

Analyse : de l'exigence à la spécification

CM2 — Formaliser : de la *story* à l'architecture

Ressource R3.03

BUT2 Informatique

10 septembre 2026

CM1 / TD1	le besoin réel est identifié, les exigences sont écrites et vérifiables
TP1	les seuils chiffrés ont été confrontés à la machine
Aujourd'hui	comment écrit-on ces exigences pour qu'elles soient à la fois validables par le client et exploitables par l'équipe ?

#	Partie	Durée
1	Pourquoi le cahier des charges de 300 pages a échoué	10 min
2	Les <i>User Stories</i> : la voix de l'utilisateur	25 min
3	Les cas d'utilisation : le scénario de l'interaction	20 min
4	Du scénario nominal à l'architecture : le port	20 min

Ce qui ne marche pas : le cahier des charges monolithique

Pendant des décennies : des documents de centaines de pages décrivant le système dans ses moindres détails techniques.

Quatre défauts rédhibitoires

- longs à rédiger, obsolètes avant d'être finis ;
- rarement lus en entier ;
- la *finalité* du projet se noie sous les détails ;
- l'équipe passe plus de temps à interpréter le document qu'à répondre au besoin.

Les méthodes agiles ont proposé l'inverse : des artefacts **courts**, **centrés sur la valeur**, et qui assument d'être **incomplets**.

Pendant des décennies : des documents de centaines de pages décrivant le système dans ses moindres détails techniques.

Quatre défauts rédhibitoires

- longs à rédiger, obsolètes avant d'être finis ;
- rarement lus en entier ;
- la *finalité* du projet se noie sous les détails ;
- l'équipe passe plus de temps à interpréter le document qu'à répondre au besoin.

Les méthodes agiles ont proposé l'inverse : des artefacts **courts**, **centrés sur la valeur**, et qui assument d'être **incomplets**.

En tant que **⟨rôle⟩**,
je veux **⟨action⟩**
afin de **⟨bénéfice⟩**.

QUI ? un acteur précis, jamais « un utilisateur ». Définir le rôle force à adopter son point de vue.

QUOI ? le comportement attendu, l'objectif à atteindre.

POURQUOI ? **le cœur de la story** : la valeur, le gain, la justification métier. C'est ce qui permet à l'équipe de proposer une meilleure solution.

Une story n'est pas une spécification

Les « 3 C » de Ron Jeffries

Card la story tient sur une fiche. Sa brièveté est *volontaire* : c'est un rappel, pas un contrat.

Conversation l'essentiel de l'information n'est pas écrit, il s'échange. La carte est « une promesse de conversation ».

Confirmation cette conversation débouche sur des **critères d'acceptation** : les conditions objectives qui diront que c'est terminé.

Voilà pourquoi une bonne story reste légère : la richesse est dans la conversation et les critères d'acceptation, **jamais dans la longueur du texte.**

Une story n'est pas une spécification

Les « 3 C » de Ron Jeffries

Card la story tient sur une fiche. Sa brièveté est *volontaire* : c'est un rappel, pas un contrat.

Conversation l'essentiel de l'information n'est pas écrit, il s'échange. La carte est « une promesse de conversation ».

Confirmation cette conversation débouche sur des **critères d'acceptation** : les conditions objectives qui diront que c'est terminé.

Voilà pourquoi une bonne story reste légère : la richesse est dans la conversation et les critères d'acceptation, **jamais dans la longueur du texte.**

Centrée sur le bénéfice métier

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

On comprend le gain. L'équipe peut même proposer mieux : gérer plusieurs adresses.

Une 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 en base. »

La première décrit un écran, la seconde une tâche. Aucune n'explique la finalité métier.

Centrée sur le bénéfice métier

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

On comprend le gain. L'équipe peut même proposer mieux : gérer plusieurs adresses.

Une 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 en base. »

La première décrit un écran, la seconde une tâche. Aucune n'explique la finalité métier.

