

Cahier d'analyse snapguard

Document enseignant — les quatre séquences et le guide pédagogique

Ressource R3.03 — BUT2 Informatique

Langage de mise en œuvre : C (POSIX), Linux / macOS

September 1, 2026

Abstract

Ce document déroule intégralement l'analyse du besoin du laboratoire *BioSeq* et sa mise en œuvre en C, en suivant le plan du cours : enquête, exigences, *User Stories*, cas d'utilisation, modèle de domaine, processus métier, conception, implémentation et tests. Il est organisé en **quatre séquences**, chacune correspondant à un triplet CM / TD / TP, et se termine par le guide pédagogique.

Il ne se distribue pas aux étudiants : il contient les corrigés de l'ensemble des activités.

Contents

Séquence 1 — Du besoin aux exigences	2
1 L'enquête : de la demande au besoin	2
1.1 La demande initiale	2
1.2 Extrait de l'entretien	2
1.3 Les « 5 Pourquoi »	2
1.4 Ce que l'enquête a fait apparaître	3
2 Les exigences	3
2.1 Glossaire du domaine (le langage ubiquitaire)	3
2.2 Règles de gestion	3
2.3 Exigences fonctionnelles	4
2.4 Exigences non fonctionnelles	4
2.5 L'exercice de bascule EF ↔ ENF	5
2.6 Hors périmètre (v1)	5
3 Notes pour la conduite des trois séances	5
3.1 CM1	5
3.2 TD1	5
3.3 TP1	6
Séquence 2 — Formaliser	7
4 Formalisation	7
4.1 Acteurs	7
4.2 Epics et User Stories	7
4.3 US-01 au filtre INVEST	8
4.4 Critères d'acceptation de US-01 (format Gherkin)	8
4.5 Diagramme de cas d'utilisation	8
4.6 CU-01 — Créer un instantané	8

5	Du modèle de domaine au modèle de conception	9
5.1	Le choix d'architecture : ports et adaptateurs	9
5.2	Comment écrit-on une interface en C ?	9
5.3	Correspondance domaine → conception	10
6	Le prototype	10
6.1	Le port : <code>src/fs_port.h</code>	10
6.2	Le métier : <code>src/snapshot.h</code>	11
6.3	Le métier : <code>src/snapshot.c</code>	11
6.4	L'adaptateur POSIX : <code>src/fs_posix.c</code>	13
6.5	L'adaptateur CLI : <code>src/main.c</code>	14
6.6	Les points « système » à faire travailler en TP	15
7	Notes pour la conduite des trois séances	15
7.1	CM2	15
7.2	TD2	16
7.3	TP2	16
	Séquence 3 — Ce qui peut mal tourner	17
8	Les scénarios d'exception de CU-01	17
9	La stratégie de test	17
9.1	Tests unitaires — sans jamais toucher au disque	17
9.2	La doublure : <code>tests/fs_fake.c</code>	19
10	Notes pour la conduite des trois séances	22
10.1	CM3	22
10.2	TD3	22
10.3	TP3	22
	Séquence 4 — Modéliser et éprouver	23
11	Le modèle de domaine	23
11.1	Étape 1 : extraction des noms	23
11.2	Étape 2 : raffinage — le travail de tri	23
11.3	Étape 3 : associations et multiplicités	23
11.4	Le diagramme (source PlantUML)	24
11.5	Ce que le modèle ne contient pas — et pourquoi	24
12	Les processus métier	24
12.1	Identification	24
12.2	P1 — Sauvegarde nocturne	24
12.3	P2 — Restauration d'urgence	25
12.4	Ce que les processus ont ajouté au modèle de domaine	25
12.5	L'automate à états de l'Instantané	25
12.6	Tests d'intégration	26
12.7	Tests d'acceptation	27
12.8	Le Makefile	28
12.9	Exécution réelle de la suite complète	28
12.10	La démonstration à faire en amphi (5 minutes, effet garanti)	29
13	Ce qu'il reste à faire : le backlog	30

14 Notes pour la conduite des trois séances	31
14.1 CM4	31
14.2 TD4	31
14.3 TP4	31
Guide pédagogique	32
15 Matrice de traçabilité	32
16 Progression : quatre paires TD/TP de 1 h 15	32
17 Le travail en groupe de 3 ou 4	33
18 Évaluation : trois notes de groupe, une note individuelle	33
19 Exercices étudiants, calqués sur ceux du cours	35
20 Grille d'évaluation d'un dossier d'analyse	36
21 Un mot sur le choix du C	36

Séquence 1 — Du besoin aux exigences vérifiables

Ce que couvre cette séquence

CM1	pourquoi l'analyse ; demande vs besoin ; les 5 Pourquoi ; EF et ENF ; ISO 25010
TD1	confrontation des lectures, jeu de rôle de l'entretien, rédaction des exigences
TP1	laboratoire des liens durs, <i>spike</i> de performance, socle du projet

Le contexte, rappel

Le laboratoire de recherche *BioSeq* (18 chercheurs, 4 doctorants) traite des jeux de données génomiques sur une dizaine de postes Linux et macOS. Il n'a pas de service informatique : Karim, ingénieur d'études, gère les machines « en plus » de son travail.

Le contexte complet, le courriel de Karim et la transcription intégrale de l'entretien constituent le **dossier client**, distribué aux étudiants avant le CM1.

1 L'enquête : de la demande au besoin

1.1 La demande initiale

Karim : « Il me faudrait un script qui fasse un `tar.gz` de `/data` toutes les nuits sur le NAS. Ça devrait être vite fait, non ? »

Trois mots trahissent une solution déjà choisie : *script*, *tar.gz*, *toutes les nuits*. Aucun ne dit **quel problème** est résolu. On enquête.

1.2 Extrait de l'entretien

Analyste : Racontez-moi la dernière fois où l'absence de sauvegarde vous a posé problème.

Karim : Il y a trois semaines. Une doctorante a lancé un script de nettoyage avec un mauvais chemin. Il a écrasé six mois de résultats intermédiaires. On s'en est aperçu... quatre jours plus tard.

Analyste : Qu'auriez-vous voulu pouvoir faire à ce moment-là ?

Karim : Récupérer le dossier tel qu'il était le jeudi précédent. Juste ce dossier, pas toute la machine.

Analyste : Et si vous aviez eu vos archives `tar.gz` de chaque nuit ?

Karim : ... Il aurait fallu que je sache dans laquelle chercher, que je la décompresse en entier — on parle de 400 Gio — pour en sortir un dossier. Et de toute façon je n'ai pas la place pour trente archives de 400 Gio. C'est bien pour ça que je n'en garde que deux, en fait.

Analyste : Donc aujourd'hui, votre profondeur d'historique réelle est de deux jours ?

Karim : (silence) Deux jours. Et l'incident a été découvert au bout de quatre.

1.3 Les « 5 Pourquoi »

1. *Pourquoi un `tar.gz` nocturne ?* — Pour avoir une copie des données.
2. *Pourquoi une copie ?* — Pour restaurer en cas de perte.
3. *Pourquoi cela n'a-t-il pas marché la dernière fois ?* — Parce que la perte a été détectée après l'écrasement des deux seules archives conservées.
4. *Pourquoi n'en garder que deux ?* — Parce que chaque archive fait la taille complète des données.
5. *Pourquoi une archive complète pour des données qui bougent peu ?* — Parce que `tar` ne sait pas faire autrement... **et c'est la cause racine.**

Cause racine

Le besoin n'est pas « faire une copie », c'est « **pouvoir revenir à l'état de n'importe quel fichier à n'importe quelle date des N derniers jours, avec l'espace disque dont on dispose** ». La contrainte d'espace n'est pas un détail d'implémentation : c'est *elle* qui fait échouer la solution demandée.

1.4 Ce que l'enquête a fait apparaître

#	Problème métier constaté	Preuve
P1	profondeur d'historique (2 j) inférieure au délai de détection d'un incident (4 j)	incident du 10 août
P2	restaurer un seul fichier impose de décompresser 400 Gio	déclaration de Karim
P3	personne ne vérifie que la sauvegarde de la nuit a réussi	question de fin d'entretien
P4	le parc est à la fois Linux et macOS	inventaire

✗ À ne pas faire : Le piège dans lequel il ne fallait pas tomber

Répondre « d'accord » à la demande initiale aurait produit un script de 15 lignes, livré en une heure, **techniquement irréprochable et métier-ment inutile** : il aurait reproduit exactement la situation qui a causé la perte de données. Le coût de l'erreur n'aurait été visible qu'au prochain incident — c'est-à-dire, selon Boehm, au tarif « production ».

2 Les exigences

2.1 Glossaire du domaine (le langage ubiquitaire)

Terme	Définition retenue
Source	Répertoire dont on veut préserver l'état.
Dépôt	Répertoire, sur un autre support, qui héberge les instantanés.
Instantané	Image complète et navigable d'une source à une date donnée.
Politique	Règles de sauvegarde d'une source : fréquence, dépôt, rétention, exclusions.
Rétention	Durée de conservation d'un instantané avant purge.
Restauration	Action de replacer un fichier ou un répertoire depuis un instantané.

Ce tableau n'est pas de la décoration : Karim disait « backup » pour désigner tantôt le dépôt, tantôt un instantané. Une heure de discussion a été économisée sur toute la suite du projet.

2.2 Règles de gestion

- **RG-01** — Un instantané est **immuable** : une fois terminé, aucune opération ne le modifie (seule la purge peut le supprimer entièrement).
- **RG-02** — Un fichier est réputé **inchangé** si sa **taille** et sa **date de modification** sont identiques à celles de l'instantané précédent.
- **RG-03** — Un fichier inchangé n'est pas recopié : l'instantané pointe vers l'exemplaire déjà présent dans le dépôt.
- **RG-04** — Un instantané interrompu n'est jamais présenté comme complet.
- **RG-05** — La purge ne peut jamais supprimer le dernier instantané complet d'une source.

🔗 Cas d'étude : Une règle de gestion est une décision, pas une évidence

RG-02 est un compromis, et il faut le dire au client : un fichier modifié puis 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 --verifier pour un contrôle mensuel complet.

Faites verbaliser ce choix par les étudiants. C'est le meilleur exemple qu'ils rencontreront de l'analyse comme arbitrage *coût / risque*, et non comme recherche de la solution parfaite.

2.3 Exigences fonctionnelles

Réf.	Exigence	Vérifiable par
EF-01	créer un instantané d'une source dans un dépôt	diff -r avec la source ne signale aucun écart
EF-02	ne recopier que les fichiers modifiés ou nouveaux depuis l'instantané précédent	le rapport indique n copies pour n fichiers modifiés
EF-03	produire un rapport : fichiers examinés, copiés, liés, en erreur	lecture de la sortie standard
EF-04	lister les instantanés d'une source, avec date et taille	commande snapguard liste
EF-05	restaurer un fichier ou un répertoire depuis un instantané choisi	comparaison octet à octet
EF-06	supprimer les instantanés dépassant la rétention de la politique	comptage après exécution
EF-07	exclure les chemins correspondant aux motifs d'exclusion	aucun *.tmp dans le dépôt
EF-08	signaler par un code de retour non nul tout instantané incomplet	echo \$?

2.4 Exigences non fonctionnelles

Passage en revue systématique des huit caractéristiques **ISO/IEC 25010**. Les cases vides sont des décisions, elles aussi.

Caractéristique	Réf.	Exigence	Métrique
Performance	ENF-PERF-01	un instantané incrémental de 500 000 fichiers dont 1 % modifiés s'achève en < 30 min	chronométrage
Performance	ENF-PERF-02	≤ 64 Mio de mémoire résidente quelle que soit la taille de la source	time -v
Fiabilité	ENF-FIA-01	une interruption ne rend jamais illisible un instantané antérieur	injection de panne
Fiabilité	ENF-FIA-02	≥ 99 % d'instantanés nocturnes réussis sur un mois glissant	journal
Sécurité	ENF-SEC-01	droits du dépôt $\leq$ droits de la source ; dépôt en 0700	stat
Sécurité	ENF-SEC-02	aucune donnée ne transite par un service tiers non maîtrisé	revue d'architecture
Compatibilité	ENF-COMP-01	un instantané est exploitable avec ls, cp, diff, tar sans l'outil	scénario d'acceptation 3
Portabilité	ENF-PORT-01	compile et passe les tests sur Linux et macOS, sans directive conditionnelle hors POSIX.1-2008	CI sur 2 cibles
Utilisabilité	ENF-UTI-01	utilisable sans interaction depuis cron : stdout/stderr séparés, code de retour significatif	exécution en crontab
Maintenabilité	ENF-MAIN-01	logique métier testable sans accès au système de fichiers ; couverture ≥ 80 %	gcov

Cas d'étude : La caractéristique qu'on oublie toujours

ENF-COMP-01 est la plus importante du lot — et c'est celle qui n'apparaît jamais spontanément. Elle dit : *si notre outil disparaît, les données restent récupérables*. Une sauvegarde dans un format propriétaire illisible sans son logiciel est un piège classique du marché. C'est aussi ce qui

disqualifie définitivement plusieurs solutions envisageables, alors qu'aucune EF ne le faisait.

2.5 L'exercice de bascule EF ↔ ENF

Prenons : « le système doit calculer une empreinte cryptographique de chaque fichier ».

- Dans **snapguard** : l'utilisateur ne demande pas d'empreintes, il demande de retrouver ses fichiers. L'empreinte n'est qu'un *moyen de garantir la fiabilité* de la détection de changement → **ENF**.
- Dans un **outil de contrôle d'intégrité** (aide, tripwire) : produire et comparer les empreintes est la finalité du produit → **EF**.

Et l'inverse : « la sauvegarde doit s'exécuter chaque nuit à 2 h ». ENF de disponibilité ? Non — si l'outil embarque son propre ordonnanceur, la planification devient une fonctionnalité (EF). Ici, on a délibérément choisi de la déléguer à cron : elle **sort du périmètre**, ce qui est une troisième réponse possible, et souvent la meilleure.

