

Analyse : de l'exigence à la spécification

CM3 — Ce qui peut mal tourner, et comment le vérifier

Ressource R3.03

BUT2 Informatique

15 septembre 2026

CM1 / TD1 / TP1	le besoin est identifié, les exigences sont mesurables
CM2 / TD2	les stories et le scénario nominal sont écrits
TP2	le port et l'adaptateur POSIX sont codés

Aujourd'hui	le chemin heureux ne représente qu'une fraction de la réalité. Que se passe-t-il quand ça se passe mal — et comment le vérifie-t-on ?
--------------------	--

#	Partie	Durée
1	Les scénarios d'exception : là où est la valeur du CU	25 min
2	Le contrat : préconditions et postconditions	10 min
3	La pyramide des tests	15 min
4	Les doublures : <i>stub</i> , <i>mock</i> , <i>fake</i>	25 min

Le scénario nominal, tout le monde le trouve.

Et tout le monde écrit à peu près le même.

Alors où est la difficulté ?

Dans ce qui n'arrive pas comme prévu. Un cas d'utilisation n'est complet que s'il anticipe les cas alternatifs et les erreurs. **Oublier ces scénarios est l'erreur classique qui mène à des applications fragiles et imprévisibles.**

Le scénario nominal, tout le monde le trouve.

Et tout le monde écrit à peu près le même.

Alors où est la difficulté ?

Dans ce qui n'arrive pas comme prévu. Un cas d'utilisation n'est complet que s'il anticipe les cas alternatifs et les erreurs. **Oublier ces scénarios est l'erreur classique qui mène à des applications fragiles et imprévisibles.**

Le scénario nominal, tout le monde le trouve.

Et tout le monde écrit à peu près le même.

Alors où est la difficulté ?

Dans ce qui n'arrive pas comme prévu. Un cas d'utilisation n'est complet que s'il anticipe les cas alternatifs et les erreurs. **Oublier ces scénarios est l'erreur classique qui mène à des applications fragiles et imprévisibles.**

7a. La creation du lien echoue (depot sur un autre systeme de fichiers).
Le systeme copie le fichier au lieu de le lier et poursuit le parcours.
L'instantane reste valide ; seul l'espace consomme augmente.
Reprise a l'etape 8.

- numéroté d'après **l'étape du nominal** où l'incident survient ;
- décrit ce que fait le système, pas ce que devrait faire le développeur ;
- se termine *toujours* par « **reprise à l'étape n** » ou « **le cas d'utilisation se termine** ».

Comment les trouver ? En interrogeant chaque étape

Question	Ce qu'elle fait apparaître
Et si la ressource n'existe pas ?	premier passage, fichier nouveau, compte inconnu
Et si l'acteur n'a pas le droit ?	permission refusée, session expirée
Et si la ressource disparaît en cours ?	volume démonté, fichier supprimé
Et si une limite est atteinte ?	disque plein, chemin trop long, quota
Et si un système externe refuse ?	paiement refusé, API indisponible
Et si deux choses arrivent en même temps ?	fichier modifié pendant la copie

Ce n'est pas de l'imagination : c'est une **grille systématique**. On la déroule sur chaque étape du nominal.

Comment les trouver ? En interrogeant chaque étape

Question	Ce qu'elle fait apparaître
Et si la ressource n'existe pas ?	premier passage, fichier nouveau, compte inconnu
Et si l'acteur n'a pas le droit ?	permission refusée, session expirée
Et si la ressource disparaît en cours ?	volume démonté, fichier supprimé
Et si une limite est atteinte ?	disque plein, chemin trop long, quota
Et si un système externe refuse ?	paiement refusé, API indisponible
Et si deux choses arrivent en même temps ?	fichier modifié pendant la copie

Ce n'est pas de l'imagination : c'est une **grille systématique**. On la déroule sur chaque étape du nominal.

-
- A Le dépôt est plein au milieu du parcours.
 - B Un fichier de la source est illisible (permission refusée).
 - C Il n'existe aucun instantané antérieur : première sauvegarde.
 - D Le dépôt est sur un autre disque : impossible d'y créer un lien.
 - E Le volume source est démonté pendant l'exécution.
 - F Un chemin dépasse la taille du tampon prévu.
 - G Un fichier est modifié *pendant* que le système le copie.
 - H Un fichier existe dans la source mais pas dans l'instantané de référence.
-

Vous avez *constaté au clavier* la carte D lors du TP1 : un lien dur ne franchit pas la frontière d'un système de fichiers. Ce n'est pas une hypothèse d'école.