INVEST : la grille de relecture (Bill Wake)

I	Independent	réalisable indépendamment des autres, donc priorisable librement
N	Negotiable	pas un contrat rigide : le point de départ d'une conversation
V	Valuable	apporte une valeur tangible à l'utilisateur ou au client
E	Estimable	l'équipe sait évaluer l'effort ; sinon elle est trop vague
S	Small	tient dans une itération ; sinon c'est une <i>Epic</i> à découper
T	Testable	des critères objectifs existent pour prouver qu'elle est finie

L'*Epic*

Une story trop grosse pour un sprint. Ce n'est pas un défaut : c'est un **conteneur** destiné à être découpé. *Si les stories sont les scènes d'un film, l'Epic en est l'acte.*

INVEST : la grille de relecture (Bill Wake)

I	Independent	réalisable indépendamment des autres, donc priorisable librement
N	Negotiable	pas un contrat rigide : le point de départ d'une conversation
V	Valuable	apporte une valeur tangible à l'utilisateur ou au client
E	Estimable	l'équipe sait évaluer l'effort ; sinon elle est trop vague
S	Small	tient dans une itération ; sinon c'est une <i>Epic</i> à découper
T	Testable	des critères objectifs existent pour prouver qu'elle est finie

L'*Epic*

Une story trop grosse pour un sprint. Ce n'est pas un défaut : c'est un **conteneur** destiné à être découpé. *Si les stories sont les scènes d'un film, l'Epic en est l'acte.*

⇒ Fil rouge — snapguard : **US-01**

*« En tant qu'**administrateur**, je veux qu'une sauvegarde quotidienne ne consomme que l'espace des fichiers réellement modifiés, afin de conserver un mois d'historique sur le NAS existant plutôt que deux jours. »*

La story qu'il ne faut pas écrire

« En tant que développeur, je veux utiliser les liens durs POSIX afin d'optimiser le stockage. »

Le rôle n'est pas un utilisateur, le « quoi » est une solution, le bénéfice est technique.

La preuve : si demain le système de fichiers offre une déduplication native, **US-01** reste valide et l'implémentation change. Écrire au niveau du besoin *préserve la liberté de conception*.

⇒ Fil rouge — snapguard : **US-01**

*« En tant qu'**administrateur**, je veux qu'une sauvegarde quotidienne ne consomme que l'espace des fichiers réellement modifiés, afin de conserver un mois d'historique sur le NAS existant plutôt que deux jours. »*

La story qu'il ne faut pas écrire

« En tant que développeur, je veux utiliser les liens durs POSIX afin d'optimiser le stockage. »

Le rôle n'est pas un utilisateur, le « quoi » est une solution, le bénéfice est technique.

La preuve : si demain le système de fichiers offre une déduplication native, **US-01** reste valide et l'implémentation change. Écrire au niveau du besoin *préserve la liberté de conception*.

Les critères d'acceptation : Gherkin

Langage du *Behaviour-Driven Development* (Dan North, 2006). Une même phrase sert de **spécification**, de **documentation** et de **test automatisé**.

```
Scenario : une sauvegarde quotidienne ne recopie que le travail du jour
  Etant donne un repertoire de travail de 20 fichiers de 100 Kio
  Et une premiere sauvegarde complete realisee lundi
  Quand je modifie 1 seul fichier puis relance la sauvegarde mardi
  Alors les deux instantanes occupent ensemble a peine plus que l'espace d'un seul
  Et le rapport indique qu'un seul fichier a ete recopie
  Et l'instantane de mardi contient pourtant les 20 fichiers
```

La règle d'or

Un critère d'acceptation ne contient **aucun nom de fonction, aucun type, aucune structure de données**. Il doit rester compréhensible par Karim — c'est ce qui permettra de le traduire en script de test sans le réécrire.

