelles fonctionnalités.

Exercice 1 : Écrire une nouvelle User Story

Contexte : Pour rendre la plateforme plus interactive, la direction souhaite que les employés puissent donner leur avis sur les idées soumises par leurs collègues. La fonctionnalité la plus simple à implémenter serait un système de vote.
Tâche : Écrivez la User Story du point de vue d'un employé qui souhaite voter pour une idée afin de montrer son soutien.
(Voir correction 10 page 48)

Exercice 2 : Identifier les éléments d'un Cas d'Utilisation

Contexte : La User Story de l'exercice 1 doit maintenant être formalisée pour l'analyse. Vous allez préparer la structure du cas d'utilisation correspondant.
Tâche : Pour le Cas d'Utilisation « **Voter pour une idée** », identifiez les éléments suivants :

- L'acteur primaire
- Une précondition essentielle
- La postcondition en cas de succès
- Une postcondition en cas d'échec

(Voir correction 10 page 48)

Exercice 3 : Rédiger les Scénarios d'Interaction

Contexte : Il est temps de détailler le "script" de l'interaction pour le vote.
Tâche : Pour le Cas d'Utilisation « **Voter pour une idée** », rédigez :

1. Le scénario nominal complet (le "happy path").
2. Au moins un scénario d'exception ou alternatif (par exemple, que se passe-t-il si l'employé a déjà voté ?).

(Voir correction 10 page 48)

Exercice 4 : Décomposer une Epic

Contexte : Les managers trouvent que voir la liste des idées ne suffit pas. Ils ont besoin d'outils pour les traiter. L'exigence globale (l'Epic) est la suivante : « En tant que manager, je veux gérer le cycle de vie des idées. »
Tâche : Décomposez cette Epic en au moins trois User Stories plus petites, indépendantes et spécifiques. Chaque story doit représenter une action de gestion distincte. *(Voir correction 10 page 48)*

4 De l'Analyse à la Conception : Modélisation Objet

Jusqu'à présent, notre démarche s'est concentrée sur la capture du besoin et sa formalisation à travers des récits (User Stories) et des scénarios (Cas d'Utilisation). Nous avons défini *ce que* le système doit faire et *comment* un utilisateur interagit avec lui pour atteindre ses objectifs. Nous nous sommes placés dans le monde de l'utilisateur, en parlant son langage.

Il est maintenant temps d'opérer une transition fondamentale : passer du **domaine du problème** (les besoins métier) au **domaine de la solution** (la structure du logiciel). C'est le passage de l'analyse à la conception. Pour construire ce pont, nous avons besoin d'un plan, d'un schéma directeur qui représente les pièces fondamentales de notre système. Ce plan, en programmation orientée objet, est le **Diagramme de Classes**.

Cependant, il existe un piège majeur à ce stade : celui de penser immédiatement en termes de code, de bases de données et d'interfaces techniques. C'est prématuré et dangereux. Avant de décider *comment* coder une 'Facture' ou *comment* la stocker en base de données, nous devons d'abord nous accorder sur ce qu'**est** une 'Facture' dans le contexte métier du client.

4.1 Le Diagramme de Classes du Domaine : Un Dictionnaire Visuel du Métier

L'objectif de cette première étape de modélisation n'est pas de concevoir le logiciel, mais de **modéliser le domaine métier lui-même**. Nous allons créer ce qu'on appelle un **Diagramme de Classes du Domaine** (ou Modèle Conceptuel de Données). Il est considéré comme le diagramme le plus important de la modélisation orientée objet.

Considérez ce diagramme non pas comme le plan d'une maison, mais comme un **dictionnaire visuel validé par le client**. Ce dictionnaire définit les concepts importants du métier, leurs propriétés et les relations qu'ils entretiennent les uns avec les autres. C'est une représentation abstraite des objets du système qui vont interagir pour réaliser les cas d'utilisation [processon.io].

- **Ce qu'il contient :**
  - Des **Classes Conceptuelles :** Les « choses » importantes du domaine métier. On les trouve souvent en identifiant les noms dans les User Stories et les Cas d'Utilisation. Exemples : 'Client', 'Produit', 'Commande', 'Facture'.
  - Des **Attributs :** Les propriétés pertinentes de ces concepts. Un 'Produit' a un 'nom' et un 'prix'. Un 'Client' a une 'adresse_de_livraison'. Il ne s'agit pas de lister toutes les données possibles, mais uniquement celles qui ont un sens pour le métier à ce stade.

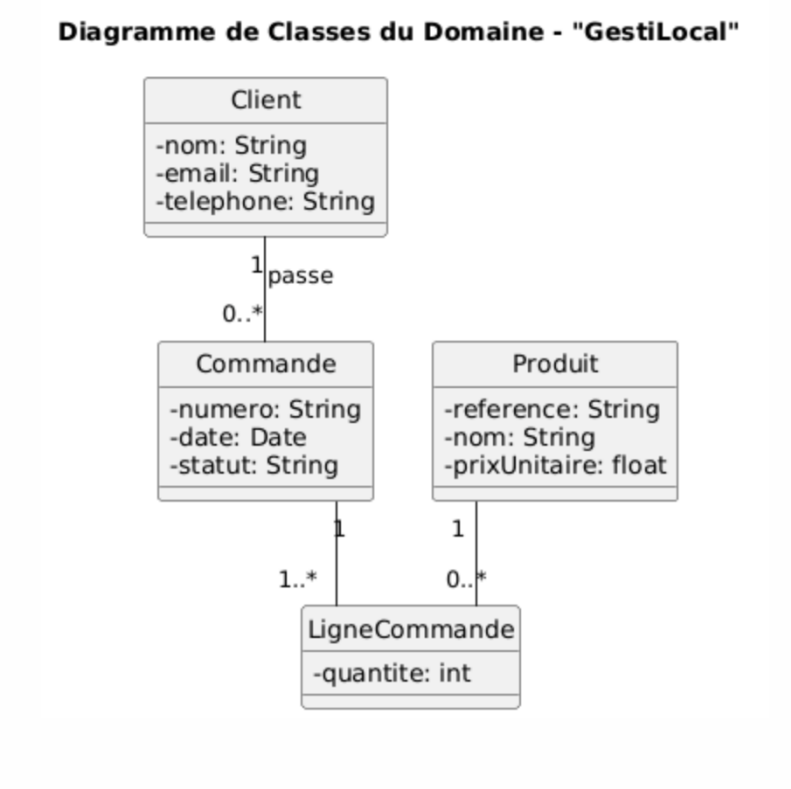
    - Des **Associations :** Les relations et les verbes d'action qui lient ces concepts entre eux. Un 'Client' *passe* une ou plusieurs 'Commandes'. Une 'Commande' *contient* plusieurs 'Produits'. Ces associations sont cruciales car elles formalisent les règles de gestion.
- **Ce qu'il exclut volontairement :**
    - Les **Méthodes/Opérations :** On ne se préoccupe pas encore des actions comme 'calculerTotal()' ou 'sauvegarderEnBase()'. Ces questions relèvent de la conception, du *comment*. Pour l'instant, on se concentre sur le *quoi*.
    - Les **Types de données techniques :** On n'indique pas si le 'prix' est un 'double', un 'float' ou un 'Decimal'. On écrit simplement « prix ». La représentation technique sera décidée plus tard.
    - Les **Classes techniques ou de "solution" :** On ne modélise pas encore des classes comme 'DatabaseManager', 'PDFController' ou 'InterfaceUtilisateur'. Ce sont des artefacts de la solution logicielle, pas des concepts du domaine métier [cs.sjsu.edu].

L'objectif final de ce diagramme de classes du domaine est triple :

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, où ce diagramme conceptuel sera enrichi pour devenir un véritable diagramme de classes de conception, prêt à être implémenté.