-
- A Le dépôt est plein au milieu du parcours.
 - B Un fichier de la source est illisible (permission refusée).
 - C Il n'existe aucun instantané antérieur : première sauvegarde.
 - D Le dépôt est sur un autre disque : impossible d'y créer un lien.
 - E Le volume source est démonté pendant l'exécution.
 - F Un chemin dépasse la taille du tampon prévu.
 - G Un fichier est modifié *pendant* que le système le copie.
 - H Un fichier existe dans la source mais pas dans l'instantané de référence.
-

Vous avez *constaté au clavier* la carte D lors du TP1 : un lien dur ne franchit pas la frontière d'un système de fichiers. Ce n'est pas une hypothèse d'école.

Deux de ces cartes ne sont pas des questions techniques

Carte D — le lien est impossible

Faut-il **échouer** ou se **replier sur la copie** ?

Décision retenue chez BioSeq : la disponibilité de la sauvegarde prime sur l'économie d'espace. On copie, on poursuit, l'instantané reste valide.

Carte B — un fichier est illisible

Faut-il tout arrêter pour un seul fichier ? Si l'on poursuit, **comment l'instantané obtenu se distingue-t-il d'un instantané complet** ?

*C'est de cette question que naît la notion de **statut** — nous y reviendrons au CM4.*

Ces deux décisions appartiennent au **client**, pas au développeur. Elles se documentent, elles ne s'improvisent pas au clavier à 23 h.

Carte D — le lien est impossible

Faut-il **échouer** ou se **replier sur la copie** ?

Décision retenue chez BioSeq : la disponibilité de la sauvegarde prime sur l'économie d'espace. On copie, on poursuit, l'instantané reste valide.

Carte B — un fichier est illisible

Faut-il tout arrêter pour un seul fichier ? Si l'on poursuit, **comment l'instantané obtenu se distingue-t-il d'un instantané complet** ?

*C'est de cette question que naît la notion de **statut** — nous y reviendrons au CM4.*

Ces deux décisions appartiennent au **client**, pas au développeur. Elles se documentent, elles ne s'improvisent pas au clavier à 23 h.

Carte D — le lien est impossible

Faut-il **échouer** ou se **replier sur la copie** ?

Décision retenue chez BioSeq : la disponibilité de la sauvegarde prime sur l'économie d'espace. On copie, on poursuit, l'instantané reste valide.

Carte B — un fichier est illisible

Faut-il tout arrêter pour un seul fichier ? Si l'on poursuit, **comment l'instantané obtenu se distingue-t-il d'un instantané complet** ?

*C'est de cette question que naît la notion de **statut** — nous y reviendrons au CM4.*

Ces deux décisions appartiennent au **client**, pas au développeur. Elles se documentent, elles ne s'improvisent pas au clavier à 23 h.

Un cas d'utilisation est un **contrat** :
si vous m'appellez dans ces conditions, je vous garantis ce résultat.

⇒ Fil rouge — snapguard : CU-01

Préconditions — la source existe et est un répertoire lisible ; le dépôt est accessible en écriture.

Postcondition de succès — un nouvel instantané complet existe ; il reflète l'état de la source ; *les instantanés antérieurs sont inchangés.*

Postcondition d'échec — aucun instantané complet n'a été ajouté ; *les instantanés antérieurs sont inchangés* ; l'incident est journalisé.

La **postcondition d'échec** est la **plus importante** — et la plus souvent oubliée. Retenez-la : votre propre code la violera au TP4.

Un cas d'utilisation est un **contrat** :
si vous m'appellez dans ces conditions, je vous garantis ce résultat.

⇒ Fil rouge — snapguard : CU-01

Préconditions — la source existe et est un répertoire lisible ; le dépôt est accessible en écriture.

Postcondition de succès — un nouvel instantané complet existe ; il reflète l'état de la source ; *les instantanés antérieurs sont inchangés.*

Postcondition d'échec — aucun instantané complet n'a été ajouté ; *les instantanés antérieurs sont inchangés* ; l'incident est journalisé.

La postcondition d'échec est la plus importante — et la plus souvent oubliée. Retenez-la : votre propre code la violera au TP4.

US-01
1 phrase



CU-01
9 étapes
+ 8 scénarios d'exception

Ce que cet écart signifie

La story et le cas d'utilisation ne s'opposent pas et ne se remplacent pas. La story garde l'équipe concentrée sur la valeur ; le cas d'utilisation **réduit l'ambiguïté pour les développeurs et les testeurs** et force à anticiper les cas limites.