Les critères d'acceptation : Gherkin

Langage du *Behaviour-Driven Development* (Dan North, 2006). Une même phrase sert de **spécification**, de **documentation** et de **test automatisé**.

```
Scenario : une sauvegarde quotidienne ne recopie que le travail du jour
  Etant donne un repertoire de travail de 20 fichiers de 100 Kio
  Et une premiere sauvegarde complete realisee lundi
  Quand je modifie 1 seul fichier puis relance la sauvegarde mardi
  Alors les deux instantanes occupent ensemble a peine plus que l'espace d'un seul
  Et le rapport indique qu'un seul fichier a ete recopie
  Et l'instantane de mardi contient pourtant les 20 fichiers
```

La règle d'or

Un critère d'acceptation ne contient **aucun nom de fonction, aucun type, aucune structure de données**. Il doit rester compréhensible par Karim — c'est ce qui permettra de le traduire en script de test sans le réécrire.

Quand la story ne suffit plus

User Story

Pourquoi la fonctionnalité est nécessaire, et **pour qui**.

Phrase courte, informelle. Une promesse de conversation.

Garde l'équipe concentrée sur la valeur.

Cas d'utilisation

Comment l'utilisateur et le système interagissent, exhaustivement.

Document structuré : préconditions, scénarios, postconditions.

Force à anticiper les cas limites.

Ils ne s'opposent pas : ils sont **complémentaires**. L'un porte l'intention, l'autre le contrat.

Quand la story ne suffit plus

User Story

Pourquoi la fonctionnalité est nécessaire, et **pour qui**.

Phrase courte, informelle. Une promesse de conversation.

Garde l'équipe concentrée sur la valeur.

Cas d'utilisation

Comment l'utilisateur et le système interagissent, exhaustivement.

Document structuré : préconditions, scénarios, postconditions.

Force à anticiper les cas limites.

Ils ne s'opposent pas : ils sont **complémentaires**. L'un porte l'intention, l'autre le contrat.

- ➊ **Identifier acteurs et cas d'utilisation** : qui interagit, et pour atteindre quel objectif ?
- ➋ **Dessiner le diagramme** de cas d'utilisation. *C'est une table des matières* : il dit qui peut faire quoi, jamais comment.
- ➌ **Rédiger les spécifications textuelles**. **C'est ici qu'est la véritable analyse.**
- ➍ **Concevoir** : diagrammes de séquence, diagramme de classes.

L'erreur classique

Croire que le diagramme est l'analyse. Un diagramme de cas d'utilisation seul ne dit ni dans quel ordre, ni ce qui se passe si le disque est plein. Sa valeur est intégralement dans les descriptions textuelles qui le suivent.

- ❶ **Identifier acteurs et cas d'utilisation** : qui interagit, et pour atteindre quel objectif ?
- ❷ **Dessiner le diagramme** de cas d'utilisation. *C'est une table des matières* : il dit qui peut faire quoi, jamais comment.
- ❸ **Rédiger les spécifications textuelles**. **C'est ici qu'est la véritable analyse.**
- ❹ **Concevoir** : diagrammes de séquence, diagramme de classes.

L'erreur classique

Croire que le diagramme est l'analyse. Un diagramme de cas d'utilisation seul ne dit ni dans quel ordre, ni ce qui se passe si le disque est plein. Sa valeur est intégralement dans les descriptions textuelles qui le suivent.

Le bon niveau : la métaphore d'altitude de Cockburn

Niveau	Exemple	Verdict
Cerf-volant	« augmenter les ventes »	trop haut
Niveau de la mer	« payer sa commande », « se connecter »	le bon
Poisson	« valider un champ », « chiffrer un mot de passe »	trop bas

Le repère

Un cas d'utilisation « au niveau de la mer » exprime un objectif que l'acteur accomplit **d'une traite**, et qu'il cocherait mentalement comme « fait » avant de quitter son écran.

Concept introduit par Ivar Jacobson (1986) ; niveaux d'objectif raffinés par Alistair Cockburn, *Writing Effective Use Cases*, 2000.