À faire : Bonne pratique

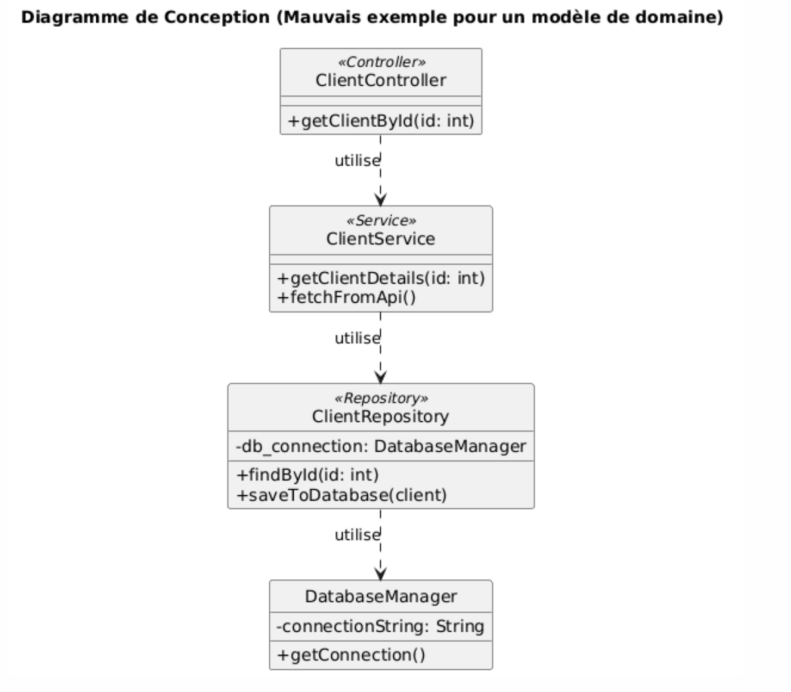
Modéliser les concepts du monde réel et leurs relations. Les noms de classe sont des substantifs, les attributs sont simples.



Classes : `Client`, `Commande`, `Produit`.
Attributs : Pour `Client` : `nom`, `email`. Pour `Produit` : `nom`, `prix`.

À ne pas faire : Mauvaise pratique

Inclure des détails d'implémentation technique.



Classes : `ClientController`, `DatabaseManager`, `ArrayList<Product>`.
Méthodes : `saveToDatabase()`, `fetchFromApi()`.
Pourquoi ? C'est un diagramme de *conception* ou d'*architecture*, pas un diagramme de domaine. Il répond à la question « comment ? », alors que l'analyse doit répondre à la question « quoi ? ».

4.2 Comment Construire le Diagramme de Classes du Domaine ?

Le passage des récits utilisateur (User Stories, Cas d'Utilisation) à un modèle structuré peut sembler abstrait, mais il suit une méthode quasi-systématique. La technique consiste à "lire entre les lignes" de nos spécifications fonctionnelles pour en extraire la structure sous-jacente du métier.

Méthodologie en 3 Étapes Clés

1. **Identifier les classes candidates :** Cherchez les **noms** et groupes nominaux importants dans vos exigences. Chaque nom représente un concept potentiel, une "chose" que le système doit gérer.
2. **Identifier les attributs :** Pour chaque classe candidate, cherchez les propriétés qui la décrivent. Ce sont souvent des noms qui dépendent d'une classe plus importante (ex : le "titre" d'une "Idée").
3. **Identifier les associations :** Cherchez les **verbes** qui lient les classes candidates entre elles. Ces verbes décrivent les relations et les actions métier (ex : un "Employé" *soumet* une "Idée").

Une fois cette première ébauche réalisée, un travail de raffinement est nécessaire : éliminer les doublons, clarifier les concepts et, surtout, déterminer les **multiplicités** (aussi appelées cardinalités) de chaque association. Pour cela, on se pose des questions simples, validées avec le client :

— Un 'Employé' peut-il soumettre *plusieurs* 'Idées' ? (Oui, donc '1' du côté de l'Employé est

lié à '0..*' du côté de l'Idée).
— Une 'Idée' peut-elle être soumise par *plusieurs* 'Employés' ? (Non, donc '1' du côté de l'Idée est lié à '1' du côté de l'Employé).

4.2.1 Application Pratique : Modélisation du Domaine "InnovateBox"

Appliquons cette méthode sur les exigences de notre application "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 est : 'Employé', 'Idée', 'Titre', 'Description', 'Manager', 'Vote'.

Étape 2 : Raffinage et Identification des Attributs Analysons cette liste :
— 'Employé' et 'Idée' sont clairement des concepts centraux. Ce sont de bonnes classes.
— 'Titre' et 'Description' ne sont pas des concepts autonomes ; ce sont des propriétés d'une 'Idée'. Ils deviennent donc des **attributs** de la classe `Idée`.
— Un 'Manager' est-il totalement différent d'un 'Employé' ? Dans notre contexte, un Manager *est un* Employé avec des droits supplémentaires. C'est un cas de **généralisation/spécialisation (héritage)**. `Manager` hérite de `Employé`.
— Le 'Vote' est-il un simple nombre (un attribut de 'Idée') ou plus ? Puisqu'un employé ne peut voter qu'une seule fois pour une idée, le système doit savoir *qui* a voté pour *quoi*. Le vote est donc une relation entre un 'Employé' et une 'Idée'. C'est une excellente candidate pour devenir une **classe-association**. La classe `Vote` permettra de mémoriser cette connexion unique.
Classes retenues : `Employé`, `Manager`, `Idée`, `Vote`.
Attributs identifiés : Pour `Idée` : 'titre', 'description'. Pour `Employé` : 'nom' (implicite).

Étape 3 : Identification des Associations et Multiplicités Nous analysons les verbes liant nos classes :
— Un 'Employé' *soumet* une 'Idée'.
  — Une 'Idée' est soumise par **exactement 1** 'Employé' (son auteur).
  — Un 'Employé' peut soumettre **0 ou plusieurs** 'Idées'.
  — Association : `Employé` 1 $\longleftrightarrow$ 0..* `Idée`.
— Un 'Employé' *vote pour* une 'Idée'. C'est ici que notre classe `Vote` intervient.
  — Un 'Vote' est émis par **exactement 1** 'Employé'.
  — Un 'Vote' concerne **exactement 1** 'Idée'.
  — Un 'Employé' peut émettre **0 ou plusieurs** 'Votes' (mais un seul par idée, règle gérée par la logique métier).
  — Une 'Idée' peut recevoir **0 ou plusieurs** 'Votes'.
  — Associations : `Employé` 1 $\longleftrightarrow$ 0..* `Vote` ET `Idée` 1 $\longleftrightarrow$ 0..* `Vote`.

Résultat : Le Diagramme de Classes du Domaine "InnovateBox" En assemblant toutes ces informations, nous obtenons un premier modèle stable du métier.

Diagramme de Classes du Domaine pour "InnovateBox"

Ce diagramme visualiserait les éléments suivants :

- Une classe `Employé` avec un attribut ‘nom‘.
- Une classe `Manager` qui hérite de `Employé` (représentée par une flèche avec un triangle vide pointant vers `Employé`).
- Une classe `Idée` avec les attributs ‘titre‘ et ‘description‘.
- Une classe `Vote` (qui pourrait avoir un attribut ‘dateDeVote‘).
- Une association "estAuteurDe" entre `Employé` et `Idée` avec les multiplicités **1** côté Employé et **0..*** côté Idée.
- La classe `Vote` est reliée à la fois à `Employé` (**1** côté Employé, **0..*** côté Vote) et à `Idée` (**1** côté Idée, **0..*** côté Vote).

Ce diagramme est notre dictionnaire visuel. Il est simple, non-technique, et peut être présenté au client pour valider que nous avons bien compris les concepts clés de son activité et leurs règles de gestion avant d’écrire la moindre ligne de code. C’est la fondation sur laquelle la conception technique pourra s’appuyer solidement.

Exercices : De la User Story au Diagramme de Classes

Les exercices suivants vous demandent de mettre en pratique la méthodologie d'extraction de concepts pour construire un Diagramme de Classes du Domaine à partir d'exigences (User Stories, Cas d'Usage).

Exercice 1 : Concepts de Base et Attributs

Contexte : Un blog simple.
User Story :

> En tant que **rédacteur**, je veux créer un nouvel **article** avec un **titre**, un **contenu** et une **date de publication** pour partager des informations avec mes lecteurs.

Tâche : Construisez le Diagramme de Classes du Domaine correspondant à cette User Story. Identifiez les classes et leurs attributs principaux.

(Voir solution 10 page 49)

Exercice 2 : Ajout d'une Association Simple

Contexte : On enrichit le blog de l'exercice 1.
Cas d'Usage : "Commenter un article"

- **Acteur :** Visiteur
- **Description :** Le **visiteur** lit un **article**. Il saisit son **nom** et le texte de son **commentaire** dans un formulaire, puis valide. Le système affiche le nouveau commentaire sous l'article.