2.6 Hors périmètre (v1)

Écrit noir sur blanc pour prévenir la dérive du périmètre — le mal qui a tué le VCF du FBI : pas de sauvegarde distante (SSH) ; pas de chiffrement ; pas d'interface graphique ; pas de gestion des liens symboliques, fichiers spéciaux, ACL ni attributs étendus ; pas de déduplication entre sources différentes.

3 Notes pour la conduite des trois séances

3.1 CM1

Le fil de la séance est celui du polycopié : le coût de Boehm, les quatre échecs, puis la distinction demande / besoin illustrée par les 5 Pourquoi de Karim. Terminez par les EF/ENF et le balayage ISO 25010 — c'est ce dont le TD1 a besoin, et rien de plus.

✓ À faire : Le moment à ne pas manquer

La classification du piège (« le système doit créer un lien dur plutôt qu'une copie »). Laissez la classe se diviser entre EF et ENF, puis tranchez : **ni l'une ni l'autre**, c'est une solution déguisée en exigence. Les étudiants n'ont pas encore fait le TP1 et ne savent pas ce qu'est un lien dur : c'est sans importance, ils jugent la *forme* de l'énoncé.

3.2 TD1

Déroulé en quatre temps : confrontation des surlignages (15 min), jeu de rôle (20 min), rédaction des exigences et balayage ISO 25010 (25 min), classification EF/ENF (15 min).

L'aide-mémoire pour jouer Karim et les huit énoncés à projeter sont dans le document *Matériel de séance*.

📖 Cas d'étude : Ce que le jeu de rôle produit vraiment

Karim répond **strictement** à ce qu'on lui demande. Une question fermée obtient « oui » ; une question suggestive obtient un acquiescement poli, et le groupe repart avec une fausse validation. C'est cette dernière situation qui marque le plus : ils comprennent qu'on peut sortir d'un entretien satisfait et complètement à côté.

3.3 TP1

Le TP1 n'est pas une séance de programmation : c'est un *spike*, une investigation technique destinée à réduire l'incertitude sur une exigence avant de s'engager. Le code produit au laboratoire est jetable ; ce qui reste, c'est la décision informée.

30 min	laboratoire d'observation : inodes, compteur de liens, les deux limites du lien dur
15 min	confrontation des ENF chiffrées du TD1 à une mesure réelle
30 min	socle du projet : dépôt, Makefile, parcours récursif

✓ À faire : La vérification de cinq minutes qui vaut pour tout le semestre

Passez dans les rangs et faites afficher `git config user.email` sur chaque poste. La note individuelle repose entièrement sur le journal Git : un dépôt à un seul auteur découvert au moment de noter est irrattrapable.

Vérifiez aussi que vous avez bien été ajouté aux dépôts (rôle *Maintainer* ou *Reporter*) : c'est demandé dans le travail préparatoire, tous ne l'auront pas fait.

✗ À ne pas faire : Ne donnez pas la réponse au laboratoire

La tentation est grande d'expliquer les liens durs avant de lancer l'exercice. Ne le faites pas : la question **Q2.4** demande d'expliquer le mécanisme à Karim *sans employer les mots* « *inode* » et « *lien dur* ». Elle n'a de sens que si la découverte a été la leur.

Séquence 2 — Formaliser : de la *story* à l'architecture

Ce que couvre cette séquence

CM2	<i>User Stories</i> , 3 C, INVEST, Gherkin ; démarche UML, Cockburn, scénario nominal ; inversion des dépendances
TD2	correction croisée INVEST, débat sur la <i>story</i> « lien dur », Gherkin, scénario nominal
TP2	le port et l'adaptateur POSIX

4 Formalisation

4.1 Acteurs

Acteur	Type	Objectif
Administrateur (Karim)	primaire, humain	garantir que les données sont récupérables
Chercheur	primaire, humain	récupérer <i>son</i> fichier, seul, sans déranger Karim
Auditeur	primaire, humain	prouver que la politique est appliquée
Ordonnanceur (cron)	primaire, système	déclencher l'exécution nocturne
Système de fichiers	secondaire, système	fournir et stocker les données

Trois rôles humains aux objectifs opposés : c'est ce qui rend les *User Stories* intéressantes. Le chercheur veut de l'**autonomie**, l'administrateur de la **maîtrise**, l'auditeur de la **preuve**. Faites-les jouer par trois étudiants différents.

4.2 Epics et User Stories

E1 Créer et conserver des instantanés (*prototype*) — **E2** Retrouver et restaurer — **E3** Piloter les politiques — **E4** Superviser et prouver.

- **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.*
- **US-02** — *En tant qu'administrateur, je veux qu'un instantané interrompu ne détruise pas les précédents, afin qu'une panne de nuit ne me laisse jamais sans sauvegarde exploitable.*
- **US-03** — *En tant qu'administrateur, je veux exclure les fichiers temporaires et les caches de calcul, afin de ne pas saturer le dépôt avec des données régénérables.*
- **US-04** — *En tant que chercheur, je veux parcourir un instantané comme un dossier ordinaire, afin de retrouver mon fichier avec les outils que je connais déjà, sans apprendre une commande de plus.*
- **US-05** — *En tant que chercheur, je veux savoir à quelles dates un fichier donné a été sauvegardé, afin de choisir la version à récupérer.*
- **US-06** — *En tant qu'administrateur, je veux être averti dès qu'une sauvegarde nocturne échoue, afin de ne pas découvrir le problème le jour où j'ai besoin de restaurer.*
- **US-07** — *En tant qu'auditeur, je veux consulter l'historique des sauvegardes des 12 derniers mois, afin de vérifier la conformité à la politique du laboratoire.*

✗ À ne pas faire : La *story* qu'il ne fallait pas écrire

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

Trois défauts : le rôle n'est pas un utilisateur, le « quoi » est une solution technique, le bénéfice est technique. Le lien dur n'est **pas** un besoin : c'est une *réponse possible* à **US-01**. La preuve : si demain le système de fichiers offre `copy_file_range` avec déduplication native, **US-01** est toujours valide et l'implémentation change. **Écrire l'US au niveau du besoin préserve la liberté de conception.**

4.3 US-01 au filtre INVEST

Lettre	Évaluation
Independent	ne dépend d'aucune autre story ; peut être livrée seule
Negotiable	le <i>comment</i> (liens durs ? déduplication par blocs ?) reste ouvert
Valuable	valeur chiffrable : 30 jours d'historique au lieu de 2, sans achat de matériel
Estimable	l'équipe sait la chiffrer (≈ 3 j)
Small	tient dans une itération
Testable	critères mesurables ci-dessous

4.4 Critères d'acceptation de US-01 (format Gherkin)

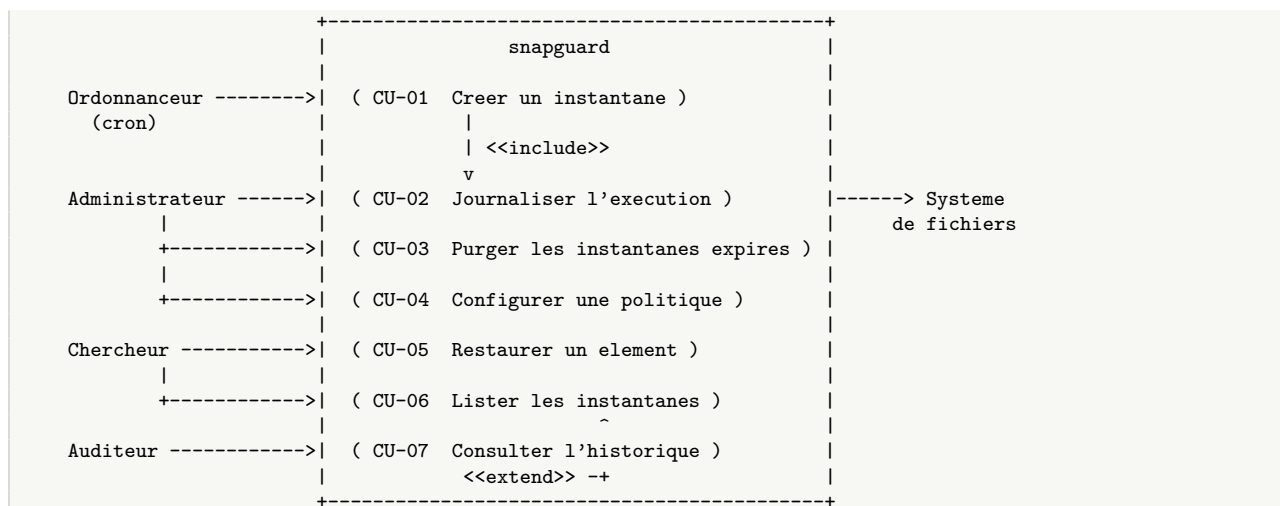
Fonctionnalite : sauvegarde incrementale d'un repertoire de travail

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 la commande se termine en succes
 Et 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

Scenario : un fichier restaure depuis lundi est identique a l'original
 Etant donne une sauvegarde realisee lundi
 Quand le fichier source est ecrase mardi
 Alors le fichier present dans l'instantane de lundi est inchange

Ces deux scénarios sont **directement exécutables** : ils constituent le fichier `tests/test_acceptation.sh` livré avec le prototype. La spécification *est* le test.

4.5 Diagramme de cas d'utilisation



Rappel à faire en cours : **ce diagramme est une table des matières**. Il ne dit ni dans quel ordre, ni ce qui se passe si le disque est plein. Toute sa valeur est dans les descriptions textuelles qui suivent.

4.6 CU-01 — Créer un instantané

Acteur primaire : Ordonnanceur (cron) — déclenchement également possible par l'Administrateur.

Acteur secondaire : Système de fichiers.

Niveau : *niveau de la mer* (Cockburn) — un objectif accompli d'une traite.

Préconditions : la source existe et est un répertoire lisible ; le dépôt est accessible en écriture ; aucun instantané n'est en cours pour cette source.

Scénario nominal

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 rencontré, le Système recrée la structure correspondante.
6. Pour chaque fichier, le Système compare taille et date de modification à celles de son homologue dans l'instantané de référence (**RG-02**).
7. Si le fichier est inchangé, le Système crée un lien dur vers l'exemplaire déjà présent dans le dépôt (**RG-03**).
8. Sinon, le Système copie le fichier dans l'instantané.
9. Le Système marque l'instantané comme complet et produit le rapport.

Les scénarios d'exception ne sont pas traités ici

Le scénario nominal s'arrête volontairement là. Les huit flux alternatifs de CU-01 font l'objet de la séquence 3 : c'est le **jeu des pannes** du TD3 qui doit les faire émerger, pas le CM2.

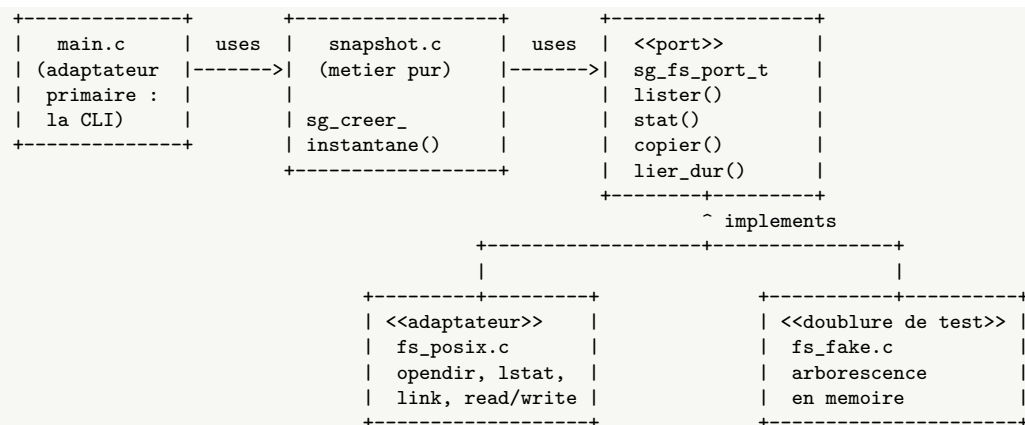
5 Du modèle de domaine au modèle de conception

5.1 Le choix d'architecture : ports et adaptateurs

La contrainte décisive vient de **ENF-MAIN-01** : *la logique métier doit être testable sans accès au système de fichiers réel*. Cette seule exigence détermine l'architecture.

Si `sg_creer_instantane()` appelle directement `opendir()` et `link()`, alors tester « un fichier inchangé doit être lié plutôt que copié » impose de fabriquer une vraie arborescence, de manipuler des `mtime` avec `utimensat`, et de tenter le test sur un point de montage séparé pour vérifier le repli 7a. Test lent, fragile, difficile à écrire — c'est le « cornet de glace ».

On applique donc le **principe d'inversion des dépendances** : le métier ne dépend pas du système de fichiers, mais d'un **contrat abstrait**, le `port sg_fs_port_t`.



5.2 Comment écrit-on une interface en C ?

Le C n'a ni interface, ni class, ni virtual. Le point est important à traiter en cours : **les principes de conception ne sont pas des propriétés du langage**. L'inversion des dépendances s'obtient en C par une **structure de pointeurs de fonctions** — c'est exactement ce que fait le noyau Linux avec ses `struct file_operations`, et la libc avec ses `FILE*`.

Concept UML	Traduction en C
Interface <code>IFileSystem</code>	<code>struct sg_fs_port</code> contenant des pointeurs de fonction
Méthode abstraite <code>copier()</code>	champ <code>int (*copier)(void *ctx, const char *, const char *)</code>
Classe qui implémente	fichier <code>.c</code> définissant des fonctions <code>static</code> + un constructeur
État de l'objet (<code>this</code>)	champ <code>void *ctx</code> passé en premier argument
Injection de dépendance	passage du port en paramètre de la fonction métier
Polymorphisme	déréférencement du pointeur de fonction à l'exécution

Les fonctions de l'adaptateur sont déclarées `static` : invisibles hors de leur unité de compilation. **Le seul symbole exporté est `sg_fs_posix()`**, qui renvoie le port. C'est l'équivalent C d'une classe dont tous les membres sont privés sauf le constructeur — encapsulation réelle, sans mot-clé dédié.