Le bon niveau : la métaphore d'altitude de Cockburn

Niveau	Exemple	Verdict
Cerf-volant	« augmenter les ventes »	trop haut
Niveau de la mer	« payer sa commande », « se connecter »	le bon
Poisson	« valider un champ », « chiffrer un mot de passe »	trop bas

Le repère

Un cas d'utilisation « au niveau de la mer » exprime un objectif que l'acteur accomplit **d'une traite**, et qu'il cocherait mentalement comme « fait » avant de quitter son écran.

Concept introduit par Ivar Jacobson (1986) ; niveaux d'objectif raffinés par Alistair Cockburn, *Writing Effective Use Cases*, 2000.

Un cas d'utilisation met en scène deux sortes d'acteurs

L'acteur primaire

Déclenche le cas d'utilisation et en attend un résultat.

Chez BioSeq : cron, ou l'administrateur.

L'acteur secondaire

Est sollicité par le système, pour que celui-ci atteigne son objectif.

Chez BioSeq : le système de fichiers.

Le système est donc au milieu de **deux conversations**.

Or une conversation avec un acteur, c'est une **frontière** du système.

Et une frontière, cela se spécifie : c'est une interface.

Un cas d'utilisation met en scène deux sortes d'acteurs

L'acteur primaire

Déclenche le cas d'utilisation et en attend un résultat.

Chez BioSeq : cron, ou l'administrateur.

L'acteur secondaire

Est sollicité par le système, pour que celui-ci atteigne son objectif.

Chez BioSeq : le système de fichiers.

Le système est donc au milieu de **deux conversations**.

Or une conversation avec un acteur, c'est une **frontière** du système.

Et une frontière, cela se spécifie : c'est une interface.

Un cas d'utilisation met en scène deux sortes d'acteurs

L'acteur primaire

Déclenche le cas d'utilisation et en attend un résultat.

Chez BioSeq : cron, ou l'administrateur.

L'acteur secondaire

Est sollicité par le système, pour que celui-ci atteigne son objectif.

Chez BioSeq : le système de fichiers.

Le système est donc au milieu de **deux conversations**.

Or une conversation avec un acteur, c'est une **frontière** du système.

Et une frontière, cela se spécifie : c'est une interface.

⇒ Fil rouge — snapguard : CU-01 « Créer un instantané » — extrait

- 1 L'Ordonnanceur déclenche la création d'un instantané pour une source donnée.
- 2 Le Système identifie le dernier instantané **complet** de cette source.
- 3 Le Système crée le répertoire du nouvel instantané, horodaté.
- 4 Le Système parcourt récursivement la source.
- 5 Pour chaque répertoire, le Système recrée la structure correspondante.
- 6 Pour chaque fichier, le Système compare taille et date à celles de l'instantané de référence.
- 7 Si le fichier est inchangé, le Système le lie plutôt que de le copier.
- 8 Sinon, le Système le copie.
- 9 Le Système marque l'instantané complet et produit le rapport.

Le chemin heureux est **facile** : tout le monde le trouve, et tout le monde écrit à peu près le même. La difficulté — et la valeur — est ailleurs. *Ce sera l'objet du CM3.*

⇒ Fil rouge — snapguard : CU-01 « Créer un instantané » — extrait

- ① L'Ordonnanceur déclenche la création d'un instantané pour une source donnée.
- ② Le Système identifie le dernier instantané **complet** de cette source.
- ③ Le Système crée le répertoire du nouvel instantané, horodaté.
- ④ Le Système parcourt récursivement la source.
- ⑤ Pour chaque répertoire, le Système recrée la structure correspondante.
- ⑥ Pour chaque fichier, le Système compare taille et date à celles de l'instantané de référence.
- ⑦ Si le fichier est inchangé, le Système le lie plutôt que de le copier.
- ⑧ Sinon, le Système le copie.
- ⑨ Le Système marque l'instantané complet et produit le rapport.