Tâche : Modifiez le diagramme précédent pour intégrer la notion de commentaire. Identifiez la nouvelle classe, ses attributs, et l'association qui la lie aux classes existantes (précisez les multiplicités).

(Voir solution 10 page 49)

Exercice 3 : Héritage (Généralisation/Spécialisation)

Contexte : Un système de gestion de forums.
User Stories :

1. En tant qu'**utilisateur**, je veux avoir un **pseudo** et un **mot de passe** pour me connecter.
2. En tant qu'**abonné**, je veux pouvoir créer une nouvelle **discussion**.
3. En tant que **modérateur**, je veux pouvoir **supprimer** un message inapproprié dans n'importe quelle discussion.

Tâche : Modélisez la structure des utilisateurs. Utilisez le concept d'héritage pour représenter la relation entre un utilisateur de base et les types plus spécifiques (abonné, modérateur).

(Voir solution 10 page 49)

Exercice 4 : La Classe-Association

Contexte : Un outil interne de gestion de projets.
Cas d'Usage : "Assigner un employé à un projet"

- **Acteur :** Chef de projet
- **Description :** Le **chef de projet** sélectionne un **employé** et un **projet**. Il définit ensuite le **rôle** de cet employé sur ce projet spécifique (ex : "Développeur", "Testeur") ainsi que la **date de début** de l'assignation.

Tâche : Modélisez la relation entre un `Employé` et un `Projet`. Justifiez l'utilisation d'une classe-association et précisez les attributs qu'elle contiendrait. N'oubliez pas les multiplicités.

(Voir solution 10 page 49)

Exercice 1 : Modélisation d'un Site E-commerce

Contexte : La base d'un site de vente en ligne.
User Stories :

1. En tant que **client**, je veux consulter les **produits** disponibles, avec leur **nom**, leur **description** et leur **prix**.
2. En tant que **client**, je veux ajouter plusieurs **produits**, en spécifiant une **quantité** pour chacun, à mon **panier**.
3. En tant que **client**, je veux passer une **commande** à partir de mon panier pour finaliser mon achat. Chaque commande a une **date** et une **adresse de livraison**.

Tâche : Proposez un Diagramme de Classes du Domaine complet pour ce système. Identifiez toutes les classes, leurs attributs, et toutes les associations avec leurs multiplicités. Portez une attention particulière à la manière de modéliser le contenu d'une commande.

(Voir solution 10 page 50)

Exercice 2 : Modélisation d'un Système de Réservation d'Événements

Contexte : Une plateforme pour acheter des billets de concerts.
User Stories :

1. En tant qu'**organisateur**, je veux créer un **événement** en précisant son **nom**, le **lieu**, la **date** et le nombre total de **places** disponibles.
2. En tant que **client**, je veux pouvoir faire une **réservation** pour un **événement** pour une ou plusieurs personnes.
3. En tant que **client**, suite à ma réservation, je veux recevoir un ou plusieurs **billets**, chacun avec un **code unique**.

Tâche : Construisez le Diagramme de Classes du Domaine pour ce système. Pensez à bien distinguer les concepts (événement, réservation, billet) et à modéliser leurs relations avec les bonnes multiplicités.

(Voir solution 10 page 50)

Exercice 3 : Modélisation d'une Plateforme pour Freelances

Contexte : Une plateforme mettant en relation des clients et des freelances pour des missions.

Exigences :

- Un **client** peut poster plusieurs **missions**. Chaque mission a un **titre** et une **description**.
- Un **freelance** peut postuler à des missions.
- Quand un accord est trouvé, un **contrat** est généré. Le contrat lie un **freelance**, une **mission**, et contient le **montant total** de la prestation et les **délais**.
- À la fin de la mission, le freelance peut générer une **facture** basée sur le contrat. La facture a un **numéro** et une **date d'émission**.

Tâche : Élaborez le Diagramme de Classes du Domaine pour cette plateforme. Il s'agit d'un cas plus complexe qui combine plusieurs patrons de modélisation.

(Voir solution 10 page 50)

5 Au-delà de la Structure : La Modélisation des Processus Métiers

Jusqu'à présent, avec le Diagramme de Classes du Domaine, nous avons créé une carte des concepts de l'entreprise. C'est une **vue statique** : elle nous dit ce que sont un `Employé` et une `Idée`, et qu'ils sont liés. Cependant, elle ne nous dit pas *comment* une idée passe de "Soumise" à "Approuvée", ni qui intervient et dans quel ordre.

Pour capturer cette dimension temporelle et comportementale, nous devons modéliser les **processus métiers**. Un processus métier est une séquence d'activités ou de tâches qui, lorsqu'elles sont exécutées dans un ordre spécifique, accomplissent un objectif métier. C'est la **vue dynamique** du système.

5.1 Comment Identifier les Processus Métiers ?

Identifier les processus est un travail d'enquête qui complète celui de la création des User Stories. Les processus se cachent souvent derrière les verbes et les objectifs métier. Pour les trouver, il faut se poser les bonnes questions :

- **Quels sont les grands événements ?** Qu'est-ce qui déclenche une série d'actions dans l'entreprise ? (Ex : "une nouvelle idée est soumise", "la fin du mois arrive", "un client passe une commande").
- **Quel est le cycle de vie d'un objet clé ?** Par quelles étapes passe une `Idée` de sa création à sa réalisation ou son rejet ? Comment une `Commande` passe-t-elle de "reçue" à "livrée" ?
- **Qui fait quoi, et quand ?** En interrogeant les utilisateurs, on peut leur demander de raconter une "journée type" ou de décrire comment ils accomplissent une tâche complexe du début à la fin.

Les Cas d'Utilisation sont une excellente source : un cas complexe comme "Gérer une commande" cache souvent un processus métier complet.

5.2 Comment Décrire les Processus Métiers ?

Une fois un processus identifié (ex : "Valider une nouvelle idée"), il faut le décrire de manière claire. Plusieurs niveaux de formalisme existent.

1. **Le Récit Textuel Structuré :** La méthode la plus simple et souvent suffisante. On liste les étapes sous forme de phrases numérotées, en précisant l'acteur pour chaque étape.
2. **Le Diagramme de Flux (Flowchart) :** Une représentation visuelle simple avec des symboles de base (ovales pour le début/fin, rectangles pour les actions, losanges pour les décisions). Très facile à comprendre pour un public non technique.
3. **BPMN (Business Process Model and Notation) :** La norme de l'industrie pour la modélisation de processus. BPMN est beaucoup plus riche et précis qu'un simple diagramme de flux. Il permet de modéliser des interactions complexes entre différents services (piscines et couloirs), des événements temporels, des exceptions, des transactions, etc. C'est l'outil de choix pour les processus critiques ou complexes.

Conseil : Commencer simple

Pour la plupart des besoins d'analyse, un **récit textuel structuré** est un excellent point de départ. Il est rapide à produire, facile à discuter avec les experts métier et contient déjà 90% de l'information nécessaire pour l'étape suivante.

5.3 La Synergie : Comment les Processus Enrichissent le Modèle de Domaine

C'est ici que la magie opère. La modélisation statique (classes) et la modélisation dynamique (processus) ne sont pas deux activités séparées ; elles se nourrissent l'une de l'autre dans une boucle de validation vertueuse.

Le processus métier met en scène les concepts du diagramme de classes.

1. **Validation du modèle existant :** En décrivant un processus étape par étape, on "teste" notre diagramme de classes. Si une étape du processus dit "Le manager ajoute une justification à sa décision", mais que notre classe `Idée` n'a pas d'attribut pour stocker cette `justification`, alors notre modèle de domaine est incomplet !

2. **Découverte de nouveaux attributs ou classes :** Le processus force la découverte de données nécessaires. L'exemple le plus courant est l'**état** (ou statut) d'un objet. Le processus "Valider une idée" va naturellement faire émerger le besoin d'un attribut `statut` sur la classe `Idée`, qui pourra prendre des valeurs comme "Soumise", "En évaluation", "Approuvée", "Rejetée".

3. **Clarification des relations :** Le processus peut révéler des relations qui n'étaient pas évidentes. Si le processus dit "Le système archive les idées rejetées après 6 mois", cela implique une notion de date et une relation avec un concept d'archive qui n'existaient peut-être pas initialement.