5.3 Correspondance domaine → conception

Classe du domaine	Devient	Remarque
Instantané	<code>sg_rapport_t</code> + arborescence	l'instantané est le répertoire ; pas de base de données en v1
Rapport	<code>struct sg_rapport_t</code>	passé par pointeur, jamais alloué dynamiquement
statut <i>EnCours/Complet/...</i> Source, Dépôt RG-02	<code>enum sg_code_t</code> + code de sortie paramètres <code>const char *</code> fonction <code>est_inchange()</code>	l'automate est porté par le code de retour la v1 traite une source à la fois une règle de gestion = une fonction nommée et testable
Politique	<i>absente de la v1</i>	déléguée à cron

✓ À faire : La consigne de TP à retenir de ce tableau

Toute règle de gestion identifiée en analyse doit être retrouvable dans le code sous la forme d'une fonction portant son nom. Si `est_inchange()` n'existe pas et que la comparaison est noyée dans une condition au milieu d'une boucle, la traçabilité exigeance → code est perdue, et personne ne saura plus où modifier le jour où Karim demandera le SHA-256.

6 Le prototype

Une **tranche verticale** : **US-01** seule, mais complète, de la ligne de commande jusqu'au test d'acceptation. Le code compile sous Linux et macOS avec `cc -std=c11 -Wall -Wextra -Wpedantic` sans avertissement, et l'ensemble des tests passe.

```

snapguard/
+- Makefile
+- src/
| +- fs_port.h          <- le contrat (aucune dependance systeme)
| +- snapshot.h/.c     <- le metier (ne connait que fs_port.h)
| +- fs_posix.c        <- l'adaptateur reel (seul fichier avec des appels systeme)
| +- main.c           <- l'adaptateur CLI
+- tests/
  +- fs_fake.h/.c      <- la doublure : un systeme de fichiers en memoire
  +- test_snapshot.c   <- tests unitaires
  +- test_integration.sh
  +- test_acceptation.sh

```

6.1 Le port : `src/fs_port.h`

Remarquez ce que ce fichier **ne contient pas** : ni `<dirent.h>`, ni `<sys/stat.h>`, ni `<unistd.h>`. C'est la garantie que le métier reste indépendant du système.

```

1  /* fs_port.h -- Le "Port" : contrat abstrait d'accès au système de fichiers.
2  *
3  * Aucune fonction POSIX n'apparaît ici : la logique métier ne dépendra que de
4  * cette abstraction (principe d'inversion des dépendances, le "D" de SOLID).
5  */
6  #ifndef SG_FS_PORT_H
7  #define SG_FS_PORT_H
8
9  #include <stddef.h>
10 #include <sys/types.h>
11 #include <time.h>
12
13 #define SG_MAX_CHEMIN 1024
14 #define SG_MAX_NOM    256

```

```

15
16 /* Métadonnées d'une entrée de répertoire (le strict nécessaire au métier). */
17 typedef struct {
18     char    nom[SG_MAX_NOM];
19     off_t   taille;
20     time_t  mtime;
21     int     est_repertoire;
22 } sg_entree_t;
23
24 typedef struct sg_fs_port {
25     void *ctx; /* état propre à l'implémentation (NULL pour POSIX) */
26
27     /* Liste les entrées de 'chemin'. Renvoie le nombre d'entrées écrites
28        dans 'tab' (au plus 'max'), ou -1 en cas d'erreur. */
29     int (*lister)(void *ctx, const char *chemin, sg_entree_t *tab, size_t max);
30
31     /* Renseigne 'info' pour 'chemin'. Renvoie 0 si trouvé, -1 sinon. */
32     int (*stat)(void *ctx, const char *chemin, sg_entree_t *info);
33
34     int (*creer_repertoire)(void *ctx, const char *chemin);
35     int (*copier)(void *ctx, const char *src, const char *dst);
36     int (*lier_dur)(void *ctx, const char *cible, const char *lien);
37 } sg_fs_port_t;
38
39 /* Adaptateur réel (POSIX : Linux, macOS, *BSD). */
40 const sg_fs_port_t *sg_fs_posix(void);
41
42 #endif /* SG_FS_PORT_H */

```

6.2 Le métier : src/snapshot.h

```

1 /* snapshot.h -- Service métier : création d'un instantané incrémental. */
2 #ifndef SG_SNAPSHOT_H
3 #define SG_SNAPSHOT_H
4
5 #include "fs_port.h"
6
7 typedef struct {
8     int fichiers_vus;
9     int fichiers_copies; /* nouveaux ou modifiés */
10    int fichiers_lies; /* inchangés -> lien dur (0 octet) */
11    int repertoires_creés;
12    int erreurs;
13 } sg_rapport_t;
14
15 /* Codes de retour du cas d'utilisation CU-01 "Créer un instantané". */
16 typedef enum {
17     SG_OK = 0,
18     SG_ERR_ARGUMENTS = -1, /* paramètre manquant / trop long */
19     SG_ERR_SOURCE = -2, /* source absente ou non répertoire */
20     SG_ERR_DESTINATION = -3, /* dépôt non créable */
21     SG_ERR_PARTIEL = -4 /* instantané terminé avec des erreurs */
22 } sg_code_t;
23
24 /* 'precedent' peut valoir NULL : on force alors une sauvegarde complète. */
25 sg_code_t sg_creer_instantane(const sg_fs_port_t *fs,
26                               const char *source,
27                               const char *precedent,
28                               const char *destination,
29                               sg_rapport_t *rapport);
30
31 #endif /* SG_SNAPSHOT_H */

```

6.3 Le métier : src/snapshot.c

Ce fichier est la traduction directe de CU-01. Les commentaires renvoient aux règles de gestion et aux flux alternatifs : c'est la trace de l'analyse dans le code.

```

1 /* snapshot.c -- Logique métier pure : aucun appel système direct.
2  * Toute interaction avec le disque passe par le port 'sg_fs_port_t'.
3  */
4 #include "snapshot.h"
5 #include <stdio.h>

```

```

6 #include <string.h>
7
8 #define SG_MAX_ENTREES 512
9
10 /* Concatène 'base' + "/" + 'nom' dans 'sortie'. Renvoie 0 si tout tient. */
11 static int joindre(char *sortie, size_t taille,
12                  const char *base, const char *nom)
13 {
14     int n = snprintf(sortie, taille, "%s/%s", base, nom);
15     return (n < 0 || (size_t)n >= taille) ? -1 : 0;
16 }
17
18 /* Règle métier centrale : un fichier est considéré inchangé si sa taille
19  * ET sa date de modification sont identiques à celles de l'instantané
20  * précédent. C'est l'heuristique retenue en RG-02 (cf. cahier d'analyse). */
21 static int est_inchange(const sg_entree_t *courant, const sg_entree_t *ancien)
22 {
23     return courant->taille == ancien->taille && courant->mtime == ancien->mtime;
24 }
25
26 static sg_code_t parcourir(const sg_fs_port_t *fs,
27                          const char *src, const char *prec, const char *dst,
28                          sg_rapport_t *r)
29 {
30     sg_entree_t entrees[SG_MAX_ENTREES];
31     int n = fs->listier(fs->ctx, src, entrees, SG_MAX_ENTREES);
32     if (n < 0) { r->erreurs++; return SG_ERR_SOURCE; }
33
34     for (int i = 0; i < n; i++) {
35         char c_src[SG_MAX_CHEMIN], c_dst[SG_MAX_CHEMIN], c_prec[SG_MAX_CHEMIN];
36
37         if (joindre(c_src, sizeof c_src, src, entrees[i].nom) != 0 ||
38             joindre(c_dst, sizeof c_dst, dst, entrees[i].nom) != 0) {
39             r->erreurs++; /* chemin trop long : on saute */
40             continue;
41         }
42         int a_precedent = (prec != NULL) &&
43             (joindre(c_prec, sizeof c_prec, prec, entrees[i].nom) == 0);
44
45         if (entrees[i].est_repertoire) {
46             if (fs->creer_repertoire(fs->ctx, c_dst) != 0) { r->erreurs++; continue; }
47             r->repertoires_creées++;
48             parcourir(fs, c_src, a_precedent ? c_prec : NULL, c_dst, r);
49             continue;
50         }
51
52         r->fichiers_vus++;
53         sg_entree_t ancien;
54         if (a_precedent &&
55             fs->stat(fs->ctx, c_prec, &ancien) == 0 &&
56             !ancien.est_repertoire &&
57             est_inchange(&entrees[i], &ancien)) {
58             if (fs->lier_dur(fs->ctx, c_prec, c_dst) == 0) { r->fichiers_lies++; continue; }
59             /* Le lien dur peut échouer (systèmes de fichiers différents) :
60              * on se rabat sur la copie. C'est le flux alternatif 6a du CU-01. */
61         }
62         if (fs->copier(fs->ctx, c_src, c_dst) == 0) r->fichiers_copies++;
63         else r->erreurs++;
64     }
65     return SG_OK;
66 }
67
68 sg_code_t sg_creer_instantane(const sg_fs_port_t *fs,
69                             const char *source,
70                             const char *precedent,
71                             const char *destination,
72                             sg_rapport_t *rapport)
73 {
74     if (!fs || !source || !destination || !rapport) return SG_ERR_ARGUMENTS;
75     if (source[0] == '\0' || destination[0] == '\0') return SG_ERR_ARGUMENTS;
76
77     memset(rapport, 0, sizeof *rapport);
78
79     sg_entree_t info;
80     if (fs->stat(fs->ctx, source, &info) != 0 || !info.est_repertoire)
81         return SG_ERR_SOURCE;

```

```

82
83     if (fs->creer_repertoire(fs->ctx, destination) != 0)
84         return SG_ERR_DESTINATION;
85
86     sg_code_t code = parcourir(fs, source, precedent, destination, rapport);
87     if (code != SG_OK)         return code;
88     if (rapport->erreurs > 0) return SG_ERR_PARTIEL;
89     return SG_OK;
90 }

```

6.4 L'adaptateur POSIX : src/fs_posix.c

Le seul fichier du projet qui contienne des appels système. C'est aussi le seul à modifier pour porter l'outil ailleurs, ou passer à `openat()` pour éviter les attaques par remplacement de chemin.

```

1  /* fs_posix.c -- Adaptateur : la seule brique qui connaît les appels système.
2   * Compile tel quel sous Linux et macOS (API POSIX.1-2008).
3   */
4  #define _POSIX_C_SOURCE 200809L
5  #include "fs_port.h"
6
7  #include <dirent.h>
8  #include <errno.h>
9  #include <fcntl.h>
10 #include <stdio.h>
11 #include <string.h>
12 #include <sys/stat.h>
13 #include <unistd.h>
14
15 static int posix_lister(void *ctx, const char *chemin, sg_entree_t *tab, size_t max)
16 {
17     (void)ctx;
18     DIR *d = opendir(chemin);
19     if (!d) return -1;
20
21     size_t n = 0;
22     struct dirent *e;
23     while (n < max && (e = readdir(d)) != NULL) {
24         if (strcmp(e->d_name, ".") == 0 || strcmp(e->d_name, "..") == 0) continue;
25
26         char complet[SG_MAX_CHEMIN];
27         if (snprintf(complet, sizeof complet, "%s/%s", chemin, e->d_name)
28             >= (int)sizeof complet) continue;
29
30         struct stat st;
31         if (lstat(complet, &st) != 0) continue;
32         if (!S_ISREG(st.st_mode) && !S_ISDIR(st.st_mode)) continue; /* hors périmètre v1 */
33
34         snprintf(tab[n].nom, SG_MAX_NOM, "%s", e->d_name);
35         tab[n].taille = S_ISDIR(st.st_mode) ? 0 : st.st_size;
36         tab[n].mtime = st.st_mtime;
37         tab[n].est_repertoire = S_ISDIR(st.st_mode) ? 1 : 0;
38         n++;
39     }
40     closedir(d);
41     return (int)n;
42 }
43
44 static int posix_stat(void *ctx, const char *chemin, sg_entree_t *info)
45 {
46     (void)ctx;
47     struct stat st;
48     if (lstat(chemin, &st) != 0) return -1;
49
50     const char *base = strrchr(chemin, '/');
51     snprintf(info->nom, SG_MAX_NOM, "%s", base ? base + 1 : chemin);
52     info->est_repertoire = S_ISDIR(st.st_mode) ? 1 : 0;
53     info->taille = info->est_repertoire ? 0 : st.st_size;
54     info->mtime = st.st_mtime;
55     return 0;
56 }
57
58 static int posix_creer_repertoire(void *ctx, const char *chemin)
59 {

```

```

60     (void)ctx;
61     if (mkdir(chemin, 0700) == 0) return 0;
62     return (errno == EEXIST) ? 0 : -1;  /* idempotence volontaire */
63 }
64
65 static int posix_copier(void *ctx, const char *src, const char *dst)
66 {
67     (void)ctx;
68     int fs_ = open(src, O_RDONLY);
69     if (fs_ < 0) return -1;
70
71     struct stat st;
72     if (fstat(fs_, &st) != 0) { close(fs_); return -1; }
73
74     /* ENF-SEC-01 : le dépôt reproduit les droits de la source, jamais plus. */
75     int fd = open(dst, O_WRONLY | O_CREAT | O_TRUNC, st.st_mode & 0777);
76     if (fd < 0) { close(fs_); return -1; }
77
78     char tampon[65536];
79     ssize_t lu;
80     int code = 0;
81     while ((lu = read(fs_, tampon, sizeof tampon)) > 0) {
82         ssize_t ecrit = 0;
83         while (ecrit < lu) {
84             ssize_t k = write(fd, tampon + ecrit, (size_t)(lu - ecrit));
85             if (k < 0) { code = -1; goto fin; }
86             ecrit += k;
87         }
88     }
89     if (lu < 0) code = -1;
90 fin:
91     close(fs_);
92     if (close(fd) != 0) code = -1;  /* ENF-FIA-01 : détecter un disque plein */
93     return code;
94 }
95
96 static int posix_liier_dur(void *ctx, const char *cible, const char *lien)
97 {
98     (void)ctx;
99     return link(cible, lien) == 0 ? 0 : -1;
100 }
101
102 const sg_fs_port_t *sg_fs_posix(void)
103 {
104     static const sg_fs_port_t port = {
105         .ctx          = NULL,
106         .lister        = posix_lister,
107         .stat          = posix_stat,
108         .creer_repertoire = posix_creer_repertoire,
109         .copier        = posix_copier,
110         .liier_dur     = posix_liier_dur
111     };
112     return &port;
113 }