Le chemin heureux est **facile** : tout le monde le trouve, et tout le monde écrit à peu près le même. La difficulté — et la valeur — est ailleurs. *Ce sera l'objet du CM3.*

Relisez le scénario nominal : à chaque étape, qui parle ?

Étape	Interlocuteur	Opération
1 L'Ordonnanceur déclenche la création	acteur primaire	<i>entrée</i>
2 Le Système identifie le dernier instantané complet	syst. de fichiers	lister
3 Le Système crée le répertoire horodaté	syst. de fichiers	creer_repertoire
4 Le Système parcourt la source	syst. de fichiers	lister
6 Le Système compare taille et date	syst. de fichiers	stat
7 Le Système lie le fichier inchangé	syst. de fichiers	lier_dur
8 Le Système copie le fichier modifié	syst. de fichiers	copier
9 Le Système produit le rapport	acteur primaire	<i>sortie</i>

La règle

Toute étape qui commence par « **Le Système...** » et touche une ressource extérieure est un **appel au port**. Les étapes 1 et 9 décrivent l'autre frontière : la ligne de commande.

Relisez le scénario nominal : à chaque étape, qui parle ?

Étape	Interlocuteur	Opération
1 L'Ordonnanceur déclenche la création	acteur primaire	<i>entrée</i>
2 Le Système identifie le dernier instantané complet	syst. de fichiers	lister
3 Le Système crée le répertoire horodaté	syst. de fichiers	creer_repertoire
4 Le Système parcourt la source	syst. de fichiers	lister
6 Le Système compare taille et date	syst. de fichiers	stat
7 Le Système lie le fichier inchangé	syst. de fichiers	lier_dur
8 Le Système copie le fichier modifié	syst. de fichiers	copier
9 Le Système produit le rapport	acteur primaire	<i>sortie</i>

La règle

Toute étape qui commence par « **Le Système...** » et touche une ressource extérieure est un **appel au port**. Les étapes 1 et 9 décrivent l'autre frontière : la ligne de commande.

lister stat creer_repertoire copier lier_dur

Ce ne sont pas cinq fonctions choisies par un architecte

Ce sont les lignes 2 à 8 du scénario nominal, dédoublonnées. La liste ne sort pas d'un catalogue technique : **elle sort de votre analyse.**

D'où le critère qui vous servira au TP : si votre interface compte dix opérations, c'est que vous décrivez la libc et non votre scénario.

Une interface qui expose tout n'abstrait rien.

lister stat creer_repertoire copier lier_dur

Ce ne sont pas cinq fonctions choisies par un architecte

Ce sont les lignes 2 à 8 du scénario nominal, dédoublonnées. La liste ne sort pas d'un catalogue technique : **elle sort de votre analyse.**

D'où le critère qui vous servira au TP : si votre interface compte dix opérations, c'est que vous décrivez la libc et non votre scénario.

Une interface qui expose tout n'abstrait rien.

lister stat creer_repertoire copier lier_dur

Ce ne sont pas cinq fonctions choisies par un architecte

Ce sont les lignes 2 à 8 du scénario nominal, dédoublonnées. La liste ne sort pas d'un catalogue technique : **elle sort de votre analyse.**

D'où le critère qui vous servira au TP : si votre interface compte dix opérations, c'est que vous décrivez la libc et non votre scénario.

Une interface qui expose tout n'abstrait rien.

Une exigence non fonctionnelle qui décide de l'architecture

⇒ Fil rouge — snapguard : **ENF-MAIN-01**

« La logique métier doit être testable sans accès au système de fichiers réel. »

Le problème concret

Un lien dur ne peut pas franchir la frontière d'un système de fichiers : notre outil devra gérer ce cas. Mais si la fonction de sauvegarde appelle directement `link()`, **comment tester ce cas ?**

Il faudrait un second système de fichiers, des droits root, un montage dédié. Pour *un* test. Le test serait lent, fragile, et personne ne l'écrirait.