5.3.1 Application Pratique à "InnovateBox"

Reprenons notre Epic de Manager : « En tant que manager, je veux gérer le cycle de vie des idées. » Ceci cache clairement un processus que nous pouvons nommer **« Processus de révision d'une idée »**.

Description textuelle du processus "Révision d'une idée"

1. **[Système]** notifie le **Manager** qu'une nouvelle **Idée** a été soumise.
2. **[Manager]** consulte l'**Idée** (son titre, sa description, son auteur).
3. **[Manager]** analyse les **Votes** associés à l'Idée pour juger de sa popularité.
4. **[Manager]** prend une décision : Approuver ou Rejeter.
5. **[Manager]** ajoute une **justification** pour documenter sa décision.
6. **[Système]** met à jour le **statut** de l'Idée en "Approuvée" ou "Rejetée" et enregistre la justification.
7. **[Système]** notifie l'**Employé** (auteur) du changement de statut de son Idée.

Impact sur notre Diagramme de Classes du Domaine : La description de ce processus nous oblige à **enrichir notre classe `Idée`**. En lisant les étapes 5 et 6, nous nous rendons compte qu'il manque des informations cruciales :

— Il faut un attribut pour suivre le cycle de vie de l'idée : 'statut : String' (ou un type énuméré).

— Il faut un attribut pour stocker la décision du manager : 'justification : String'.

Notre diagramme de classes, ainsi mis à jour, devient plus robuste et plus fidèle à la réalité du métier. Il n'est plus seulement une liste de concepts, mais une structure capable de supporter les opérations de l'entreprise.

Exercices : Modélisation des Processus et du Domaine

Exercice 1 : Identification de Processus

Contexte : Une petite boutique en ligne vend des produits artisanaux. Un client peut parcourir le site, ajouter des produits à son panier, passer une commande, payer et la recevoir. Si un produit ne convient pas, il peut le retourner.
Tâche : Identifiez et listez au moins trois processus métiers majeurs de cette boutique en ligne. Donnez-leur un nom clair (ex : "Gérer le stock").

(Voir solution 10 page 51)

Exercice 2 : Description d'un Processus

Contexte : En reprenant l'exemple de la boutique en ligne de l'exercice précédent, concentrons-nous sur le retour d'un produit.
Tâche : Décrivez sous forme de récit textuel structuré (liste numérotée) le processus « Traitement d'un retour client ». Précisez à chaque étape quel acteur (Client, Employé de la boutique, Système) intervient.

(Voir solution 10 page 51)

Exercice 3 : Impact sur le Modèle de Domaine

Contexte : Vous disposez d'un diagramme de classes très simple pour la boutique en ligne, contenant uniquement les classes `Client`, `Commande` et `Produit`.
Tâche : En vous basant sur le processus « Traitement d'un retour client » décrit à l'exercice 2, identifiez les classes, attributs ou associations manquants dans le modèle de domaine initial. Justifiez chaque ajout.

(Voir solution 10 page 51)

Exercice 1 : Extension d'InnovateBox - Les Commentaires

Contexte : Pour enrichir l'interaction sur "InnovateBox", la direction souhaite que les employés puissent ajouter des commentaires sur les idées soumises. Les managers doivent également pouvoir modérer ces commentaires en supprimant ceux qui sont inappropriés.
Tâches :

1. Décrivez le processus métier de « Modération d'un commentaire » du point de vue du manager.
2. Mettez à jour le Diagramme de Classes du Domaine d'InnovateBox (celui vu dans le cours) pour intégrer la notion de commentaire et de modération. Listez les nouvelles classes, les nouveaux attributs et les nouvelles associations avec leurs multiplicités.

(Voir solution 10 page 52)

Exercice 2 : Modélisation d'un Nouveau Domaine - La Bibliothèque

Contexte : Une petite bibliothèque municipale souhaite informatiser la gestion des prêts de livres. Un adhérent peut emprunter jusqu'à 5 livres pour une durée de 3 semaines. Le système doit pouvoir suivre quels livres sont sortis et par qui.

Tâches :

1. Décrivez le processus métier principal : « Emprunter un livre ».
2. À partir de ce processus et du contexte général, proposez une première version du Diagramme de Classes du Domaine pour ce système de bibliothèque. Identifiez les classes, les attributs pertinents et les associations clés avec leurs multiplicités. Pensez à la différence entre un "livre" en tant qu'œuvre et un "exemplaire" physique.

(Voir solution 10 page 53)

6 Synthèse de la Méthode sur un Cas Pratique

Au fil de ce document, nous avons exploré une démarche structurée pour transformer une idée vague en un plan de développement robuste. De l'enquête initiale à la modélisation conceptuelle, chaque étape est un maillon essentiel pour réduire les risques, aligner les équipes et s'assurer que le produit final répond à un besoin réel.

Pour illustrer la puissance de cette méthode, appliquons-la à un projet moderne et complexe : **le développement d'une IA générative pour aider le service RH d'une grande entreprise à recruter**.

6.1 Étape 1 : L'Enquête - De l'Idée au Besoin Réel

Tout commence par une demande du directeur des RH : « Je veux que nous utilisions l'IA générative pour optimiser nos recrutements. Ça nous fera gagner du temps. »

Cette demande est une intention, pas une spécification. Notre travail d'analyste commence ici. Nous ne disons pas « oui » immédiatement, nous menons l'enquête. Après plusieurs entretiens avec les recruteurs et les managers, nous découvrons les problèmes réels (les *besoins*) cachés derrière la demande initiale :

- **Problème 1 :** Les recruteurs passent en moyenne 4 heures par semaine à rédiger des descriptions de poste. Les textes manquent souvent d'homogénéité et peuvent contenir des biais involontaires.
- **Problème 2 :** Pour un poste populaire, ils reçoivent plus de 200 CVs. Le tri initial est chronophage et peu qualitatif.
- **Problème 3 :** La communication avec les candidats est impersonnelle. Rédiger des emails personnalisés (invitations, refus) prend un temps considérable qui pourrait être alloué à des entretiens qualitatifs.

Nous avons transformé une idée vague (« une IA pour recruter ») en problèmes métiers concrets. Nous pouvons maintenant définir les exigences.

6.2 Étape 2 : Définition des Exigences (EF et ENF)

Forte de cette analyse, nous pouvons lister des exigences vérifiables.

Exigences Fonctionnelles (Que doit faire le système ?)

- **EF-01 :** Le système doit générer une proposition de description de poste à partir d'un titre de poste et d'une liste de compétences clés.

— **EF-02 :** Le système doit analyser une liste de CVs (format PDF) et les classer par ordre de pertinence par rapport à une description de poste donnée.
— **EF-03 :** Le système doit rédiger une proposition d'email (invitation à un entretien, refus poli) en se basant sur le profil d'un candidat et le poste.

Exigences Non-Fonctionnelles (Comment doit-il le faire ?) Les ENF sont particulièrement critiques pour un projet d'IA.

— **Performance (ENF-Perf-01) :** Le traitement de 100 CVs ne doit pas dépasser 5 minutes.
— **Confidentialité (ENF-Secu-01) :** Aucune donnée personnelle identifiable (nom, email, photo) d'un candidat ne doit être envoyée à des API externes (ex : OpenAI, Google). Le traitement doit être local ou via des services garantissant la confidentialité des données (cloud privé, etc.).
— **Éthique et Équité (ENF-Ethic-01) :** Le modèle de classification des CVs doit être audité pour prouver qu'il ne discrimine pas sur la base du genre, de l'origine ethnique, de l'âge ou de l'adresse (critères à définir légalement). Cette exigence est **fondamentale** et non-négociable.
— **Utilisabilité (ENF-Usa-01) :** L'interface doit permettre à un recruteur non-technique d'utiliser 90% des fonctionnalités sans formation préalable.

6.3 Étape 3 : Formalisation (User Stories et Cas d'Utilisation)

Transformons l'exigence **EF-01** en une User Story.

User Story US-01 :
`En tant que recruteur, je veux générer automatiquement une ébauche de description de poste afin de gagner du temps et d'harmoniser nos offres d'emploi.`

Cette User Story devient ensuite un Cas d'Utilisation détaillé (« Générer une description de poste ») avec un scénario nominal (le recruteur saisit un titre, l'IA génère un texte, le recruteur l'édite et le valide) et des scénarios d'exception (que se passe-t-il si le titre est trop vague ? si l'IA ne renvoie rien ? si le texte généré est hors-sujet ?).