```

6.5 L'adaptateur CLI : src/main.c

```

1  /* main.c -- Adaptateur primaire : la ligne de commande.
2  *
3  *   snapguard <source> <depot> [instantane_precedent]
4  *
5  * Conventions Unix respectées (ENF-UTI-01) :
6  *   - code de sortie 0 en cas de succès, non nul sinon ;
7  *   - rapport lisible sur stdout, erreurs sur stderr ;
8  *   - aucune interaction : utilisable depuis cron ou un script.
9  */
10 #include "snapshot.h"
11 #include <stdio.h>
12 #include <string.h>
13 #include <time.h>
14
15 static void usage(const char *prog)
16 {
17     fprintf(stderr,

```

```

18         "Usage : %s <source> <depot> [instantane_precedent]\n"
19         " source          repertoire a sauvegarder\n"
20         " depot           repertoire ou creer l'instantane\n"
21         " instantane_precedent instantane de reference (incremental)\n",
22         prog);
23     }
24
25     int main(int argc, char **argv)
26     {
27         if (argc < 3 || argc > 4) { usage(argv[0]); return 2; }
28
29         const char *source      = argv[1];
30         const char *depot       = argv[2];
31         const char *precedent   = (argc == 4) ? argv[3] : NULL;
32
33         sg_rapport_t rapport;
34         sg_code_t code = sg_creer_instantane(sg_fs_posix(), source, precedent,
35                                             depot, &rapport);
36
37         switch (code) {
38         case SG_ERR_ARGUMENTS:
39             fprintf(stderr, "snapguard: arguments invalides\n");          return 2;
40         case SG_ERR_SOURCE:
41             fprintf(stderr, "snapguard: source illisible : %s\n", source); return 3;
42         case SG_ERR_DESTINATION:
43             fprintf(stderr, "snapguard: depot non creable : %s\n", depot); return 4;
44         default: break;
45         }
46
47         printf("Instantane : %s\n", depot);
48         printf(" fichiers examines : %d\n", rapport.fichiers_vus);
49         printf(" copies : %d\n", rapport.fichiers_copies);
50         printf(" lies (inchanges) : %d\n", rapport.fichiers_lies);
51         printf(" repertoires crees : %d\n", rapport.repertoires_crees);
52         printf(" erreurs : %d\n", rapport.erreurs);
53
54         return (code == SG_ERR_PARTIEL) ? 1 : 0;
55     }

```

6.6 Les points « système » à faire travailler en TP

Élément du code	Notion Unix	Question à poser
link()	lien dur, compteur de références	si je supprime l'instantané de lundi, mardi perd-il ses données ? (non)
lstat() vs stat()	suivi des liens symboliques	que se passerait-il avec stat sur un lien vers / ?
mkdir(...,0700) + EEXIST	idempotence, umask	pourquoi 0700 et pas 0755 ?
open(dst,...,st_mode&0777)	préservation des droits	et si la source est en 0600 ?
boucle write() partielle	les écritures courtes sont légales	pourquoi pas un seul write() ?
close() testé en retour	erreurs différées (NFS, disque plein)	peut-on perdre des données alors que tous les write() ont réussi ? (oui)
codes de sortie 0 à 4	convention Unix, \$?	comment cron saura-t-il qu'il faut alerter ?
stdout vs stderr	données / diagnostics	que donne snapguard > rapport.txt en cas d'erreur ?

7 Notes pour la conduite des trois séances

7.1 CM2

Quatre parties : l'échec du cahier des charges monolithique (10 min), les *User Stories* avec les 3 C, INVEST et Gherkin (25 min), les cas d'utilisation jusqu'au scénario nominal (20 min), puis le passage à l'architecture (20 min).

✓ À faire : L'enchaînement qui fait la cohérence du CM2

Le scénario nominal dicte les opérations dont le métier a besoin ; cette liste *est* l'interface ; et une ENF de maintenabilité impose que le métier en dépende plutôt que de POSIX. Déroulez-le dans cet ordre, et l'inversion des dépendances n'apparaît pas comme un principe plaqué mais comme la conséquence de ce qui précède.

7.2 TD2

Correction croisée INVEST (15 min), débat sur la story « lien dur » (20 min), critères Gherkin de **US-01** (20 min), scénario nominal collectif et extraction des opérations système (20 min).

🔗 Cas d'étude : Le débat sur la story « lien dur »

Beaucoup l'auront écrite : ils sortent du TP1 émerveillés par le mécanisme. Sollicitez l'*avocat du diable* de chaque groupe et posez la question décisive : *si demain le système de fichiers offrait une déduplication native, cette story serait-elle encore valide ?*

C'est le moment le plus efficace de tout le semestre pour faire comprendre qu'une exigence formulée au niveau du besoin **préserve la liberté de conception**.

7.3 TP2

Le port a été esquissé au TD2 ; le TP le fige et l'implémente. Confrontation des contrats (10 min), rédaction de `fs_port.h` (20 min), adaptateur POSIX (35 min), vérification (10 min).

✓ À faire : Laissez-les garder leur port

Si leur contrat coche les six cases de la grille, c'est le leur pour les quatre TP, même s'il diffère du modèle. Un port de référence reste disponible en secours : le prendre n'est pas une faute, mais doit être noté dans le dossier avec sa raison.

✗ À ne pas faire : Le piège de la séance

Les groupes qui listent huit ou dix opérations décrivent la libc, pas leur scénario nominal. Renvoyez-les au tableau du TD2 : *quelle étape réclame cette opération ?* Si aucune, elle saute. Une interface qui expose tout n'abstrait rien.

Séquence 3 — Ce qui peut mal tourner

Ce que couvre cette séquence

CM3	scénarios d'exception, décisions métier, pré/postconditions, pyramide de Cohn, doublures
TD3	jeu des pannes, arbitrages sur les cartes B et D, conception de la doublure
TP3	RG-02 , liens durs, replis, laboratoire de mesures

8 Les scénarios d'exception de CU-01

Scénarios alternatifs et d'exception

- **2a. Aucun instantané complet antérieur** — bascule en sauvegarde complète : chaque fichier est copié. Reprise à l'étape 3.
- **6a. Fichier absent de l'instantané de référence** (nouveau) — traité comme modifié : copie. Reprise à l'étape 8.
- **7a. La création du lien dur échoue** (dépôt sur un autre système de fichiers, limite du nombre de liens) — repli sur la copie, poursuite. L'instantané reste valide, seul l'espace consommé augmente. *Décision : la disponibilité de la sauvegarde prime sur l'économie d'espace.*
- **8a. Fichier illisible** (permission refusée) — journalisation du chemin, incrément du compteur d'erreurs, poursuite. L'instantané sera marqué *partiel*.
- **8b. Dépôt plein** — interruption du parcours, instantané marqué *échoué*, instantanés antérieurs intacts (**RG-01**), code d'erreur.
- **4a. La source disparaît en cours de parcours** (démontage) — même traitement que 8b.
- **9a. Compteur d'erreurs non nul** — instantané *partiel* : conservé, mais il ne pourra pas servir de référence au suivant.

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

Postcondition d'échec : aucun instantané complet ajouté ; antérieurs inchangés ; incident journalisé et signalé par le code de retour.

🔗 Cas d'étude : Là où le cas d'utilisation gagne contre la User Story

US-01 tient en une phrase. CU-01 fait apparaître **sept scénarios d'exception** — dont 7a et 9a, qui sont de véritables décisions métier que personne n'aurait prises spontanément au clavier. Faites l'expérience en TD : demandez d'abord d'implémenter à partir de la seule **US-01**, puis distribuez CU-01. Le nombre de cas oubliés est le meilleur argument pédagogique du cours.

9 La stratégie de test

Niveau	Fichier	Nb	Durée	Ce qu'il prouve
Unitaire	test_snapshot.c	18	< 10 ms	les règles de gestion sont correctes
Intégration	test_integration.sh	9	≈ 0,2 s	l' adaptateur POSIX tient la promesse du port
Acceptation	test_acceptation.sh	7	≈ 1 s	la valeur promise par US-01 est au rendez-vous

9.1 Tests unitaires — sans jamais toucher au disque

La doublure `fs_fake.c` est un **Fake** au sens de Meszaros : une implémentation simplifiée mais fonctionnelle (une arborescence en mémoire). Elle joue en plus le rôle de **Mock** en journalisant chaque opération, ce qui permet de vérifier des *interactions* et pas seulement des états.

Assertion	Nature	Ce qu'elle vérifie
<code>r.fichiers_lies == 2</code>	<i>état</i> (stub)	le résultat du calcul
<code>fake_journal_contient("LINK ...")</code>	<i>interaction</i> (mock)	que le lien pointe vers le bon instantané
<code>fake_compter("MKDIR") == 0</code>	<i>absence</i> d'interaction	qu'une source invalide ne produit aucun effet de bord

Le champ `echec_lien_dur` du fake permet de tester le **flux alternatif 7a** — repli sur la copie quand `link()` échoue — sans avoir besoin de deux systèmes de fichiers. Reproduire cette situation avec des tests d'intégration seuls demanderait des privilèges root et un montage dédié.

```

1  /* test_snapshot.c -- Tests unitaires : le metier est teste SANS toucher au disque. */
2  #include "../src/snapshot.h"
3  #include "fs_fake.h"
4  #include <stdio.h>
5  #include <string.h>
6
7  static int total = 0, echecs = 0;
8
9  #define VERIFIER(cond, libelle) \
10     do { \
11         total++; \
12         if (cond) printf(" [OK]      %s\n", libelle); \
13         else { echecs++; printf(" [ECHEC]  %s (ligne %d)\n", libelle, __LINE__); } \
14     } while (0)
15
16  /* Arbre commun : /src (rep) contenant a.txt, b.txt et /src/docs/c.txt */
17  static void arbre_source(fake_fs_t *f)
18  {
19     fakeajouter(f, "/src", 0, 0, 1);
20     fakeajouter(f, "/src/a.txt", 100, 1000, 0);
21     fakeajouter(f, "/src/b.txt", 200, 2000, 0);
22     fakeajouter(f, "/src/docs", 0, 0, 1);
23     fakeajouter(f, "/src/docs/c.txt", 50, 3000, 0);
24 }
25
26  /* CU-01, scenario nominal, cas "aucun instantane precedent" */
27  static void test_premiere_sauvegarde_copie_tout(void)
28  {
29     printf("Test 1 : premiere sauvegarde -> tout est copie\n");
30     fake_fs_t f; fake_init(&f); arbre_source(&f);
31     sg_fs_port_t port = fake_port(&f);
32     sg_rapport_t r;
33
34     sg_code_t code = sg_creer_instantane(&port, "/src", NULL, "/depot/2026-08-05", &r);
35
36     VERIFIER(code == SG_OK, "code de retour SG_OK");
37     VERIFIER(r.fichiers_vus == 3, "3 fichiers examines");
38     VERIFIER(r.fichiers_copies == 3, "3 fichiers copies");
39     VERIFIER(r.fichiers_lies == 0, "aucun lien dur");
40     VERIFIER(r.repertoires_creés == 1, "sous-repertoire docs recree");
41     VERIFIER(fake_journal_contient(&f, "COPY /src/docs/c.txt -> /depot/2026-08-05/docs/c.txt"),
42             "recursivite : c.txt copie au bon endroit");
43 }
44
45  /* RG-02 : taille + mtime identiques => lien dur, aucune copie */
46  static void test_fichier_inchange_est_lie(void)
47  {
48     printf("Test 2 : fichier inchange -> lien dur (0 octet consomme)\n");
49     fake_fs_t f; fake_init(&f); arbre_source(&f);
50     fakeajouter(&f, "/depot/j1", 0, 0, 1);
51     fakeajouter(&f, "/depot/j1/a.txt", 100, 1000, 0); /* identique */
52     fakeajouter(&f, "/depot/j1/b.txt", 200, 2000, 0); /* identique */
53     sg_fs_port_t port = fake_port(&f);
54     sg_rapport_t r;
55
56     sg_creer_instantane(&port, "/src", "/depot/j1", "/depot/j2", &r);
57
58     VERIFIER(r.fichiers_lies == 2, "a.txt et b.txt sont lies");
59     VERIFIER(r.fichiers_copies == 1, "seul c.txt (absent de j1) est copie");
60     VERIFIER(fake_journal_contient(&f, "LINK /depot/j1/a.txt -> /depot/j2/a.txt"),
61             "le lien pointe bien vers l'instantane precedent");
62 }
63