Une ENF de **maintenabilité** vient de déterminer toute l'architecture. Elle ne sert aucun utilisateur — elle sert l'équipe.

Une exigence non fonctionnelle qui décide de l'architecture

⇒ Fil rouge — snapguard : **ENF-MAIN-01**

« La logique métier doit être testable sans accès au système de fichiers réel. »

Le problème concret

Un lien dur ne peut pas franchir la frontière d'un système de fichiers : notre outil devra gérer ce cas. Mais si la fonction de sauvegarde appelle directement `link()`, **comment tester ce cas ?**

Il faudrait un second système de fichiers, des droits root, un montage dédié. Pour *un* test. Le test serait lent, fragile, et personne ne l'écrirait.

Une ENF de **maintenabilité** vient de déterminer toute l'architecture. Elle ne sert aucun utilisateur — elle sert l'équipe.

Une exigence non fonctionnelle qui décide de l'architecture

⇒ Fil rouge — snapguard : **ENF-MAIN-01**

« *La logique métier doit être testable sans accès au système de fichiers réel.* »

Le problème concret

Un lien dur ne peut pas franchir la frontière d'un système de fichiers : notre outil devra gérer ce cas. Mais si la fonction de sauvegarde appelle directement `link()`, **comment tester ce cas ?**

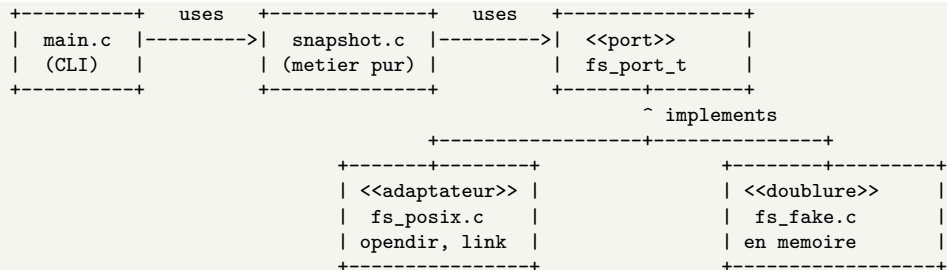
Il faudrait un second système de fichiers, des droits root, un montage dédié. Pour *un* test. Le test serait lent, fragile, et personne ne l'écrirait.

Une ENF de **maintenabilité** vient de déterminer toute l'architecture. Elle ne sert aucun utilisateur — elle sert l'équipe.

Le principe d'inversion des dépendances (le « D » de SOLID)

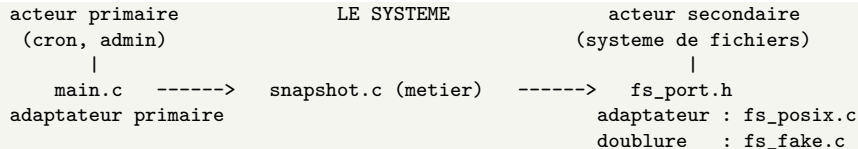
Robert C. Martin

« Les modules de haut niveau ne doivent pas dépendre des modules de bas niveau ; les deux doivent dépendre d'abstractions. »



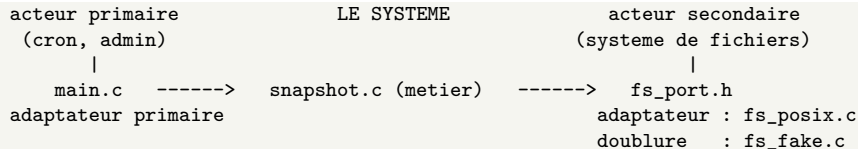
Le métier ne dépend ni de POSIX, ni de macOS : il dépend d'un **contrat**.

Deux sortes d'acteurs, deux sortes de ports



C'est de là que vient le nom de l'architecture : **ports et adaptateurs**.
Un port par frontière, un adaptateur par technologie qui s'y branche.