Exercice pour le lecteur :

1. Rédigez la User Story correspondant à l'exigence fonctionnelle **EF-02** (analyse des CVs).
2. Décrivez en quelques lignes le scénario nominal et au moins deux scénarios d'exception pour le cas d'utilisation correspondant.

6.4 Étape 4 : Modélisation (Diagramme de Classes du Domaine)

En analysant les exigences et les récits, nous identifions les concepts clés de notre domaine. Nous ne pensons pas au code, nous pensons au métier du recruteur.

Les concepts évidents sont : 'Recruteur', 'OffreEmploi', 'Candidature', 'Candidat', 'CV', 'Compétence'.

Nous pouvons les esquisser dans un premier diagramme de classes du domaine qui montre leurs relations :

— Un 'Recruteur' publie une ou plusieurs 'OffreEmploi'.
— Une 'OffreEmploi' requiert plusieurs 'Compétence'.
— Un 'Candidat' soumet une 'Candidature' pour une 'OffreEmploi'.
— Une 'Candidature' est associée à un et un seul 'CV'.

— Le ‘CV‘ met en évidence plusieurs ‘Compétence‘.

Ce modèle visuel devient un support de discussion inestimable avec le service RH pour valider que nous parlons bien le même langage. C’est le plan conceptuel de notre futur système.

Exercice pour le lecteur :

1. Ajoutez le concept d’‘Entretien‘ au diagramme. Quelle(s) relation(s) entretient-il avec ‘Recruteur‘, ‘Candidature‘ et ‘OffreEmploi‘ ?
2. Définissez les multiplicités (cardinalités) pour les relations esquissées ci-dessus (ex : un ‘Candidat‘ peut avoir zéro ou plusieurs ‘Candidature‘).

Ce qu’il faut retenir

Cette démarche complète, de l’enquête à la modélisation, n’est pas une contrainte bureaucratique. C’est une stratégie de **réduction des risques**. Sur un projet aussi sensible que l’IA dans les RH, où les enjeux techniques, éthiques et légaux sont immenses, cette approche structurée est la seule garantie de :

1. **Construire le bon produit :** Celui qui résout un vrai problème métier (gain de temps, harmonisation) tout en respectant les contraintes (éthique, confidentialité).
2. **Construire le produit correctement :** En fournissant une base claire et validée pour la conception technique, les tests et, au final, le développement.

Passer du temps sur l’analyse et la spécification n’est jamais du temps perdu. C’est un investissement qui assure que l’effort de développement, toujours coûteux, est dirigé vers le bon objectif.

7 Du Modèle au Prototype : Implémentation et Test

Le Diagramme de Classes du Domaine nous a donné un vocabulaire commun et une vision structurée du métier. Il est temps de le traduire en un plan technique pour notre prototype. Nous passons du *modèle conceptuel* au *modèle de conception*.

7.1 Du Modèle de Domaine au Modèle de Conception

Notre classe conceptuelle ‘OffreEmploi‘ devient une classe de conception ‘JobOffer‘ (nous adoptons ici une convention de nommage en anglais, standard dans le développement). La différence majeure est l’apparition des **méthodes** (les opérations) et des **classes de service** techniques.

Pour notre User Story **US-01** (« Générer une description de poste »), notre modèle de conception pourrait ressembler à ceci :

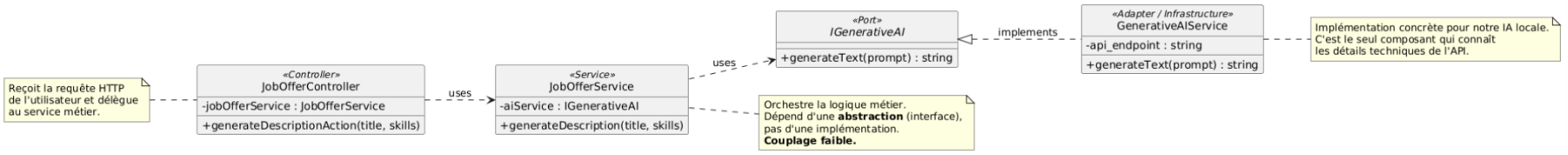


FIGURE 1 – Diagramme de Classes de Conception (Corrigé) illustrant l'architecture découplée du service de génération. Le service métier (`JobOfferService`) dépend de l'interface `IGenerativeAI`, ce qui le rend indépendant de l'implémentation concrète de l'IA.

Note : Un diagramme de classes de conception simple. La classe `JobOfferController` reçoit les requêtes de l'utilisateur. Elle utilise `JobOfferService` pour orchestrer la logique métier. Ce dernier utilise `GenerativeAIService`, un composant technique dédié à la communication avec notre IA locale, pour générer le texte. Comme on peut le voir dans la figure 1, l'architecture est maintenant découplée. Nous choisissons de nous concentrer sur une seule tranche verticale de l'application : l'implémentation complète de la User Story **US-01**.

7.2 Implémentation en Pseudo-code

Pour matérialiser cette architecture, nous utilisons du pseudo-code. Il est indépendant du langage et se concentre sur la logique et les interactions. Nous supposons que notre IA générative locale est accessible via une API à l'adresse 'http ://localhost :11434'.

7.2.1 Composant 1 : Le Service d'IA Générative

Ce composant est une abstraction. Son unique rôle est de communiquer avec l'API de l'IA.

```
CLASSE GenerativeAIService
    -- Attributs
    api_endpoint = "http://localhost:11434/api/generate"

    -- Méthodes
    FONCTION generateText(prompt)
        -- Construit la requête pour l'API locale
        requete = {
            "model": "rh-expert-model", -- Nom du modèle local
            "prompt": prompt,
            "stream": FAUX
        }

        -- Envoie la requête et attend la réponse
        reponse_api = ENVOYER_REQUETE_POST(api_endpoint, requete)

        -- Vérifie si la requête a réussi
        SI reponse_api.code_statut != 200 ALORS
            LANCER ErreurAPI("L'API de l'IA est indisponible ou a échoué.")
        FIN SI

        -- Extrait et retourne le texte généré de la réponse JSON
        texte_genere = reponse_api.json["response"]
        RETOURNER texte_genere
    FIN FONCTION
FIN CLASSE
```

7.2.2 Composant 2 : Le Service Métier

Ce service contient la logique spécifique à notre cas d'usage. Il construit le prompt et appelle le service d'IA.

```
CLASSE JobOfferService
    -- Attributs
    ia_service = NOUVEAU GenerativeAIService()
```

```
    -- Méthodes
    FONCTION generateDescription(titre_poste, competences_cles)
        -- Valider les entrées (non vides, etc.)
        SI titre_poste EST VIDE OU competences_cles EST VIDE ALORS
            LANCER ErreurValidation("Le titre et les compétences sont requis.")
        FIN SI

        -- Crée le prompt parfait pour l'IA, en lui donnant du contexte
        prompt_template = "Agis en tant qu'expert en recrutement. Rédige une
        description de poste professionnelle et inclusive pour le titre : '{titre}'.
        Le candidat idéal doit maîtriser les compétences suivantes : {competences}.
        Le ton doit être engageant et refléter une culture d'entreprise positive.
        N'utilise pas de jargon excessif."

        prompt_final = FORMATER_TEXTE(prompt_template,
                                      titre=titre_poste,
                                      competences=competences_cles)

        -- Appelle le service d'IA pour obtenir la description
        description_generee = ia_service.generateText(prompt_final)

        RETOURNER description_generee
    FIN FONCTION
FIN CLASSE
```

7.3 La Stratégie de Test du Prototype

Un code sans test est un code cassé par nature. Pour notre prototype, nous définissons trois niveaux de tests, chacun avec un objectif précis.

7.3.1 Tests Unitaires : Isoler et Vérifier chaque Brique

L'objectif est de tester chaque classe en isolation, en simulant (avec des *mocks* ou *stubs*) ses dépendances.