```

```

64  /* RG-02 : mtime different => le fichier est recopie */
65  static void test_fichier_modifie_est_copie(void)
66  {
67      printf("Test 3 : fichier modifie -> copie\n");
68      fake_fs_t f; fake_init(&f); arbre_source(&f);
69      fake_ajouter(&f, "/depot/j1", 0, 0, 1);
70      fake_ajouter(&f, "/depot/j1/a.txt", 100, 1000, 0); /* identique */
71      fake_ajouter(&f, "/depot/j1/b.txt", 200, 1999, 0); /* mtime autre */
72      sg_fs_port_t port = fake_port(&f);
73      sg_rapport_t r;
74
75      sg_creer_instantane(&port, "/src", "/depot/j1", "/depot/j2", &r);
76
77      VERIFIER(r.fichiers_lies == 1, "seul a.txt est lie");
78      VERIFIER(r.fichiers_copies == 2, "b.txt (modifie) et c.txt sont copies");
79      VERIFIER(fake_journal_contient(&f, "COPY /src/b.txt -> /depot/j2/b.txt"),
80              "b.txt a bien ete recopie depuis la source");
81  }
82
83  /* CU-01, flux alternatif 6a : link() echoue -> repli sur la copie */
84  static void test_repli_copie_si_lien_impossible(void)
85  {
86      printf("Test 4 : echec du lien dur -> repli sur la copie (flux 6a)\n");
87      fake_fs_t f; fake_init(&f); arbre_source(&f);
88      f.echec_lien_dur = 1;
89      fake_ajouter(&f, "/depot/j1", 0, 0, 1);
90      fake_ajouter(&f, "/depot/j1/a.txt", 100, 1000, 0);
91      sg_fs_port_t port = fake_port(&f);
92      sg_rapport_t r;
93
94      sg_code_t code = sg_creer_instantane(&port, "/src", "/depot/j1", "/depot/j2", &r);
95
96      VERIFIER(code == SG_OK, "l'instantane reste un succes");
97      VERIFIER(r.fichiers_lies == 0, "aucun lien comptabilise");
98      VERIFIER(r.fichiers_copies == 3, "les 3 fichiers ont ete copies");
99  }
100
101  /* CU-01, precondition : la source doit exister et etre un repertoire */
102  static void test_source_invalide(void)
103  {
104      printf("Test 5 : source absente -> SG_ERR_SOURCE, aucune ecriture\n");
105      fake_fs_t f; fake_init(&f);
106      sg_fs_port_t port = fake_port(&f);
107      sg_rapport_t r;
108
109      sg_code_t code = sg_creer_instantane(&port, "/introuvable", NULL, "/depot/j1", &r);
110
111      VERIFIER(code == SG_ERR_SOURCE, "code SG_ERR_SOURCE");
112      VERIFIER(fake_compter(&f, "MKDIR") == 0, "aucun repertoire cree (pas d'effet de bord)");
113      VERIFIER(fake_compter(&f, "COPY") == 0, "aucune copie tentee");
114  }
115
116  int main(void)
117  {
118      printf("=== Tests unitaires snapguard ===\n\n");
119      test_premiere_sauvegarde_copie_tout();
120      test_fichier_inchange_est_lie();
121      test_fichier_modifie_est_copie();
122      test_repli_copie_si_lien_impossible();
123      test_source_invalide();
124      printf("\n%d verifications, %d echec(s)\n", total, echecs);
125      return echecs == 0 ? 0 : 1;
126  }

```

9.2 La doublure : tests/fs_fake.c

```

1  /* fs_fake.h -- Doublure de test : un système de fichiers en mémoire.
2  *
3  * C'est un "Fake" au sens de Meszaros : une implémentation simplifiée mais
4  * fonctionnelle du port. Il journalise en plus chaque operation, ce qui
5  * permet de l'utiliser comme "Mock" (verification des interactions).
6  */
7  #ifndef SG_FS_FAKE_H
8  #define SG_FS_FAKE_H

```

```

9
10 #include "../src/fs_port.h"
11
12 #define FAKE_MAX_FICHIERS 64
13 #define FAKE_MAX_JOURNAL 128
14
15 typedef struct {
16     char    chemin[SG_MAX_CHEMIN];
17     off_t   taille;
18     time_t  mtime;
19     int     est_repertoire;
20 } fake_noeud_t;
21
22 typedef struct {
23     fake_noeud_t noeuds[FAKE_MAX_FICHIERS];
24     int          nb_noeuds;
25     char         journal[FAKE_MAX_JOURNAL][SG_MAX_CHEMIN];
26     int          nb_journal;
27     int          echec_lien_dur; /* 1 = link() echoue toujours */
28 } fake_fs_t;
29
30 void fake_init(fake_fs_t *f);
31 void fake_ajouter(fake_fs_t *f, const char *chemin, off_t taille,
32                  time_t mtime, int est_repertoire);
33 int  fake_journal_contient(const fake_fs_t *f, const char *motif);
34 int  fake_compter(const fake_fs_t *f, const char *prefixe_operation);
35 sg_fs_port_t fake_port(fake_fs_t *f);
36
37 #endif /* SG_FS_FAKE_H */

```

```

1 #include "fs_fake.h"
2 #include <stdio.h>
3 #include <string.h>
4
5 static void journaliser(fake_fs_t *f, const char *format, const char *a, const char *b)
6 {
7     if (f->nb_journal >= FAKE_MAX_JOURNAL) return;
8     snprintf(f->journal[f->nb_journal++], SG_MAX_CHEMIN, format, a, b);
9 }
10
11 static fake_noeud_t *trouver(fake_fs_t *f, const char *chemin)
12 {
13     for (int i = 0; i < f->nb_noeuds; i++)
14         if (strcmp(f->noeuds[i].chemin, chemin) == 0) return &f->noeuds[i];
15     return NULL;
16 }
17
18 /* 'chemin' est-il un enfant DIRECT de 'parent' ? */
19 static const char *enfant_direct(const char *chemin, const char *parent)
20 {
21     size_t n = strlen(parent);
22     if (strncmp(chemin, parent, n) != 0 || chemin[n] != '/') return NULL;
23     const char *reste = chemin + n + 1;
24     return strchr(reste, '/') ? NULL : reste;
25 }
26
27 static int fake_lister(void *ctx, const char *chemin, sg_entree_t *tab, size_t max)
28 {
29     fake_fs_t *f = ctx;
30     fake_noeud_t *rep = trouver(f, chemin);
31     if (!rep || !rep->est_repertoire) return -1;
32
33     size_t n = 0;
34     for (int i = 0; i < f->nb_noeuds && n < max; i++) {
35         const char *nom = enfant_direct(f->noeuds[i].chemin, chemin);
36         if (!nom) continue;
37         snprintf(tab[n].nom, SG_MAX_NOM, "%s", nom);
38         tab[n].taille = f->noeuds[i].taille;
39         tab[n].mtime = f->noeuds[i].mtime;
40         tab[n].est_repertoire = f->noeuds[i].est_repertoire;
41         n++;
42     }
43     return (int)n;
44 }
45
46 static int fake_stat(void *ctx, const char *chemin, sg_entree_t *info)

```

```

47 {
48     fake_fs_t *f = ctx;
49     fake_noeud_t *n = trouver(f, chemin);
50     if (!n) return -1;
51     const char *base = strrchr(chemin, '/');
52     snprintf(info->nom, SG_MAX_NOM, "%s", base ? base + 1 : chemin);
53     info->taille = n->taille;
54     info->mtime = n->mtime;
55     info->est_repertoire = n->est_repertoire;
56     return 0;
57 }
58
59 static int fake_creer_repertoire(void *ctx, const char *chemin)
60 {
61     fake_fs_t *f = ctx;
62     journaliser(f, "MKDIR %s%s", chemin, "");
63     if (!trouver(f, chemin)) fake_ajouter(f, chemin, 0, 0, 1);
64     return 0;
65 }
66
67 static int fake_copier(void *ctx, const char *src, const char *dst)
68 {
69     fake_fs_t *f = ctx;
70     journaliser(f, "COPY %s -> %s", src, dst);
71     fake_noeud_t *s = trouver(f, src);
72     if (!s) return -1;
73     fake_ajouter(f, dst, s->taille, s->mtime, 0);
74     return 0;
75 }
76
77 static int fake_liier_dur(void *ctx, const char *cible, const char *lien)
78 {
79     fake_fs_t *f = ctx;
80     journaliser(f, "LINK %s -> %s", cible, lien);
81     if (f->echec_lien_dur) return -1;
82     fake_noeud_t *c = trouver(f, cible);
83     if (!c) return -1;
84     fake_ajouter(f, lien, c->taille, c->mtime, 0);
85     return 0;
86 }
87
88 void fake_init(fake_fs_t *f) { memset(f, 0, sizeof *f); }
89
90 void fake_ajouter(fake_fs_t *f, const char *chemin, off_t taille,
91                 time_t mtime, int est_repertoire)
92 {
93     if (f->nb_noeuds >= FAKE_MAX_FICHIERS) return;
94     fake_noeud_t *n = &f->noeuds[f->nb_noeuds++];
95     snprintf(n->chemin, SG_MAX_CHEMIN, "%s", chemin);
96     n->taille = taille;
97     n->mtime = mtime;
98     n->est_repertoire = est_repertoire;
99 }
100
101 int fake_journal_contient(const fake_fs_t *f, const char *motif)
102 {
103     for (int i = 0; i < f->nb_journal; i++)
104         if (strstr(f->journal[i], motif)) return 1;
105     return 0;
106 }
107
108 int fake_compter(const fake_fs_t *f, const char *prefixe)
109 {
110     int c = 0;
111     for (int i = 0; i < f->nb_journal; i++)
112         if (strncmp(f->journal[i], prefixe, strlen(prefixe)) == 0) c++;
113     return c;
114 }
115
116 sg_fs_port_t fake_port(fake_fs_t *f)
117 {
118     sg_fs_port_t p = {
119         .ctx = f,
120         .lister = fake_lister,
121         .stat = fake_stat,
122         .creer_repertoire = fake_creer_repertoire,

```

```

123     .copier          = fake_copier,
124     .lier_dur        = fake_liier_dur
125 };
126 return p;
127 }

```

10 Notes pour la conduite des trois séances

10.1 CM3

Les exceptions (25 min), le contrat pré/postconditions (10 min), la pyramide des tests (15 min), les doublures (25 min).

✓ À faire : Le lien à rendre explicite

Les étudiants ont *constaté au clavier*, au TP1, qu'un lien dur ne franchit pas la frontière d'un système de fichiers. La carte D n'est donc pas une hypothèse d'école : c'est le comportement qu'ils ont observé. Dites-le — c'est ce qui transforme une liste d'exceptions abstraites en problèmes qu'ils reconnaissent.

10.2 TD3

Jeu des pannes (30 min), mise en commun et arbitrages (25 min), conception de la doublure (20 min). Les huit cartes à découper sont dans le *Matériel de séance*.

🔗 Cas d'étude : Distribuez impérativement les cartes B et D

Ce sont les deux seules qui appellent une décision **métier** et non technique. La D oppose disponibilité de la sauvegarde et économie d'espace ; la B fait naître la notion de *statut*, dont le TD4 tirera l'automate à états. Sans elles, la mise en commun perd son intérêt.

Avec six groupes tirant trois cartes chacun, prévoyez deux jeux complets.

10.3 TP3

Point de blocage collectif (10 min), **RG-02** isolée (10 min), incrémental et replis (25 min), laboratoire de mesures (25 min), attribution des variantes (5 min).

✓ À faire : Le laboratoire de mesures est le sommet du semestre

C'est le seul moment où la promesse faite au client se vérifie *au chiffre*. `ls -i` montre le partage d'inodes, `du` montre que deux instantanés tiennent dans l'espace d'un seul. Ne le sacrifiez pas au débogage : si un groupe n'a pas fini l'incrémental, faites-lui faire les mesures sur le dépôt d'un autre groupe.

La question qui fait le piège du `du`

`du -sk depot/j2` lancé seul annonce 2000 Kio ; `du -sk depot` n'en annonce que 2100 pour les deux instantanés réunis — dans une même invocation, `du` ne compte qu'une fois chaque inode. *Quel chiffre est le bon ?* Les deux : la question mal posée est « quelle place prend mardi ? », la bonne est « quelle place prend le dépôt ? ».

Séquence 4 — Modéliser, puis éprouver la robustesse

Ce que couvre cette séquence

CM4	modèle de domaine, langage ubiquitaire, classe-association, processus métier, automate à états
TD4	tri des concepts, multiplicités, processus nocturne, automate et transitions interdites
TP4	expérience kill -9, correction atomique, revue croisée

11 Le modèle de domaine

11.1 Étape 1 : extraction des noms

En surlignant les substantifs des exigences, des US et de CU-01 : *Source, Dépôt, Instantané, Fichier, Répertoire, Politique, Rétention, Exclusion, Administrateur, Chercheur, Rapport, Erreur, Date, Taille, Lien dur, Restauration*.

11.2 Étape 2 : raffinage — le travail de tri

Candidat	Décision	Justification
Source, Dépôt, Instantané, Politique	classes	concepts autonomes, dont le client parle spontanément
Taille, Date, Rétention	attributs	ne vivent pas indépendamment de leur porteur
Lien dur	rejeté	solution technique, pas concept métier : RG-03 dit « ne pas recopier », pas « faire un lien »
Répertoire	fusionné	un répertoire et un fichier partagent chemin et date
Rapport	classe	existence propre : conservé, consulté, audité (US-07)
Administrateur, Chercheur, Auditeur Exclusion	héritage classe-association	mêmes attributs d'identité, droits différents porte des attributs propres à la relation Politique × Source

✓ À faire : La discussion à mener en TD sur « Lien dur »

C'est le meilleur exercice de discrimination analyse/conception de tout le cours. Le lien dur est la trouvaille technique du projet, celle dont les étudiants sont le plus fiers — et elle n'a **aucune place** dans le modèle du domaine. Karim ne dit jamais « lien dur ». Il dit « je ne veux pas payer trente fois le même fichier ».