Deux sortes d'acteurs, deux sortes de ports



C'est de là que vient le nom de l'architecture : **ports et adaptateurs**.
Un port par frontière, un adaptateur par technologie qui s'y branche.

Deux décisions distinctes, qu'il ne faut pas confondre

Le cas d'utilisation dit *où sont les frontières*

Il fait apparaître deux interfaces : celle de l'acteur primaire, celle de l'acteur secondaire. À ce stade, rien n'impose encore le sens de la dépendance — le métier pourrait très bien appeler `link()` lui-même.

L'exigence non fonctionnelle dit *de quel côté va la dépendance*

ENF-MAIN-01 — *la logique métier doit être testable sans accès au système de fichiers réel* —
tranche : c'est le métier qui dépendra du contrat abstrait, et l'implémentation POSIX qui viendra s'y brancher.

Frontière et sens de la dépendance sont deux questions séparées. L'analyse répond à la première, la qualité attendue à la seconde.

Deux décisions distinctes, qu'il ne faut pas confondre

Le cas d'utilisation dit *où sont les frontières*

Il fait apparaître deux interfaces : celle de l'acteur primaire, celle de l'acteur secondaire. À ce stade, rien n'impose encore le sens de la dépendance — le métier pourrait très bien appeler `link()` lui-même.

L'exigence non fonctionnelle dit *de quel côté va la dépendance*

ENF-MAIN-01 — *la logique métier doit être testable sans accès au système de fichiers réel* —
tranche : c'est le métier qui dépendra du contrat abstrait, et l'implémentation POSIX qui viendra s'y brancher.

Frontière et sens de la dépendance sont deux questions séparées. L'analyse répond à la première, la qualité attendue à la seconde.

Deux décisions distinctes, qu'il ne faut pas confondre

Le cas d'utilisation dit *où sont les frontières*

Il fait apparaître deux interfaces : celle de l'acteur primaire, celle de l'acteur secondaire. À ce stade, rien n'impose encore le sens de la dépendance — le métier pourrait très bien appeler `link()` lui-même.

L'exigence non fonctionnelle dit *de quel côté va la dépendance*

ENF-MAIN-01 — *la logique métier doit être testable sans accès au système de fichiers réel* —
tranche : c'est le métier qui dépendra du contrat abstrait, et l'implémentation POSIX qui viendra s'y brancher.

Frontière et **sens de la dépendance** sont deux questions séparées. L'analyse répond à la première, la qualité attendue à la seconde.

- **Testabilité** — on injecte une doublure à la place du vrai système de fichiers. C'est ce qui rend le test du lien impossible. . . possible.
- **Évolutivité** — changer de mécanisme de stockage ne touche qu'une classe d'implémentation, jamais la logique métier ni les tests.
- **Découplage** — les composants évoluent indépendamment.

Pour mémoire : SOLID

Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion.

- **Testabilité** — on injecte une doublure à la place du vrai système de fichiers. C'est ce qui rend le test du lien impossible. . . possible.
- **Évolutivité** — changer de mécanisme de stockage ne touche qu'une classe d'implémentation, jamais la logique métier ni les tests.
- **Découplage** — les composants évoluent indépendamment.

Pour mémoire : SOLID

Single Responsibility, **O**pen/Closed, **L**iskov Substitution, **I**nterface Segregation, **D**ependency Inversion.