Test Unitaire 1 : Le service métier construit-il le bon prompt ?

```
TEST unitaire_JobOfferService_construit_bon_prompt()
    -- Arrange (Préparation)
    -- On crée un "faux" service d'IA qui n'appellera pas la vraie API
    ia_service_mock = MOCK(GenerativeAIService)
    job_offer_service = NOUVEAU JobOfferService(ia_service_mock)

    titre = "Développeur Python"
    competences = "API REST, Docker"

    -- Act (Action)
    job_offer_service.generateDescription(titre, competences)

    -- Assert (Vérification)
    -- On vérifie que la méthode ‘generateText‘ du mock a été appelée
    -- avec un prompt qui contient bien nos mots-clés.
```

```
    prompt_attendu_partiel_1 = "Développeur Python"
    prompt_attendu_partiel_2 = "API REST, Docker"

    VERIFIER_QUE(ia_service_mock.generateText A_ETE_APPELE_AVEC_UN_ARGUMENT_CONTENANT
                  prompt_attendu_partiel_1)
    VERIFIER_QUE(ia_service_mock.generateText A_ETE_APPELE_AVEC_UN_ARGUMENT_CONTENANT
                  prompt_attendu_partiel_2)
FIN TEST
```

7.3.2 Tests d'Intégration : Vérifier la Collaboration des Briques

L'objectif est de vérifier que nos composants communiquent correctement entre eux et avec l'API de l'IA (en version de test).

Test d'Intégration 1 : Le service métier s'intègre-t-il bien avec l'API ?

```
TEST integration_JobOfferService_avec_API_locale()
    -- Arrange
    -- Pour ce test, on utilise les vrais services, pas de mocks.
    -- On s'assure que le serveur d'IA local est bien démarré.
    job_offer_service = NOUVEAU JobOfferService()

    -- Act
    description = job_offer_service.generateDescription("Testeur QA", "Selenium")

    -- Assert
    -- On ne peut pas tester le texte exact, mais on peut vérifier qu'on
    -- a bien reçu une réponse cohérente.
    VERIFIER_QUE(description N_EST_PAS_VIDE)
    VERIFIER_QUE(TYPE_DE(description) EST CHAINE_DE_CARACTERES)
    VERIFIER_QUE(LONGUEUR(description) > 50)
FIN TEST
```

7.3.3 Tests d'Acceptation : Vérifier que le Besoin est Rempli

Ce test final simule le comportement de l'utilisateur et vérifie que la User Story est respectée de bout en bout. On utilise souvent une syntaxe lisible par le métier (Gherkin : Given/When/-Then).

Test d'Acceptation 1 : US-01 - Génération de la description

```
FONCTIONNALITÉ: Génération de description de poste

SCÉNARIO: Un recruteur génère une description de poste avec succès

ETANT DONNÉ que je suis un recruteur sur la page de création d'une offre d'emploi
QUAND je saisis "Ingénieur DevOps" dans le champ "Titre du poste"
ET que je saisis "Kubernetes, CI/CD, Terraform" dans le champ "Compétences clés"
ET que je clique sur le bouton "Générer la description"
ALORS le champ "Description" doit se remplir en moins de 10 secondes
ET le texte généré doit contenir les mots "Kubernetes" et "CI/CD"
ET le texte généré doit avoir une longueur d'au moins 200 mots.
```

7.4 Conclusion et Prochaines Étapes

Avec cette approche, nous avons un prototype fonctionnel pour une seule fonctionnalité, mais celui-ci est robuste, testé et validé. Le processus peut maintenant être répété pour la prochaine User Story, probablement la **US-02 (Analyse des CVs)**.

Exercice pour le lecteur : En vous basant sur la structure ci-dessus, décrivez en pseudo-code le service ‘CVAnalysisService‘ qui prendrait en entrée une liste de fichiers PDF (CVs) et une description de poste. Quelles seraient les principales difficultés et quels tests unitaires et d’intégration seraient indispensables pour garantir l’équité et l’absence de biais (ENF-Ethic-01) ?

8 Annexe 1 : Échecs Notables de Projets Informatiques

L'histoire de l'informatique est jalonnée de projets qui, malgré des budgets colossaux et des ambitions élevées, ont échoué de manière spectaculaire. Ces cas d'école sont riches d'enseignements et illustrent les risques liés à une mauvaise analyse des besoins, une gestion de projet défaillante ou une sous-estimation de la complexité.

Cas d'étude : Le Virtual Case File (VCF) du FBI

Contexte : Après le 11 septembre 2001, le FBI lance un projet pour moderniser son système de gestion d'enquêtes, entièrement basé sur le papier, afin de permettre aux agents de partager et d'analyser l'information numériquement.

Le problème : Le projet a souffert d'une dérive incontrôlée des objectifs (*scope creep*), avec des exigences changeant constamment. La direction du projet était incapable de formaliser un besoin clair, laissant les développeurs sans cap.

Le Coût (Financier et Humain) : Après 3 ans et **170 millions de dollars dépensés**, le projet est abandonné en 2005 car jugé inutilisable et déjà obsolète. Le coût humain réside dans le retard critique pris par le FBI pour se doter d'un outil essentiel à la sécurité nationale. [computerworld.com]

Cas d'étude : Le programme informatique de la NHS (Royaume-Uni)

Contexte : Lancé en 2002, ce projet visait à créer un dossier médical électronique centralisé pour chaque patient en Angleterre, l'un des plus grands projets informatiques civils au monde.

Le problème : Une complexité colossale abordée avec une approche "big bang" (tout d'un coup). Les utilisateurs finaux (médecins, infirmières) n'ont pas été suffisamment consultés, conduisant à une forte résistance. Les systèmes développés par différents fournisseurs étaient incompatibles.

Le Coût (Financier et Humain) : Officiellement abandonné en 2011. Le coût pour le contribuable est estimé à plus de **10 milliards de livres sterling**. Humainement, c'est une décennie de frustration pour le personnel soignant et un échec cuisant de modernisation du système de santé. [zdnet.com]

Cas d'étude : Le système de bagages de l'aéroport de Denver

Contexte : Pour son nouvel aéroport en 1995, Denver commande un système de tri de bagages entièrement automatisé, composé de milliers de chariots autonomes sur des kilomètres de rails. Une révolution technologique sur le papier.

Le problème : La complexité logicielle et matérielle a été massivement sous-estimée. Les tests, menés dans des conditions irréalistes, n'ont pas révélé les nombreux bugs du logiciel qui rendaient le système totalement non fiable (chariots perdus, bagages endommagés).

Le Coût (Financier et Humain) : L'échec du système a retardé l'ouverture de l'aéroport de 16 mois. Coût du retard : environ 1,1 million de dollars par jour, soit plus de **560 millions de dollars** au total. Ce fut un désastre d'image et un

chaos logistique pendant des années. [spectrum.ieee.org]

Cas d'étude : La sonde spatiale Mariner 1

Contexte : En 1962, la première mission américaine de survol de la planète Vénus.

Le problème : Une erreur de transcription dans une seule ligne de code. L'omission d'un trait d'union ('-') dans une formule mathématique du logiciel de guidage a été interprétée par la fusée comme une anomalie de trajectoire majeure, la faisant dévier de sa course.

Le Coût (Financier et Humain) : L'officier de sécurité a été contraint de détruire la fusée en vol pour éviter qu'elle ne s'écrase sur une zone habitée. Coût de la mission : **18,5 millions de dollars de 1962**. C'est l'exemple le plus célèbre de la plus petite erreur de programmation aux conséquences les plus chères. [blog.kodezi.com]

9 Annexe 2 : Exemple de Dialogue d'Analyse des Besoins

Le dialogue suivant illustre comment un analyste, par une série de questions ouvertes, peut transformer une demande initiale vague en une liste de problèmes métiers concrets et exploitables. La scène se déroule entre l'analyste du projet et la Directrice des Ressources Humaines (DRH).

Transcription de l'entretien : Projet d'IA pour le Recrutement

Directrice RH : Bonjour. Merci de prendre le temps. Comme je vous le disais, je souhaite que nous soyons plus innovants. Je veux lancer un projet d'IA générative pour optimiser nos recrutements. Tous nos concurrents s'y mettent, et je suis persuadée que cela peut nous faire gagner beaucoup de temps.

Analyste : Bonjour. C'est une excellente initiative, le potentiel est en effet énorme. Pour m'assurer que nous orientons bien le projet, pourriez-vous me décrire le parcours d'un recrutement chez vous, de A à Z ? Où sentez-vous le plus de « friction » aujourd'hui ?