11.3 Étape 3 : associations et multiplicités

Les questions posées à Karim, et ses réponses :

- Une politique peut-elle couvrir plusieurs sources ? — Oui, « les données du labo, c'est trois dossiers ». → Politique 1 - 1..* Inclusion
- Une même source peut-elle relever de deux politiques ? — Oui : /data est sauvegardé quotidiennement en local et hebdomadairement sur le site distant, avec des exclusions différentes. → association *n..m* porteuse d'attributs → **classe-association obligatoire**.
- Un instantané peut-il concerner plusieurs sources ? — Non. → Source 1 - 0..* Instantané
- Un instantané peut-il exister sans dépôt ? — Non. → Dépôt 1 - 0..* Instantané
- Un instantané a-t-il toujours une référence ? — Non, le premier n'en a pas. → auto-association 0..1 - 0..* (« dérive de »)

11.4 Le diagramme (source PlantUML)

```
@startuml
skinparam classAttributeIconSize 0
hide circle

class Utilisateur { identifiant \n nom }
class Administrateur
class Chercheur
class Auditeur
Utilisateur <|-- Administrateur
Utilisateur <|-- Chercheur
Utilisateur <|-- Auditeur

class Politique { nom \n frequence \n retentionJours }
class Source { chemin \n description }
class Depot { chemin \n espaceDisponible }
class Instantane { horodatage \n statut \n dateDebut \n dateFin }
class Rapport { fichiersVus \n fichiersCopies \n fichiersLies \n octetsEcrits \n erreurs }
class Inclusion { motifsExclusion \n recursif }

Administrateur "1" -- "0..*" Politique : definit >
Politique "1..*" -- "1..*" Source
(Politique, Source) . Inclusion
Politique "0..*" --> "1" Depot : cible >
Source "1" -- "0..*" Instantane : est capturee par >
Depot "1" -- "0..*" Instantane : heberge >
Instantane "1" -- "1" Rapport : produit >
Instantane "0..1" <-- "0..*" Instantane : derive de

@enduml
```

Compilez ce bloc avec `plantuml` et insérez le PDF obtenu par `\includegraphics`. La source est volontairement livrée en clair : c'est elle que les étudiants doivent savoir écrire, pas l'image.

11.5 Ce que le modèle ne contient pas — et pourquoi

Absent	Pourquoi
LienDur, Inode, DescripteurFichier	vocabulaire de la machine, pas du client
InstantaneDAO, PolitiqueParser, FileWalker	artefacts de solution → diagramme de conception
char*, size_t, time_t	types techniques : « horodatage » suffit à ce stade
creerInstantane(), purger()	opérations → conception ; ici on décrit le quoi

12 Les processus métier

12.1 Identification

- **Quels grands événements ?** L'échéance nocturne ; la découverte d'une perte ; la saturation du dépôt.
- **Quel cycle de vie ?** Celui de l'Instantané, et celui d'une Politique.
- **Qui fait quoi, quand ?** Karim la nuit (rien : c'est `cron`), Karim le matin (il vérifie), le chercheur en urgence (il restaure).

Trois processus : **P1** Sauvegarde nocturne, **P2** Restauration d'urgence, **P3** Purge et surveillance de l'espace.

12.2 P1 — Sauvegarde nocturne

1. **[Ordonnanceur]** déclenche l'exécution à l'heure prévue par la Politique.
2. **[Système]** vérifie que le dépôt est monté et dispose de l'espace estimé.
3. **[Système]** identifie le dernier instantané complet, qui servira de référence.
4. **[Système]** crée l'instantané au statut *EnCours*, enregistre l'heure de début.
5. **[Système]** parcourt la source et applique **RG-02** et **RG-03**.
6. **[Système]** clôt l'instantané : *Complet*, *Partiel* ou *Échoué*.
7. **[Système]** enregistre le Rapport et journalise le résultat via `syslog`.

8. [Système] notifie l'Administrateur **si et seulement si** le statut n'est pas *Complet*.
9. [Administrateur] consulte le rapport le lendemain et, en cas d'échec, corrige puis relance.

🔗 Cas d'étude : L'étape 8 mérite qu'on s'y arrête

L'étudiant écrit spontanément « le système notifie l'administrateur ». Karim, lui, a été très clair : « *Si je reçois un mail toutes les nuits, dans un mois je les filtre, et le jour où ça plante je ne le vois pas.* » **Notifier uniquement les échecs est une exigence métier, pas une paresse d'implémentation.** C'est le pendant exact du « pas de bouton du tout » de l'exemple d'introduction du cours.

12.3 P2 — Restauration d'urgence

1. [Chercheur] constate la disparition ou l'altération d'un fichier.
2. [Chercheur] liste les instantanés disponibles pour la source.
3. [Chercheur] identifie la date antérieure à l'incident.
4. [Chercheur] parcourt l'instantané comme un répertoire ordinaire (**ENF-COMP-01**) et localise le fichier.
5. [Chercheur] copie le fichier vers un emplacement de travail — **jamais** par-dessus l'original.
6. [Chercheur] vérifie le contenu avant de valider la restauration.

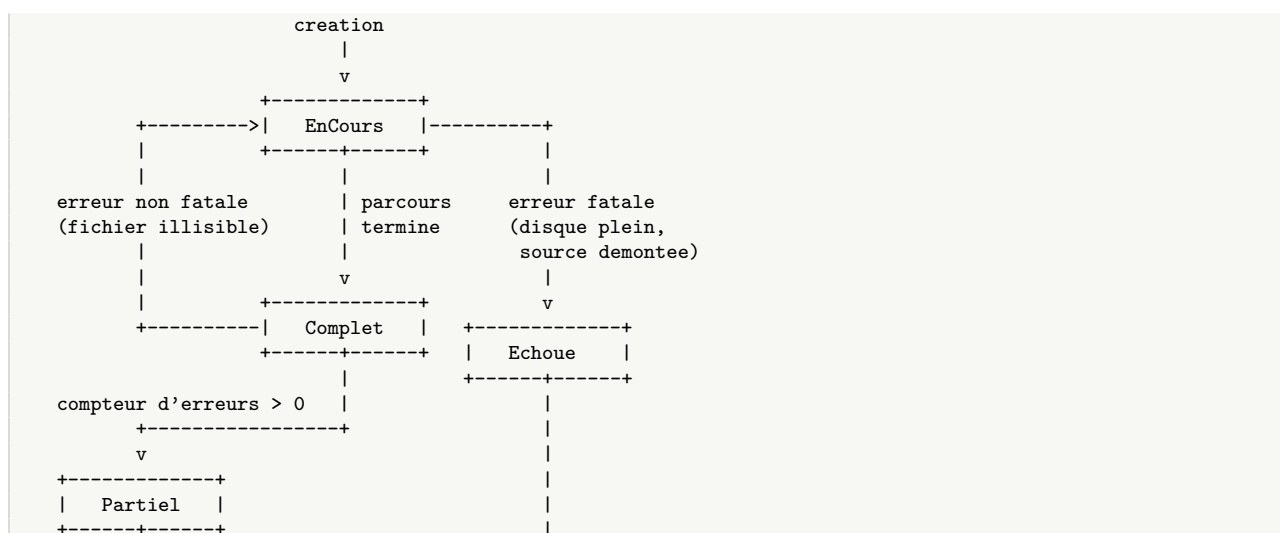
Ce processus a une conséquence forte sur l'architecture : les étapes 4 et 5 **n'utilisent pas l'outil**. C'est délibéré, et c'est ce qui a fait naître **ENF-COMP-01**. Un processus métier peut donc imposer une contrainte d'architecture qu'aucune exigence fonctionnelle n'exprimait.

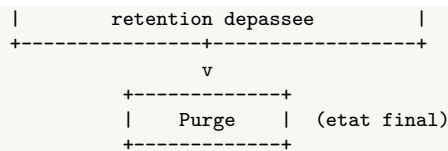
12.4 Ce que les processus ont ajouté au modèle de domaine

Découverte	Origine	Conséquence sur le diagramme
l'instantané a un statut	P1 (4, 6)	attribut statut sur Instantané
on veut mesurer la durée	P1 (4, 6)	attributs dateDebut / dateFin
seul un instantané <i>complet</i> peut servir de référence	P1 (3)	l'auto-association « dérive de » est contrainte par le statut
il faut connaître l' espace disponible avant de commencer	P1 (2)	attribut espaceDisponible sur Dépôt
le rapport est conservé et consulté plus tard	P1 (9), US-07	Rapport devient une classe, non un affichage éphémère

Cinq ajouts au modèle, tous issus de la seule description d'un processus. À faire constater aux étudiants avant/après : c'est la démonstration la plus convaincante de la boucle vertueuse statique ↔ dynamique.

12.5 L'automate à états de l'Instantané





Transitions interdites, à énoncer explicitement :

- *Complet* → *EnCours* : un instantané est immuable (**RG-01**). On refait un instantané, on ne répare pas l'ancien.
- *Purgé* → quoi que ce soit : état final.
- *Échoué* → *Complet* : un instantané échoué ne se rattrape pas.
- *Complet* → *Purgé* si c'est le dernier complet de la source : interdit par **RG-05**.

Ces quatre règles sont **invisibles sur le diagramme de classes**. Elles n'existent que parce qu'on a dessiné l'automate : c'est l'argument à donner aux étudiants pour justifier un troisième diagramme.

12.6 Tests d'intégration

On utilise ici le **vrai** adaptateur POSIX et un vrai répertoire temporaire. La vérification décisive porte sur les **numéros d'inode** : deux chemins partageant le même inode sont bien le même fichier physique — preuve directe que le mécanisme d'économie d'espace fonctionne.

```

#!/bin/sh
# test_integration.sh -- Verifie la collaboration reelle avec le systeme de
# fichiers POSIX (adaptateur fs_posix). Portable Linux / macOS : POSIX sh.
# (pas de set -e : chaque verification gere son propre code retour)
BIN=./snapguard
TMP=$(mktemp -d)
trap 'rm -rf "$TMP"' EXIT

ok=0; ko=0
verifier() { if [ "$1" = "0" ]; then echo " [OK]      $2"; ok=$((ok+1));
              else echo " [ECHEC] $2"; ko=$((ko+1)); fi }
inode() { ls -li "$1" | awk '{print $1}'; }

echo "=== Tests d'integration snapguard ==="

mkdir -p "$TMP/src/docs" "$TMP/depot"
printf 'alpha\n' > "$TMP/src/a.txt"
printf 'beta\n' > "$TMP/src/docs/b.txt"

echo "IT-1 : sauvegarde complete sur disque reel"
$BIN "$TMP/src" "$TMP/depot/j1" > "$TMP/sortie1" 2>&1
verifier $? "code de sortie 0"
[ -f "$TMP/depot/j1/a.txt" ] && [ -f "$TMP/depot/j1/docs/b.txt" ]
verifier $? "arborescence reproduite a l'identique"
cmp -s "$TMP/src/a.txt" "$TMP/depot/j1/a.txt"
verifier $? "contenu du fichier copie identique a la source"

echo "IT-2 : sauvegarde incrementale, aucun fichier modifie"
$BIN "$TMP/src" "$TMP/depot/j2" "$TMP/depot/j1" > "$TMP/sortie2" 2>&1
[ "$(inode "$TMP/depot/j1/a.txt")" = "$(inode "$TMP/depot/j2/a.txt")" ]
verifier $? "a.txt de j2 est un LIEN DUR vers celui de j1 (meme inode)"
grep -q "lies (inchanges) : 2" "$TMP/sortie2"
verifier $? "le rapport annonce 2 fichiers lies"

echo "IT-3 : un fichier est modifie entre deux instantanes"
printf 'alpha modifie\n' > "$TMP/src/a.txt"
$BIN "$TMP/src" "$TMP/depot/j3" "$TMP/depot/j2" > "$TMP/sortie3" 2>&1
[ "$(inode "$TMP/depot/j2/a.txt")" != "$(inode "$TMP/depot/j3/a.txt")" ]
verifier $? "a.txt modifie : nouvel inode, donc vraie copie"
cmp -s "$TMP/src/a.txt" "$TMP/depot/j3/a.txt"
verifier $? "j3 contient bien la nouvelle version"
cmp -s "$TMP/depot/j1/a.txt" "$TMP/depot/j2/a.txt"
verifier $? "j1 conserve intacte l'ancienne version (non-alteration du passe)"

echo "IT-4 : source inexistante"
if $BIN "$TMP/nexistepas" "$TMP/depot/j4" >/dev/null 2>&1; then false; else [ $? -eq 3 ]; fi
verifier $? "code de sortie 3 et message sur stderr"

echo ""
  
```

```
echo "$((ok+ko)) verifications, $ko echec(s)"
[ "$ko" -eq 0 ]
```

12.7 Tests d'acceptation

Structure Gherkin rendue exécutable en shell POSIX. Ces scénarios sont la traduction littérale des critères d'acceptation rédigés plus haut : **la spécification est le test.**

```
#!/bin/sh
# test_acceptation.sh -- Tests d'acceptation : on verifie la VALEUR promise par
# les User Stories, pas l'implementation. Structure Gherkin (Etant donne /
# Quand / Alors) rendue executable en shell POSIX.
BIN=./snapguard
TMP=$(mktemp -d)
trap 'rm -rf "$TMP"' EXIT

ok=0; ko=0
alors() { if [ "$1" = "0" ]; then echo "    ALORS $2 ..... OK"; ok=$((ok+1));
    else echo "    ALORS $2 ..... ECHEC"; ko=$((ko+1)); fi }
ko_taille() { du -sk "$1" | awk '{print $1}'; }

echo "FONCTIONNALITE : sauvegarde incrementale d'un repertoire de travail"
echo ""

echo "  SCENARIO 1 : une sauvegarde quotidienne ne recopie que le travail du jour"
mkdir -p "$TMP/travail" "$TMP/depot"
i=1; while [ $i -le 20 ]; do
    dd if=/dev/zero of="$TMP/travail/fichier$i.dat" bs=1024 count=100 2>/dev/null
    i=$((i+1))
done
echo "    ETANT DONNE un repertoire de travail de 20 fichiers de 100 Kio"
$BIN "$TMP/travail" "$TMP/depot/lundi" > /dev/null 2>&1
echo "    ET une premiere sauvegarde complete realisee lundi"
printf 'travail du mardi\n' >> "$TMP/travail/fichier7.dat"
echo "    QUAND je modifie 1 seul fichier puis relance la sauvegarde mardi"
$BIN "$TMP/travail" "$TMP/depot/mardi" "$TMP/depot/lundi" > "$TMP/rapport" 2>&1
res=$?

alors $res "la commande se termine en succes"
# Attention : "du -sk depot/mardi" compterait les fichiers lies a leur taille
# pleine. On mesure donc l'occupation REELLE du depot entier : dans une meme
# invocation, du(1) ne compte qu'une seule fois les inodes partagés.
t_lundi=$(ko_taille "$TMP/depot/lundi")
t_total=$(ko_taille "$TMP/depot")
[ "$t_total" -lt $(( t_lundi + t_lundi / 2 )) ]
alors $? "deux instantanes occupent presque l'espace d'un seul ($t_total Kio au lieu de $(( t_lundi * 2 )) Kio)"
grep -q "copies          : 1" "$TMP/rapport"
alors $? "le rapport indique qu'un seul fichier a ete recopie"
[ "$(ls "$TMP/depot/mardi" | wc -l)" -eq 20 ]
alors $? "mardi contient pourtant les 20 fichiers (restauration complete possible)"

echo ""
echo "  SCENARIO 2 : la sauvegarde tient dans la fenetre de nuit (ENF-PERF-01)"
mkdir -p "$TMP/gros"
i=1; while [ $i -le 500 ]; do printf 'x' > "$TMP/gros/f$i"; i=$((i+1)); done
echo "    ETANT DONNE un repertoire de 500 fichiers"
debut=$(date +%s)
$BIN "$TMP/gros" "$TMP/depot/perf" > /dev/null 2>&1
duree=$(( (date +%s) - debut )
echo "    QUAND je lance une sauvegarde complete"
[ "$duree" -le 10 ]
alors $? "elle s'acheve en moins de 10 secondes (mesure : ${duree}s)"

echo ""
echo "  SCENARIO 3 : le depot est utilisable avec les outils Unix habituels"
echo "    ETANT DONNE un instantane existant"
diff -r "$TMP/travail" "$TMP/depot/mardi" > /dev/null 2>&1
alors $? "diff -r ne signale aucun ecart avec la source"
tar cf "$TMP/archive.tar" -C "$TMP/depot" mardi 2>/dev/null
alors $? "l'instantane s'archive avec tar sans traitement particulier"

echo ""
echo "$((ok+ko)) verifications, $ko echec(s)"
[ "$ko" -eq 0 ]
```