Les huit exceptions ci-dessus, personne ne les aurait trouvées spontanément au clavier.

US-01
1 phrase



CU-01
9 étapes
+ 8 scénarios d'exception

Ce que cet écart signifie

La story et le cas d'utilisation ne s'opposent pas et ne se remplacent pas. La story garde l'équipe concentrée sur la valeur ; le cas d'utilisation **réduit l'ambiguïté pour les développeurs et les testeurs** et force à anticiper les cas limites.

Les huit exceptions ci-dessus, personne ne les aurait trouvées spontanément au clavier.

Niveau	Quantité	Caractéristiques
Acceptation (<i>sommet</i>)	peu	lents, coûteux, proches du métier
Intégration (<i>milieu</i>)	moyenne	vérifient la collaboration des briques
Unitaires (<i>base</i>)	beaucoup	très rapides, isolés, peu coûteux

Pourquoi une pyramide et non un rectangle ?

Un test unitaire s'exécute en quelques millisecondes et **pointe la ligne fautive** : on peut en écrire des milliers. Un test de bout en bout est lent et fragile ; en abuser produit une suite interminable et pénible à maintenir.

Mike Cohn, *Succeeding with Agile*, 2009. L'anti-patron inverse porte un nom : le *cornet de glace*.

Niveau	Quantité	Caractéristiques
Acceptation (<i>sommet</i>)	peu	lents, coûteux, proches du métier
Intégration (<i>milieu</i>)	moyenne	vérifient la collaboration des briques
Unitaires (<i>base</i>)	beaucoup	très rapides, isolés, peu coûteux

Pourquoi une pyramide et non un rectangle ?

Un test unitaire s'exécute en quelques millisecondes et **pointe la ligne fautive** : on peut en écrire des milliers. Un test de bout en bout est lent et fragile ; en abuser produit une suite interminable et pénible à maintenir.

Mike Cohn, *Succeeding with Agile*, 2009. L'anti-patron inverse porte un nom : le *cornet de glace*.

⇒ Fil rouge — snapguard : la suite de tests de snapguard

Niveau	Durée	Ce qu'il prouve
Unitaire	< 10 ms	les règles de gestion sont correctes
Intégration	0,2 s	l' adaptateur POSIX tient la promesse du port
Acceptation	1 s	la valeur promise par US-01 est au rendez-vous

Le test d'acceptation est directement issu des critères Gherkin du CM2.
La spécification est le test.

⇒ Fil rouge — snapguard : la suite de tests de snapguard

Niveau	Durée	Ce qu'il prouve
Unitaire	< 10 ms	les règles de gestion sont correctes
Intégration	0,2 s	l' adaptateur POSIX tient la promesse du port
Acceptation	1 s	la valeur promise par US-01 est au rendez-vous

Le test d'acceptation est directement issu des critères Gherkin du CM2.

La spécification est le test.

Le problème : comment tester la carte D ?

Rappel de l'incident

Le dépôt est sur un autre disque : la création du lien dur échoue, et le système doit se replier sur la copie.

Pour le tester « pour de vrai », il faudrait : un second système de fichiers, des droits root, un montage dédié.

Pour la carte A : un disque réellement plein. Pour la carte E : démonter un volume en plein parcours.

La solution vient de l'architecture du CM2

Le métier ne parle au système que par le **port**. Il suffit donc de lui fournir **une autre implémentation de ce port** — une arborescence en mémoire, qui échoue quand on le lui demande.

Le problème : comment tester la carte D ?

Rappel de l'incident

Le dépôt est sur un autre disque : la création du lien dur échoue, et le système doit se replier sur la copie.

Pour le tester « pour de vrai », il faudrait : un second système de fichiers, des droits root, un montage dédié.

Pour la carte A : un disque réellement plein. Pour la carte E : démonter un volume en plein parcours.

La solution vient de l'architecture du CM2

Le métier ne parle au système que par le **port**. Il suffit donc de lui fournir **une autre implémentation de ce port** — une arborescence en mémoire, qui échoue quand on le lui demande.

Le problème : comment tester la carte D ?

Rappel de l'incident

Le dépôt est sur un autre disque : la création du lien dur échoue, et le système doit se replier sur la copie.

Pour le tester « pour de vrai », il faudrait : un second système de fichiers, des droits root, un montage dédié.

Pour la carte A : un disque réellement plein. Pour la carte E : démonter un volume en plein parcours.