Directrice RH : Bien sûr. Tout commence par la création d'une offre. Un manager a un besoin, il nous envoie quelques lignes, et un de mes recruteurs doit rédiger la description de poste complète pour notre site carrière et pour LinkedIn.

Analyste : Et cette phase de rédaction, comment se passe-t-elle ?

Directrice RH : (Soupire) C'est un premier point de friction, justement. C'est long. Chaque recruteur a son style, les managers veulent des retouches... On peut facilement perdre une demi-journée juste pour ça. Et au final, nos annonces manquent d'homogénéité. J'ai aussi toujours peur qu'un texte maladroit introduise des biais involontaires.

Analyste : Je note. Donc, un premier problème concret est la **lenteur et le manque d'uniformité dans la rédaction des offres**. Une fois l'offre publiée, que se passe-t-il ?

Directrice RH : Alors là... c'est le déluge. Pour un poste de développeur, on peut recevoir 200, parfois 300 CVs en une semaine.

Analyste : 300 CVs... Comment votre équipe gère-t-elle ce volume ?

Directrice RH : Manuellement. Un recruteur passe des heures, parfois deux jours, à ouvrir chaque PDF pour faire un premier tri. C'est fastidieux, répétitif, et à faible valeur ajoutée. Le vrai travail d'un recruteur, c'est de parler aux gens, pas de cocher des cases dans un tableur. Je suis aussi persuadée qu'avec la fatigue, on passe à côté de profils intéressants.

Analyste : C'est très clair. Le deuxième grand défi est donc le **tri initial des CVs, qui est chronophage et peu qualitatif**. Admettons que le recruteur ait sa liste de 20 candidats prometteurs. Quelle est la suite ?

Directrice RH : Il faut contacter ces 20 personnes pour un premier entretien. Mais il faut aussi et surtout répondre aux 280 autres pour leur annoncer que leur candidature n'est pas retenue.

Analyste : Et j'imagine que c'est une tâche sensible...

Directrice RH : Plus que sensible. C'est notre image de marque, notre « marque employeur ». Un email de refus impersonnel et standard, c'est désastreux. Mais rédiger 280 emails un minimum personnalisés... c'est tout simplement impossible. Donc on se rabat sur des modèles bateau, et je sais que ce n'est pas bon. Le temps passé sur ces tâches administratives est du temps qui n'est pas alloué aux entretiens de fond.

Analyste : Parfait. C'est un troisième point crucial : la **gestion de la communication de masse avec les candidats, qui est à la fois impersonnelle et pourtant très chronophage**.

Analyste (Synthèse) : Merci beaucoup, c'est infiniment plus clair. Au lieu d'une idée générale d'« IA pour le recrutement », nous avons maintenant trois problèmes métiers précis à résoudre :

1. Accélérer et harmoniser la rédaction des descriptions de poste.
2. Automatiser le pré-tri des CVs pour en extraire les plus pertinents.
3. Personnaliser à grande échelle la communication avec les candidats (invitations et refus).

Nous allons pouvoir commencer à imaginer des solutions ciblées pour chacun de ces points.

Directrice RH : Exactement. Vu comme ça, je vois déjà beaucoup mieux la valeur ajoutée potentielle.

10 Corrections des exercices

Correction de l'Exercice 1

En tant que employé, je veux voter pour une idée qui me semble pertinente afin de aider les managers à identifier les suggestions les plus populaires..
Alternative possible pour le bénéfice : *« ...afin de montrer mon soutien à mon collègue. »*

Correction de l'Exercice 2

Acteur Primaire : Employé

Précondition : L'Employé est authentifié et visualise la page de détail d'une idée sur laquelle il n'a pas encore voté.

Postcondition de succès : Le compteur de votes de l'idée est incrémenté de 1. Le système a enregistré que l'Employé a voté pour cette idée.

Postcondition d'échec : Le compteur de votes de l'idée n'est pas modifié. L'état du système reste inchangé.

Correction de l'Exercice 3

Scénario Nominal

1. L'Employé clique sur le bouton « Voter » à côté d'une idée.
2. Le Système vérifie que l'Employé est authentifié.
3. Le Système vérifie que l'Employé n'a pas déjà voté pour cette idée.
4. Le Système incrémente le compteur de votes de l'idée de +1 en base de données.
5. Le Système enregistre que cet Employé a voté pour cette idée.
6. Le Système met à jour l'interface pour montrer le nouveau total de votes et désactive le bouton de vote pour cet utilisateur sur cette idée.

Scénario d'Exception : L'utilisateur a déjà voté

3a. À l'étape 3, le Système détecte que l'Employé a déjà voté pour cette idée.

4a. Le Système ne modifie pas le compteur de votes. Il affiche un message d'information discret, par exemple : « Vous avez déjà voté pour cette idée ». Le cas d'utilisation se termine.

Correction de l'Exercice 4

Voici trois exemples de décomposition de l'Epic « Gérer le cycle de vie des idées » :

- **Story 1 (Approuver/Rejeter) :** *En tant que manager, je veux approuver ou rejeter une idée afin de indiquer rapidement si une suggestion est retenue pour une étude plus approfondie..*
- **Story 2 (Changer le statut) :** *En tant que manager, je veux changer le statut d'une idée (ex : "En étude", "Planifiée", "Réalisée") afin de suivre la progression de l'idée et informer les autres employés de son avancement..*
- **Story 3 (Archiver) :** *En tant que manager, je veux archiver les idées rejetées ou terminées depuis longtemps afin de garder la liste des idées actives claire et pertinente..*

Annexe : Solutions sur les Diagrammes de Classes

Correction de l'Exercice TD 1 : Concepts de Base et Attributs

Classe `Rédacteur` : — **Attributs :** 'nom' (implicite).

Classe `Article` : — **Attributs :** 'titre', 'contenu', 'dateDePublication'.

Association : — Relation "estAuteurDe" entre `Rédacteur` et `Article`.
- — **Multiplicités :** Un `Rédacteur` peut être l'auteur de **0..*** `Articles`. Un `Article` a **1** seul `Rédacteur`.

Correction de l'Exercice TD 2 : Ajout d'une Association Simple

Nouvelle Classe `Commentaire` : — **Attributs :** 'auteurNom' (le nom saisi par le visiteur), 'texte'.

Nouvelle Association : — Relation "concerne" entre `Commentaire` et `Article`.
- — **Multiplicités :** Un `Article` peut avoir **0..*** `Commentaires`. Un `Commentaire` est lié à **1** seul `Article`.

Correction de l'Exercice TD 3 : Héritage (Généralisation/Spécialisation)

Classe de base (parente) `Utilisateur` : — **Attributs :** 'pseudo', 'motDePasse'.

Classe dérivée (enfant) `Abonné` : — Hérite de `Utilisateur`. N'ajoute pas d'attributs spécifiques dans ce cas, mais possède des capacités différentes.

Classe dérivée (enfant) `Modérateur` : — Hérite de `Utilisateur`.

Relation d'héritage : — 'Abonné' *est un* 'Utilisateur'.
- — 'Modérateur' *est un* 'Utilisateur'.
- — Graphiquement, une flèche avec un triangle vide pointe des classes enfants vers la classe parente.

Correction de l'Exercice TD 4 : La Classe-Association

La relation entre un `Employé` et un `Projet` n'est pas simple, car elle possède ses propres données ('rôle', 'dateDeDebut'). C'est le cas parfait pour une classe-association.

Classes de base : `Employé` (avec attribut 'nom') et `Projet` (avec attribut 'nomProjet').

Classe-Association `Assignation` : — **Attributs :** 'rôle', 'dateDeDebut'.
- — **Justification :** Ces attributs n'appartiennent ni à l'employé (son rôle change par projet) ni au projet (chaque employé a un rôle différent). Ils appartiennent à la relation entre les deux.

Associations : — Un `Employé` peut avoir **0..*** `Assignations`.
- — Un `Projet` peut avoir **1..*** `Assignations` (un projet a au moins un employé).
- — Une `Assignation` lie **1** `Employé` et **1** `Projet`.

Correction de l'Exercice TP 1 : Modélisation d'un Site E-commerce

Classe `Client` : 'nom', 'email', 'adresseDeLivraison'.

Classe `Produit` : 'nom', 'description', 'prix'.