12.8 Le Makefile

```
CC      = cc
CFLAGS  = -std=c11 -Wall -Wextra -Wpedantic -O2
BIN      = snapguard
TESTBIN  = run_tests

all: $(BIN)

$(BIN): src/main.c src/snapshot.c src/fs_posix.c
    $(CC) $(CFLAGS) -o $@ $^

$(TESTBIN): tests/test_snapshot.c tests/fs_fake.c src/snapshot.c
    $(CC) $(CFLAGS) -o $@ $^

test: $(TESTBIN)
    ./$$(TESTBIN)

integration: $(BIN)
    ./tests/test_integration.sh

acceptation: $(BIN)
    ./tests/test_acceptation.sh

clean:
    rm -f $(BIN) $(TESTBIN)

.PHONY: all test integration acceptation clean
```

12.9 Exécution réelle de la suite complète

Sortie obtenue avec `make test && make integration && make acceptation` (gcc 13, Linux x86-64) :

```
=== Tests unitaires snapguard ===

Test 1 : premiere sauvegarde -> tout est copie
[OK]    code de retour SG_OK
[OK]    3 fichiers examines
[OK]    3 fichiers copies
[OK]    aucun lien dur
[OK]    sous-repertoire docs recree
[OK]    recursivite : c.txt copie au bon endroit
Test 2 : fichier inchange -> lien dur (0 octet consomme)
[OK]    a.txt et b.txt sont lies
[OK]    seul c.txt (absent de j1) est copie
[OK]    le lien pointe bien vers l'instantane precedent
Test 3 : fichier modifie -> copie
[OK]    seul a.txt est lie
[OK]    b.txt (modifie) et c.txt sont copies
[OK]    b.txt a bien ete recopie depuis la source
Test 4 : echec du lien dur -> repli sur la copie (flux 6a)
[OK]    l'instantane reste un succes
[OK]    aucun lien comptabilise
[OK]    les 3 fichiers ont ete copies
Test 5 : source absente -> SG_ERR_SOURCE, aucune ecriture
[OK]    code SG_ERR_SOURCE
[OK]    aucun repertoire cree (pas d'effet de bord)
[OK]    aucune copie tentee

18 verifications, 0 echec(s)
=== Tests d'integration snapguard ===
IT-1 : sauvegarde complete sur disque reel
[OK]    code de sortie 0
[OK]    arborescence reproduite a l'identique
[OK]    contenu du fichier copie identique a la source
IT-2 : sauvegarde incrementale, aucun fichier modifie
[OK]    a.txt de j2 est un LIEN DUR vers celui de j1 (meme inode)
[OK]    le rapport annonce 2 fichiers lies
IT-3 : un fichier est modifie entre deux instantanes
[OK]    a.txt modifie : nouvel inode, donc vraie copie
[OK]    j3 contient bien la nouvelle version
[OK]    j1 conserve intacte l'ancienne version (non-alteration du passe)
IT-4 : source inexistante
[OK]    code de sortie 3 et message sur stderr
```

```

9 verifications, 0 echec(s)
FONCTIONNALITE : sauvegarde incrementale d'un repertoire de travail

SCENARIO 1 : 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 la commande se termine en succes ..... OK
ALORS deux instantanes occupent presque l'espace d'un seul (2116 Kio au lieu de 4008 Kio) ..... OK
ALORS le rapport indique qu'un seul fichier a ete recopie ..... OK
ALORS mardi contient pourtant les 20 fichiers (restauration complete possible) ..... OK

SCENARIO 2 : la sauvegarde tient dans la fenetre de nuit (ENF-PERF-01)
ETANT DONNE un repertoire de 500 fichiers
QUAND je lance une sauvegarde complete
ALORS elle s'acheve en moins de 10 secondes (mesure : 0s) ..... OK

SCENARIO 3 : le depot est utilisable avec les outils Unix habituels
ETANT DONNE un instantane existant
ALORS diff -r ne signale aucun ecart avec la source ..... OK
ALORS l'instantane s'archive avec tar sans traitement particulier ..... OK

7 verifications, 0 echec(s)

```

12.10 La démonstration à faire en amphi (5 minutes, effet garanti)

```

$ mkdir -p /tmp/demo/src /tmp/demo/depot
$ for i in 1 2 3; do dd if=/dev/zero of=/tmp/demo/src/f$i.dat bs=1024 count=500; done

$ ./snapguard /tmp/demo/src /tmp/demo/depot/lundi
$ echo "modif" >> /tmp/demo/src/f2.dat
$ ./snapguard /tmp/demo/src /tmp/demo/depot/mardi /tmp/demo/depot/lundi
Instantane : /tmp/demo/depot/mardi
fichiers examines : 3
copies : 1
lies (inchanges) : 2
repertoires crees : 0
erreurs : 0

$ ls -li /tmp/demo/depot/lundi /tmp/demo/depot/mardi
/tmp/demo/depot/lundi:
2802741 -rw-r--r-- 2 root root 512000 f1.dat      <-- 2 liens
2802742 -rw-r--r-- 1 root root 512000 f2.dat      <-- 1 lien (version de lundi)
2802743 -rw-r--r-- 2 root root 512000 f3.dat      <-- 2 liens

/tmp/demo/depot/mardi:
2802741 -rw-r--r-- 2 root root 512000 f1.dat      <-- MEME inode que lundi
2802745 -rw-r--r-- 1 root root 512006 f2.dat      <-- inode different : vraie copie
2802743 -rw-r--r-- 2 root root 512000 f3.dat      <-- MEME inode que lundi

$ du -sh /tmp/demo/depot/lundi /tmp/demo/depot/mardi
1.5M    /tmp/demo/depot/lundi
508K    /tmp/demo/depot/mardi

$ du -sh /tmp/demo/depot
2.0M    /tmp/demo/depot      <-- et non 3.0 Mio

```

Trois observations à faire commenter :

1. La colonne du **compteur de liens** (2 pour f1.dat) et l'**identité des inodes** sont la preuve physique du mécanisme.
2. du affiche 508 K pour mardi parce que, **dans une même invocation**, il ne compte qu'une fois chaque inode. Excellent piège : lancé séparément, du -sh mardi afficherait 1,5 M. *Quel chiffre est « le bon » ?* Les deux — la question mal posée, c'est « quelle place prend mardi ? » ; la bonne, c'est « quelle place prend le dépôt ? ».
3. **Le fichier de lundi n'a pas bougé** alors que la source a été modifiée : c'est le second critère d'acceptation de **US-01**, vérifiable à l'œil nu.

13 Ce qu'il reste à faire : le backlog

It.	Contenu	Compétence remobilisée
2	US-03 : exclusions (*.tmp, __pycache__) via fnmatch(3)	classe-association Inclusion ; règles de gestion
3	US-06 : statut, journalisation syslog(3), code de retour exploité par cron	automate à états ; ENF de fiabilité
4	EF-06 : purge selon rétention, en respectant RG-05	préconditions ; opérations destructrices ; --dry-run
5	US-04/US-05 : snapguard liste et localisation d'un fichier dans le temps	nouveaux cas d'utilisation ; acteur Chercheur
6	ENF-FIA-01 : écriture dans un répertoire temporaire puis rename(2) atomique	scénario 8b ; atomicité
7	RG-02 renforcée : option --verifier par empreinte SHA-256	bascule EF/ENF ; arbitrage coût/risque
8	ENF-PORT-01 : intégration continue Linux + macOS	portabilité ; éviter les #if defined(__APPLE__)

Cas d'étude : L'itération 6 est la plus formatrice

Elle consiste à découvrir que l'implémentation actuelle **viole ENF-FIA-01** : si snapguard est tué en cours de route, il laisse un instantané partiel qui *ressemble* à un instantané complet et pourra servir de référence au suivant, propageant l'erreur. La correction (écrire dans .snapguard-tmp/ puis rename() en fin de parcours) coûte dix lignes. **Faites-la trouver aux étudiants en leur demandant simplement : « quels tests manquent ? »** C'est la meilleure illustration possible du fait qu'une ENF non testée n'est pas une ENF satisfaite.

14 Notes pour la conduite des trois séances

14.1 CM4

Le modèle de domaine (25 min), la méthode de construction (20 min), les processus métier (15 min), l'automate à états (15 min).

✓ À faire : Préparez la chute du TP4

Terminez le CM4 sur l'affirmation que l'automate rend explicite : *un instantané interrompu ne peut pas être « Complet »*. Annoncez qu'au TP ils lanceront une sauvegarde, la tueront d'un kill -9, puis regarderont ce que leur outil a laissé sur le disque. C'est cette promesse qui donne son ressort à la dernière séance.

14.2 TD4

Tri des concepts (20 min), associations et multiplicités (20 min), processus nocturne (15 min), automate à états (20 min).

🔗 Cas d'étude : Le cas « Lien dur », troisième et dernier acte

Il figurera dans presque toutes les listes de substantifs : ils sortent de trois TP passés dessus. C'est le meilleur exercice de discrimination analyse / conception de tout le cours, précisément parce qu'ils y sont attachés.

L'argument qui emporte l'adhésion : **Karim ne dit jamais « lien dur »**. Il dit « je ne veux pas payer trente fois le même fichier ».

14.3 TP4

Expérience kill -9 et diagnostic (25 min), correction atomique (25 min), revue croisée notée (25 min).

✓ À faire : Laissez-les découvrir seuls

Ne dites pas ce que l'expérience va montrer. Laissez-les lancer la commande, lister depot, et constater que j2 ressemble en tout point à un instantané complet. Puis laissez-les construire j3 à partir de j2 et voir le diff -r échouer. La propagation du défaut d'une nuit sur l'autre est ce qui frappe le plus.

La revue croisée en 25 minutes

Les groupes s'apparient et échangent leurs dépôts. Insistez : **on n'ouvre pas le code, on exécute**. Ce qui est noté n'est pas la grille cochée mais les défauts trouvés au-delà d'elle, chacun accompagné de la commande qui le reproduit. Une revue qui ne trouve rien est une revue qui n'a pas cherché.

C'est aussi la seule occasion du semestre où ils voient un autre code que le leur.

Guide pédagogique

15 Matrice de traçabilité

À projeter en fin de cours : elle montre d'un seul coup d'œil que chaque artefact découle du précédent, et qu'aucun besoin n'est perdu en route. C'est aussi la grille de relecture d'un rendu étudiant.

Problème	Exigence	US	CU	Classe(s)	Conception	Test
P1 historique court	EF-02	US-01	CU-01 §6-8	Instantané, Source	<code>est_inchange()</code>	U2, U3, IT2, ACC1
P1	ENF-PERF-01	US-01	CU-01	—	boucle unique	ACC2
P2 restauration lourde	ENF-COMP-01	US-04	CU-05	Instantané	arborescence native	ACC3
P3 échec silencieux	EF-03, EF-08	US-06	CU-02	Rapport	<code>sg_rapport_t</code>	U5, IT4
P4 parc hétérogène	ENF-PORT-01	—	—	—	<code>fs_posix.c</code> isolé	CI 2 cibles
—	ENF-MAIN-01	—	—	—	<code>port sg_fs_port_t</code>	18 tests unitaires
RG-01 immuabilité	ENF-FIA-01	US-02	CU-01 §8b	automate Instantané	<i>itération 6</i>	IT3 (partiel)

Les deux lignes les plus instructives sont les vides. **ENF-MAIN-01** n'est rattachée à aucune *User Story* : c'est une exigence qui ne sert **aucun utilisateur**, seulement l'équipe. Elle est pourtant celle qui a déterminé toute l'architecture. À l'inverse, P4 ne produit aucun code spécifique : elle se traduit par une *discipline* (ne pas sortir de POSIX) et un dispositif de vérification.

16 Progression : quatre paires TD/TP de 1 h 15

Le fascicule étudiant tient en huit séances de 1 h 15, organisées en **paires** : un TD, puis immédiatement son TP. La contrainte que cela impose est forte et structure tout le découpage : **chaque TP ne doit être exploitable qu'avec le seul produit du TD qui le précède**. Aucun travail intermédiaire n'est possible à l'intérieur d'une paire, puisque les deux séances peuvent se suivre dans la même demi-journée.