La solution vient de l'architecture du CM2

Le métier ne parle au système que par le **port**. Il suffit donc de lui fournir **une autre implémentation de ce port** — une arborescence en mémoire, qui échoue quand on le lui demande.

Stub fournit des réponses préprogrammées. « *Quand on te demande ce fichier, renvoie toujours ces métadonnées.* » Il sert à **contrôler les entrées** du code testé.

Mock vérifie les **interactions**. On lui fixe des attentes : « *la méthode `lier_dur` doit être appelée une fois, avec cet argument* ». Le test échoue si elles ne sont pas respectées.

Fake une implémentation **simplifiée mais fonctionnelle** : une base de données en mémoire, un système de fichiers en mémoire.

Stub : vérifie un *état* — « ai-je reçu la bonne valeur ? »

Mock : vérifie un *comportement* — « la bonne méthode a-t-elle été appelée ? »

Confondre les deux, c'est écrire des tests fragiles qui cassent au moindre changement d'implémentation.

Stub fournit des réponses préprogrammées. « *Quand on te demande ce fichier, renvoie toujours ces métadonnées.* » Il sert à **contrôler les entrées** du code testé.

Mock vérifie les **interactions**. On lui fixe des attentes : « *la méthode `lier_dur` doit être appelée une fois, avec cet argument* ». Le test échoue si elles ne sont pas respectées.

Fake une implémentation **simplifiée mais fonctionnelle** : une base de données en mémoire, un système de fichiers en mémoire.

Stub : vérifie un *état* — « ai-je reçu la bonne valeur ? »

Mock : vérifie un *comportement* — « la bonne méthode a-t-elle été appelée ? »

Confondre les deux, c'est écrire des tests fragiles qui cassent au moindre changement d'implémentation.

La doublure de snapguard

```
typedef struct {
    noeud_t noeuds[64];      /* une arborescence en memoire      */
    int      nb_noeuds;
    char     journal[128][MAX_CHEMIN]; /* trace de chaque ecriture */
    int      nb_journal;
    int      echec_lien_dur; /* 1 => lier_dur() echoue : carte D ! */
} fake_fs_t;
```

C'est un **fake** (arborescence fonctionnelle en mémoire) qui joue aussi le rôle de **mock** grâce à son journal.

Le champ `echec_lien_dur` rend la carte D testable en trois lignes, sans root et sans second disque.

La doublure de snapguard

```
typedef struct {  
    noeud_t noeuds[64];      /* une arborescence en memoire      */  
    int      nb_noeuds;  
    char     journal[128][MAX_CHEMIN]; /* trace de chaque ecriture */  
    int      nb_journal;  
    int      echec_lien_dur; /* 1 => lier_dur() echoue : carte D ! */  
} fake_fs_t;
```

C'est un **fake** (arborescence fonctionnelle en mémoire) qui joue aussi le rôle de **mock** grâce à son journal.

Le champ `echec_lien_dur` rend la carte D testable en trois lignes, sans root et sans second disque.

Assertion	Nature	Ce qu'elle vérifie
<code>r.fichiers_lies == 2</code>	<i>état</i>	le résultat du calcul
<code>journal_contient("LINK /depot/j1/...")</code>	<i>interaction</i>	que le lien pointe vers le bon instantané
<code>compter("MKDIR") == 0</code>	<i>absence</i>	qu'une source invalide n'a produit aucun effet de bord

La troisième est la plus intéressante

Vérifier une **absence** — « le programme n'a rien écrit » — est presque impossible proprement sans doublure : il faudrait inspecter tout un disque pour prouver qu'il ne s'est rien passé.

Assertion	Nature	Ce qu'elle vérifie
<code>r.fichiers_lies == 2</code>	<i>état</i>	le résultat du calcul
<code>journal_contient("LINK /depot/j1/...")</code>	<i>interaction</i>	que le lien pointe vers le bon instantané
<code>compter("MKDIR") == 0</code>	<i>absence</i>	qu'une source invalide n'a produit aucun effet de bord

La troisième est la plus intéressante

Vérifier une **absence** — « le programme n'a rien écrit » — est presque impossible proprement sans doublure : il faudrait inspecter tout un disque pour prouver qu'il ne s'est rien passé.

Une règle de gestion doit être une fonction nommée

⇒ Fil rouge — snapguard : RG-02

Un fichier est réputé inchangé si sa taille et sa date de modification sont identiques à celles de l'instantané précédent.

```
/* RG-02 : detection d'un fichier inchangé depuis l'instantané précédent. */  
static int est_inchange(const entree_t *courant, const entree_t *ancien)  
{ ... }
```

Pourquoi ne pas l'écrire en ligne dans la boucle ?

