

Guide pédagogique

Organisation des douze séances, évaluation, exercices

Ressource R3.03 — BUT2 Informatique

September 1, 2026

Contents

1	Matrice de traçabilité	1
2	Progression : quatre paires TD/TP de 1 h 15	1
3	Le travail en groupe de 3 ou 4	2
4	Évaluation : trois notes de groupe, une note individuelle	2
5	Exercices étudiants, calqués sur ceux du cours	4
6	Grille d'évaluation d'un dossier d'analyse	5
7	Un mot sur le choix du C	5

1 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.

2 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'explicitier 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 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	

3 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.

4 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.

5 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 ?

6 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 → exigence → US → CU → classe → 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

7 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 → 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