Classe `Commande` : 'date', 'statut' (ex : "En cours", "Expédiée").

Classe `LigneDeCommande` (Classe-Association) : — **Attributs :** 'quantite'.
- **Justification :** La quantité dépend à la fois du produit ET de la commande dans laquelle il se trouve.

Associations : — `Client` 1 – 0..* `Commande`. (Un client peut passer plusieurs commandes).
- `Commande` 1 – 1..* `LigneDeCommande`. (Une commande a au moins une ligne).
- `Produit` 1 – 0..* `LigneDeCommande`. (Un produit peut être dans plusieurs lignes de commande).

Correction de l'Exercice TP 2 : Modélisation d'un Système de Réservation

Classe `Organisateur` : 'nom'.

Classe `Client` : 'nom', 'email'.

Classe `Evenement` : 'nom', 'lieu', 'date', 'nbPlacesTotal'.

Classe `Reservation` : 'dateReservation', 'nbPersonnes'.

Classe `Billet` : 'codeUnique', 'statut' ("Valide", "Scanné").

Associations : — `Organisateur` 1 – 1..* `Evenement`.
- `Client` 1 – 0..* `Reservation`.
- `Evenement` 1 – 0..* `Reservation`. (Un événement peut avoir plusieurs réservations).
- `Reservation` 1 – 1..* `Billet`. (Une réservation génère un ou plusieurs billets).

Correction de l'Exercice TP 3 : Modélisation d'une Plateforme pour Freelances

Classe `Client` : 'nomEntreprise', 'emailContact'.

Classe `Freelance` : 'nom', 'email', 'specialite'.

Classe `Mission` : 'titre', 'description', 'statut' ("Ouverte", "En cours", "Terminée").

Classe `Contrat` : 'montantTotal', 'delais', 'dateSignature'.

Classe `Facture` : 'numeroFacture', 'dateEmission', 'statut' ("Emise", "Payée").

Associations : — 'Client' 1 – 1..* 'Mission'. (Un client propose des missions).
- 'Mission' 1 – 0..1 'Contrat'. (Une mission peut ne pas encore avoir de contrat).
- 'Freelance' 1 – 0..* 'Contrat'. (Un freelance peut avoir plusieurs contrats).
- 'Contrat' 1 – 1 'Facture'. (Un contrat aboutit à une facture).

Note : Le `Contrat` agit comme une classe-association complexe liant une `Mission` et un `Freelance`.

Annexe : Solutions des Exercices (Processus)

Correction de l'Exercice TD 1 : Identification de Processus

Voici plusieurs processus métiers majeurs pour une boutique en ligne :

- **Processus de gestion d'une commande :** Ensemble des étapes allant du paiement par le client à la livraison du colis.
- **Processus de traitement d'un retour client :** De la demande de retour par le client jusqu'à son remboursement.
- **Processus de gestion du catalogue produit :** Ajout, modification ou suppression de produits mis en vente par un employé.
- **Processus de réapprovisionnement du stock :** Commande de nouveaux produits auprès des fournisseurs lorsque le stock est bas.

Correction de l'Exercice TD 2 : Description d'un Processus

Processus : « Traitement d'un retour client »

1. **[Client]** initie une demande de retour depuis son espace client.
2. **[Système]** vérifie que la commande est éligible au retour (ex : moins de 14 jours).
3. **[Système]** génère une étiquette de retour et l'envoie au client.
4. **[Client]** emballe le produit et le dépose au transporteur avec l'étiquette.
5. **[Employé]** réceptionne le colis retourné à l'entrepôt.
6. **[Employé]** inspecte le produit pour vérifier son état.
7. **[Employé]** valide le retour dans le système.
8. **[Système]** déclenche le remboursement du montant de la commande au client.
9. **[Système]** notifie le client que le remboursement a été effectué.

Correction de l'Exercice TD 3 : Impact sur le Modèle de Domaine

Le processus de retour révèle plusieurs manques dans le modèle 'Client'–'Commande'–'Produit' :

- **Classe manquante :** `Retour`
  - **Justification :** Le processus manipule un concept central, le "retour", qui a sa propre existence et son propre cycle de vie. Il ne peut pas être un simple attribut de la commande.
  - **Association :** Cette classe `Retour` serait liée à exactement **1** `Commande`.
- **Attribut manquant :** `statut` **sur la classe** `Retour`
  - **Justification :** Le processus montre que le retour passe par plusieurs états (ex : "Demandé", "Reçu", "Inspecté", "Remboursé"). Cet attribut est indispensable pour savoir où en est chaque retour.
- **Attribut manquant :** `dateDemande` **sur la classe** `Retour`
  - **Justification :** L'étape 2 du processus ("vérifie que la commande est éligible") implique de connaître la date de la demande pour la comparer à la date de la commande.

Correction de l'Exercice TP 1 : Extension d'InnovateBox

1. Processus « Modération d'un commentaire »

1. **[Système]** détecte un commentaire signalé par un employé OU **[Manager]** consulte proactivement les commentaires d'une idée.
2. **[Manager]** lit le contenu du **Commentaire** et évalue s'il est inapproprié.
3. ***(Décision)*** Si le commentaire est jugé acceptable, le processus se termine.
4. ***(Décision)*** Si le commentaire est jugé inapproprié, le **Manager** clique sur "Supprimer".
5. **[Système]** marque le commentaire comme masqué ou le supprime de la base de données.
6. **[Système]** met à jour l'affichage de l'idée pour ne plus montrer le commentaire.

2. Mise à jour du Diagramme de Classes du Domaine

Nouvelle Classe : `Commentaire` Pour représenter un commentaire.

- **Attributs :** 'texte : String', 'dateCreation : Date', 'statut : String' (ex : "Visible", "Signalé", "Masqué").

Nouvelle Association (Auteur du commentaire) :

- Lie la classe `Employé` à la classe `Commentaire`.
- **Multiplicités :** Un `Employé` peut écrire **0..*** `Commentaires`. Un `Commentaire` est écrit par **1** seul `Employé`.

Nouvelle Association (Commentaire sur une idée) :

- Lie la classe `Idée` à la classe `Commentaire`.
- **Multiplicités :** Une `Idée` peut avoir **0..*** `Commentaires`. Un `Commentaire` appartient à **1** seule `Idée`.

Correction de l'Exercice TP 2 : Modélisation d'un Nouveau Domaine

1. Processus « Emprunter un livre »

1. **[Adhérent]** choisit un **Exemplaire** physique en rayon et se présente au comptoir.
2. **[Bibliothécaire]** scanne la carte de l'**Adhérent**.
3. **[Système]** vérifie le statut de l'adhérent (cotisation à jour, pas de retard, moins de 5 prêts en cours).
4. **[Bibliothécaire]** scanne le code-barres de l'**Exemplaire**.
5. **[Système]** vérifie que l'exemplaire est disponible.
6. **[Système]** crée un nouvel **Emprunt** liant l'adhérent et l'exemplaire, avec la date du jour comme date de prêt et la date du jour + 3 semaines comme date de retour prévue.
7. **[Système]** met à jour le statut de l'**Exemplaire** à "Emprunté".
8. **[Bibliothécaire]** informe l'adhérent de la date de retour.

2. Diagramme de Classes du Domaine de la Bibliothèque

Classe `Livre` : L'œuvre intellectuelle.
- **Attributs :** 'ISBN', 'titre', 'auteur'.

Classe `Exemplaire` : La copie physique d'un livre.
- **Attributs :** 'idExemplaire' (code-barres), 'statut' ("Disponible", "Emprunté", "En réparation").
- **Association :** Un `Livre` peut avoir **1..*** `Exemplaires`. Un **Exemplaire** correspond à **1** seul `Livre`.

Classe `Adherent` : L'utilisateur de la bibliothèque.
- **Attributs :** 'idAdherent', 'nom', 'adresse'.

Classe `Emprunt` (Classe-Association) : Formalise l'acte de prêt.
- **Attributs :** 'dateEmprunt', 'dateRetourPrevue', 'dateRetourReelle' (optionnel).
- **Association :** Un `Emprunt` lie **1** `Adherent` et **1** `Exemplaire`. Un `Adherent` peut avoir **0..*** `Emprunts`. Un `Exemplaire` peut être dans **0..1** `Emprunt` à un instant T.