Pour le test : on teste une fonction, pas une condition noyée dans une boucle.

Pour l'évolution : le jour où Karim demandera une détection par empreinte, tout le monde saura où modifier.

Toute règle de gestion identifiée en analyse doit être retrouvable dans le code sous la forme d'une fonction portant son nom.

Une règle de gestion doit être une fonction nommée

⇒ Fil rouge — snapguard : RG-02

Un fichier est réputé inchangé si sa taille et sa date de modification sont identiques à celles de l'instantané précédent.

```
/* RG-02 : detection d'un fichier inchangé depuis l'instantané précédent. */  
static int est_inchange(const entree_t *courant, const entree_t *ancien)  
{ ... }
```

Pourquoi ne pas l'écrire en ligne dans la boucle ?

Pour le test : on teste une fonction, pas une condition noyée dans une boucle.

Pour l'évolution : le jour où Karim demandera une détection par empreinte, tout le monde saura où modifier.

Toute règle de gestion identifiée en analyse doit être retrouvable dans le code sous la forme d'une fonction portant son nom.

RG-02 est un compromis, et il faut le dire

Un fichier modifié, restauré à sa taille d'origine, avec touch pour remettre la date, **passerait inaperçu**.
La détection sûre (empreinte SHA-256) coûte une lecture intégrale de 400 Gio chaque nuit.

Karim a tranché : **taille + date**, avec une option de vérification complète mensuelle.

L'analyse n'est pas la recherche de la solution parfaite.
C'est un arbitrage coût / risque, explicite et documenté.

RG-02 est un compromis, et il faut le dire

Un fichier modifié, restauré à sa taille d'origine, avec touch pour remettre la date, **passerait inaperçu**.
La détection sûre (empreinte SHA-256) coûte une lecture intégrale de 400 Gio chaque nuit.

Karim a tranché : **taille + date**, avec une option de vérification complète mensuelle.

L'analyse n'est pas la recherche de la solution parfaite.
C'est un arbitrage coût / risque, explicite et documenté.

RG-02 est un compromis, et il faut le dire

Un fichier modifié, restauré à sa taille d'origine, avec touch pour remettre la date, **passerait inaperçu**.
La détection sûre (empreinte SHA-256) coûte une lecture intégrale de 400 Gio chaque nuit.

Karim a tranché : **taille + date**, avec une option de vérification complète mensuelle.

L'analyse n'est pas la recherche de la solution parfaite.
C'est un arbitrage coût / risque, explicite et documenté.

TD3 — Le jeu des pannes

Chaque groupe tire **trois cartes incident** et rédige le flux alternatif correspondant. Mise en commun, puis arbitrage collectif sur les cartes B et D.

Puis vous concevez sur papier la **doublure** qui rendra ces exceptions testables.

TP3 — Coder les décisions du TD

RG-02 comme fonction nommée, les liens durs, le repli de la carte D, les replis des cartes B et F.

Puis le **laboratoire de mesures** : ls -i et du prouvent, chiffres à l'appui, que la promesse de **US-01** est tenue.

👉 TD3 — Le jeu des pannes

Chaque groupe tire **trois cartes incident** et rédige le flux alternatif correspondant. Mise en commun, puis arbitrage collectif sur les cartes B et D.

Puis vous concevez sur papier la **doublure** qui rendra ces exceptions testables.

👉 TP3 — Coder les décisions du TD

RG-02 comme fonction nommée, les liens durs, le repli de la carte D, les replis des cartes B et F.

Puis le **laboratoire de mesures** : ls -i et du prouvent, chiffres à l'appui, que la promesse de **US-01** est tenue.

- ❶ Le scénario nominal est facile ; **la valeur d'un cas d'utilisation est dans ses scénarios d'exception.**
- ❷ On les trouve par une **grille systématique**, pas par imagination.
- ❸ Certaines exceptions appellent une **décision métier** : elle appartient au client et se documente.
- ❹ Un cas d'utilisation est un contrat : préconditions, postcondition de succès, et surtout **postcondition d'échec.**
- ❺ Pyramide des tests : beaucoup d'unitaires, peu de bout en bout.
- ❻ Une doublure rend testable ce qui ne l'était pas. *Stub* : un état. *Mock* : une interaction. *Fake* : une implémentation simplifiée.
- ❼ Une règle de gestion se code sous la forme d'une **fonction portant son nom.**