Les principes de conception ne sont pas des propriétés du langage.

```
typedef struct fs_port {  
    void *ctx;                                /* l'etat de l'implementation : le "this" */  
    int (*lister)(void *ctx, const char *chemin, entree_t *tab, size_t max);  
    int (*stat) (void *ctx, const char *chemin, entree_t *info);  
    int (*copier)(void *ctx, const char *src, const char *dst);  
    /* ... */  
} fs_port_t;  
  
const fs_port_t *fs_posix(void);  /* le SEUL symbole exporte par l'adaptateur */
```

C'est exactement ce que fait le noyau Linux avec ses struct file_operations, et la libc avec ses FILE*. Les fonctions de l'adaptateur sont static : invisibles hors de leur unité de compilation. **Encapsulation réelle, sans mot-clé dédié.**

Les principes de conception ne sont pas des propriétés du langage.

```
typedef struct fs_port {  
    void *ctx;                                /* l'etat de l'implementation : le "this" */  
    int (*lister)(void *ctx, const char *chemin, entree_t *tab, size_t max);  
    int (*stat) (void *ctx, const char *chemin, entree_t *info);  
    int (*copier)(void *ctx, const char *src, const char *dst);  
    /* ... */  
} fs_port_t;  
  
const fs_port_t *fs_posix(void);  /* le SEUL symbole exporte par l'adaptateur */
```

C'est exactement ce que fait le noyau Linux avec ses struct file_operations, et la libc avec ses FILE*. Les fonctions de l'adaptateur sont static : invisibles hors de leur unité de compilation. **Encapsulation réelle, sans mot-clé dédié.**

Concept objet	Traduction en C
Interface	struct de pointeurs de fonctions
Méthode abstraite	un champ <code>int (*copier)(void *ctx, ...)</code>
Classe qui implémente	un <code>.c</code> avec des fonctions <code>static</code> + un constructeur
L'état de l'objet (<code>this</code>)	le champ <code>void *ctx</code> , premier argument
Injection de dépendance	le port passé en paramètre de la fonction métier
Polymorphisme	déréférencement du pointeur de fonction à l'exécution

👉 TD2 — Formaliser les besoins de BioSeq

Correction croisée de vos stories avec INVEST, **débat** sur la story « lien dur », rédaction des critères Gherkin de **US-01**, puis rédaction collective du **scénario nominal**.

Production finale : la liste des opérations système que votre scénario réclame.

👉 TP2 — Coder ce contrat

Vous transformez cette liste en `fs_port.h` (au plus six opérations, aucun en-tête système), puis vous écrivez l'adaptateur POSIX — le *seul* fichier du projet autorisé à contenir des appels système.

Vérification mécanique : `make verif` échoue si un `opendir()` traîne ailleurs.

👉 TD2 — Formaliser les besoins de BioSeq

Correction croisée de vos stories avec INVEST, **débat** sur la story « lien dur », rédaction des critères Gherkin de **US-01**, puis rédaction collective du **scénario nominal**.

Production finale : la liste des opérations système que votre scénario réclame.

👉 TP2 — Coder ce contrat

Vous transformez cette liste en `fs_port.h` (au plus six opérations, aucun en-tête système), puis vous écrivez l'adaptateur POSIX — le *seul* fichier du projet autorisé à contenir des appels système.

Vérification mécanique : `make verif` échoue si un `opendir()` traîne ailleurs.

Rédigez **deux User Stories** pour BioSeq, avec des **rôles différents**, au format
« *En tant que . . . , je veux . . . , afin de . . .* »

Ne cherchez pas la perfection : elles seront corrigées entre vous, avec INVEST, en ouverture du TD.

- ❶ Une *User Story* porte le **pourquoi** : rôle précis, action, bénéfice métier. Elle est courte *volontairement*.
- ❷ Les « 3 C » : Card, Conversation, Confirmation. La richesse est dans la conversation et les critères d'acceptation.
- ❸ INVEST relit une story ; une story trop grosse est une *Epic* à découper.
- ❹ Un critère d'acceptation Gherkin ne mentionne aucun élément d'implémentation : **la spécification devient le test**.
- ❺ Le cas d'utilisation porte le **comment** de l'interaction, au « niveau de la mer ».
- ❻ Le scénario nominal dicte les opérations dont le métier a besoin — donc l'**interface**. Une ENF de maintenabilité impose l'inversion des dépendances.