Paire	Le TD produit...	... que le TP consomme le jour même
1	des exigences chiffrées	un <i>spike</i> qui les met à l'épreuve : mesure réelle, correction des seuils
2	le scénario nominal du cas d'utilisation	la liste des opérations système qu'il réclame, donc le port et son adaptateur
3	les scénarios d'exception (cartes incident) et la conception de la doublure	le code de l'incrémental et de ses replis, puis les mesures
4	l' automate à états de l'instantané	l'expérience qui prend le code en défaut sur cet automate, et sa correction

✓ À faire : Le fil de causalité entre les huit séances

C'est ce qui donne sa cohérence à l'ensemble, et il vaut la peine de l'expliciter aux étudiants à mi-parcours : la limite des liens durs *constatée* au TP1 devient la carte incident D du TD3, qui devient le repli codé au TP3, qui devient le test T4 rendu possible par la doublure conçue au TD3. De même, le fichier illisible de la carte B fait naître la notion de *statut*, qui devient l'automate du TD4, qui fournit au TP4 l'affirmation que l'expérience `kill -9` vient contredire.

Rien n'est introduit « parce que c'est au programme » : chaque notion arrive parce que la précédente l'a rendue nécessaire.

Séance	Ce qui se fait en séance	TAM après
TAM 0	lire le dossier BioSeq, surligner demande / besoin	45 min
TD1	5 Pourquoi, jeu de rôle de l'entretien, balayage ISO 25010	—
TP1	laboratoire des liens durs , <i>spike</i> de performance, socle du projet	1 h 30
TD2	correction croisée INVEST, débat sur la story « lien dur », Gherkin, scénario nominal	—
TP2	le port, l' adaptateur POSIX , <i>make</i> <i>verif</i>	2 h
TD3	jeu des pannes , arbitrages métier, conception de la doublure	—
TP3	RG-02 , liens durs, repli « carte D », laboratoire de mesures	2 h 30
TD4	tri des concepts, multiplicités, processus, automate à états	—
TP4	expérience <i>kill -9</i> , atomicité, revue croisée , soutenances	2 h 30
Total	encadré : 10 h travail personnel : ≈ 9 h	

17 Le travail en groupe de 3 ou 4

Sur 1 h 15 et un seul clavier, un groupe de quatre se transforme spontanément en un étudiant qui code et trois qui regardent. Le fascicule impose donc des **rôles nommés qui tournent à chaque séance** : en TD, *client* (qui refuse tout ce que Karim n'a pas dit), *scribe*, *avocat du diable*, *rapporteur* ; en TP, *pilote* (qui n'écrit que ce qui a été dit à voix haute), *copilote*, *testeur*, *intendant*.

🔗 Cas d'étude : Le rôle de « client » vaut à lui seul l'organisation en groupe

Confier à un étudiant la mission de **refuser** toute affirmation qui ne figure pas dans la transcription de l'entretien produit un effet immédiat : le groupe cesse d'inventer les besoins du client. C'est le remède le plus efficace au défaut le plus répandu en analyse — prêter au client des intentions qu'il n'a jamais formulées. Insistez pour que ce rôle tourne : chacun doit avoir été, une fois, celui qui dit « il ne l'a jamais dit ».

✗ À ne pas faire : La répartition qui fait échouer les groupes

« Toi le code, moi les tests, vous le dossier. » Deux séances plus tard, personne ne comprend plus le travail des autres, le dossier décrit un code qui n'existe pas, et la soutenance individuelle est catastrophique pour trois membres sur quatre. Le fascicule l'interdit explicitement : **un seul écran, ensemble**, sauf indication contraire. Seules les variantes personnelles se traitent séparément, et c'est délibéré.

18 Évaluation : trois notes de groupe, une note individuelle

Note	Porte sur	Rendu	Coef.
G1	Spécification : exigences, backlog, cas d'utilisation complet (TD1–TD3)	avant le TD4	2
G2	Prototype : code, tests, modèle de domaine, traçabilité	final	3
G3	Revue croisée : le rapport produit sur l'outil d'un autre groupe	fin du TP4	1
I	Contribution personnelle : variante personnelle, évaluée sur le journal Git	final	2

Les grilles détaillées figurent dans le fascicule étudiant. Trois remarques sur les choix qui les sous-tendent.

Pourquoi G1 est rendue avant le TD4

Pour que le retour arrive alors qu'il peut encore servir. Un dossier d'analyse rendu en même temps que le code n'est plus corrigé que pour la note ; rendu à mi-parcours, il permet de reprendre une exigence mal formulée avant qu'elle ne se propage dans le prototype. C'est la courbe de Boehm appliquée à l'évaluation elle-même.

Pourquoi la revue croisée est notée

Une revue non notée devient une formalité : on coche, on rend, on passe. En la notant sur la **qualité des défauts trouvés** — deux au minimum au-delà de la grille, chacun avec la commande qui le reproduit — on obtient un exercice de test exploratoire sérieux, et une note qui se corrige en dix minutes puisqu'elle est remise le jour même. C'est aussi la seule occasion pour les étudiants de voir un autre code que le leur.

La note individuelle sans oral : l'audit du dépôt

Une soutenance individuelle de cinq minutes coûte deux heures pour un groupe de vingt-quatre étudiants. Si ce temps n'est pas disponible, la note individuelle se fonde entièrement sur le **journal Git** — à condition d'en faire un instrument sérieux plutôt qu'un comptage de commits.

Le script `outils/audit-git.sh` produit, pour un dépôt donné, sept sections : commits et volume de lignes par auteur, répartition dans le temps, présence à chacune des quatre séances, nombre de fichiers distincts touchés, qualité des messages, et cohérence des identités.

La fraude tentée	Ce qui la révèle
cinquante commits d'une ligne la veille du rendu	répartition dans le temps (§3) et volume de lignes (§2)
un membre qui ne fait que sa variante	nombre de fichiers distincts touchés (§5)
un membre absent de deux séances sur quatre	présence par date de TP (§4)
un étudiant qui committe pour un camarade	cohérence nom / courriel (§7), et la matrice de contribution

✗ À ne pas faire : Les trois biais qu'il faut corriger à la main

Git mesure ce qui a été *écrit*, pas ce qui a été *compris*.

- L'étudiant qui a porté l'analyse et peu codé apparaît faible.
- Le *pair programming* bien mené produit des commits d'un seul auteur pour un travail à deux.
- Un membre peut committer sous le nom d'un autre.

C'est pourquoi la **matrice de contribution signée** reste obligatoire au rendu : elle permet au groupe de déclarer ce que Git ne voit pas. En cas d'écart entre la matrice et l'audit, cinq minutes avec le groupe suffisent — et cela ne concerne qu'une poignée de groupes, pas tous.

🔍 Cas d'étude : Le point de contrôle est au TP1, pas au TP4

Tout repose sur une vérification de cinq minutes en début de projet : passer dans les rangs au TP1 et faire afficher `git config user.email` sur chaque poste. Le point d'arrêt du TP1 exige d'ailleurs « au moins un commit par membre présent ».

Sans cela, vous découvrirez au moment de noter un historique à un seul auteur — et il sera trop tard pour le reconstituer. **Une note individuelle fondée sur Git se prépare à la première séance, pas à la dernière.**

✓ À faire : Si vous récupérez un jour du temps d'oral

Le dispositif le plus efficace contre le passager clandestin reste la **question tirée au hasard dans le dossier du groupe** : elle porte sur le travail collectif, pas sur la contribution personnelle, et distingue celui qui a participé aux décisions de celui qui a exécuté une tâche. Quinze questions sont proposées dans le matériel de séance.

Version dégradée à coût quasi nul : les faire répondre *par écrit*, en dix minutes, en ouverture du TP4 — une question tirée au sort par étudiant, correction au vol.

19 Exercices étudiants, calqués sur ceux du cours

Exercice A — L'analyste-enquêteur

Énoncé

Karim ajoute : « Et je veux aussi un voyant vert sur mon écran qui me dit que la sauvegarde d'hier soir s'est bien passée. »

1. Quelle est la demande, quelle solution présuppose-t-elle ?
2. Formulez deux hypothèses sur le besoin réel.
3. Rédigez trois questions ouvertes pour trancher.
4. Sachant que Karim n'allume son poste qu'à 9 h 30 et qu'il est en déplacement deux jours par semaine, que proposeriez-vous à la place d'un voyant ?

Piste de correction : le besoin n'est pas de *voir* que ça a marché, mais **d'être prévenu quand ça n'a pas marché**, où qu'il soit. Un voyant est une solution *pull* pour un besoin *push*. C'est le « bouton PDF » du cours, transposé.

Exercice B — Classification EF / ENF

Classez et justifiez en une phrase :

1. « Le système doit créer un lien dur plutôt qu'une copie pour un fichier inchangé. »
2. « Un instantané de 500 000 fichiers doit s'achever en moins de 30 minutes. »
3. « Le système doit exclure les chemins correspondant aux motifs de la politique. »
4. « Le dépôt doit être créé avec les permissions 0700. »
5. « Le système doit signaler par un code de retour non nul un instantané incomplet. »
6. « L'outil doit compiler sur Linux et macOS sans directive conditionnelle. »
7. « Le système doit permettre de restaurer un fichier depuis un instantané choisi. »
8. « Un instantané interrompu ne doit jamais rendre illisibles les instantanés antérieurs. »

Le n°1 est le piège : formulée ainsi, c'est **une solution déguisée en exigence**. La bonne exigence est **EF-02** ; le lien dur est un choix de conception. Faites-la corriger, pas seulement classer.

Exercice C — Rédiger un cas d'utilisation

Rédigez CU-05 « Restaurer un élément » : acteur, préconditions, scénario nominal, **au moins quatre scénarios d'exception**, postconditions. Indices : que se passe-t-il si un fichier du même nom existe déjà à la destination ? si le chercheur n'a pas le droit de lire l'instantané d'un collègue ? s'il restaure un répertoire de 80 Gio sur un disque qui en offre 20 ?

Exercice D — Le modèle du domaine

Un étudiant propose :

```
Instantane { chemin, listeInodes, fdSource, bufferCopie[65536] }
1 -- 1 GestionnaireDeFichiers { copier(), lierDur(), fermer() }
```

Relevez **quatre erreurs**, en distinguant celles qui relèvent de la confusion analyse/conception de celles qui relèvent d'une mauvaise identification des concepts.

Exercice E — L'automate à états

Karim demande une reprise sur incident : « si la sauvegarde plante à 80 %, je veux pouvoir la reprendre où elle en était le lendemain ».

1. Dessinez la transition que cette demande introduit dans l'automate.
2. Quelle règle de gestion cette transition viole-t-elle ?
3. Proposez une modélisation qui satisfait le besoin **sans violer RG-01**.

Piste : l'instantané reste immuable ; c'est un **nouvel** instantané qui prend le précédent partiel comme référence. On ne répare pas le passé, on repart de lui. Excellente occasion de montrer qu'une contrainte de modélisation, loin de bloquer, oriente vers une meilleure solution.

Exercice F — Doublures de test

Dans `test_snapshot.c`, identifiez pour chacune des 18 assertions s'il s'agit d'une vérification **d'état** (stub) ou **d'interaction** (mock). Puis : quelle assertion casserait si l'on remplaçait `link()` par `clonefile()` sous macOS, alors même que le comportement métier resterait correct ? Que cela dit-il de la fragilité des tests d'interaction ?

20 Grille d'évaluation d'un dossier d'analyse

Critère	Indicateur de réussite	Pts
Enquête	la demande initiale est distinguée du besoin, avec traçabilité de l'écart	3
Exigences	toutes les ENF sont mesurables ; les 8 familles ISO 25010 revues	4
User Stories	rôles réels et distincts ; bénéfice métier ; INVEST vérifié sur 2 US	3
Cas d'utilisation	≥ 3 scénarios d'exception non triviaux	3
Modèle de domaine	aucune classe technique ; multiplicités justifiées ; classe-association pertinente	4
Processus & états	≥ 1 enrichissement du modèle explicitement issu d'un processus	3
Traçabilité	matrice besoin $\rightarrow$ exigence $\rightarrow$ US $\rightarrow$ CU $\rightarrow$ classe $\rightarrow$ test	3
Prototype	compile sans avertissement ; règles de gestion identifiables comme fonctions	4
Tests	les 3 niveaux présents ; ≥ 1 scénario d'exception testé	3
Total		30

21 Un mot sur le choix du C

Trois arguments à donner aux étudiants qui demanderont « pourquoi pas Java ou Python, ce serait plus court ? » :

1. **L'absence d'interface rend le principe visible.** En Java, implements fait le travail et l'inversion des dépendances passe pour une formalité syntaxique. En C, il faut la construire à la main — donc la comprendre. Les étudiants qui auront écrit `struct sg_fs_port` ne confondront plus jamais un principe de conception avec un mot-clé.
2. **La proximité du système rend les ENF tangibles.** « Fiabilité » reste une abstraction jusqu'au jour où l'on comprend qu'un `close()` peut échouer après des `write()` réussis.
3. **Rien n'est caché.** Pas de ramasse-miettes, pas de framework : entre l'exigence et l'appel système, il n'y a que le code de l'étudiant. La chaîne besoin $\rightarrow$ code est parcourable d'un bout à l'autre en une séance.

Annexe — Utilisation du prototype

```
$ make          # construit ./snapguard
$ make test     # tests unitaires (aucun acces disque)
$ make integration # tests d'integration (repertoire temporaire)
$ make acceptance # scenarios Gherkin de US-01

$ ./snapguard /home/karim/data /mnt/nas/depot/2026-08-05
$ ./snapguard /home/karim/data /mnt/nas/depot/2026-08-06 /mnt/nas/depot/2026-08-05
```

Intégration cron correspondant à la Politique du laboratoire :

```
# sauvegarde quotidienne a 2 h ; alerte uniquement en cas d'echec (P1 etape 8)
0 2 * * * /usr/local/bin/sauvegarde-nuit.sh || \
    echo "Echec snapguard $(date)" | mail -s "Alerte sauvegarde" karim@bioseq.fr
```

Code	Signification
0	instantané complet
1	instantané partiel (au moins une erreur non fatale)
2	arguments invalides
3	source illisible
4	dépôt non créable