Formaliser un scénario et le contrat d'une fonction

Analyse : de l'exigence à la spécification

Ressource R3.03

BUT Informatique, début de deuxième année

Version du 24 septembre 2026

D'une intention à une décision précise

Question du cours

Comment écrire assez précisément une règle pour qu'un client puisse la valider et qu'une équipe puisse la tester ?

Nous poursuivons le cas de la bibliothèque avec le renouvellement d'un emprunt.

- La story exprime l'utilité.
- Le scénario décrit l'interaction.
- Le contrat et les exemples rendent une décision vérifiable.

Déroulé de la séance

#	Partie	Durée
1	Rappel et objectif utilisateur	10 min
2	Stories et critères d'acceptation	20 min
3	Scénario nominal et contrat	15 min
4	Table de décision et exercice	15 min
5	Lecture C : adresses et const	10 min
6	Synthèse et passage au TD	5 min

Une story explicite un bénéfice

Exemple

En tant que lecteur, je veux demander la prolongation de mon emprunt afin de terminer ma lecture sans me déplacer inutilement.

Acteur	Qui rencontre le problème ?
Action	Que souhaite-t-il accomplir ?
Bénéfice	Pourquoi cette action lui est-elle utile ?

La story n'indique pas encore combien de prolongations sont autorisées.

Une story ouvre une conversation

Formulation insuffisante	**Question de relecture**
En tant que développeur, je veux une fonction `renouveler`.	Quel service attend le lecteur ?
En tant que lecteur, je veux gérer les emprunts.	Quelle action termine-t-il dans une interaction ?
En tant que lecteur, je veux renouveler pour renouveler.	Quel bénéfice concret manque ?

Une tâche technique peut être nécessaire. La nommer tâche technique évite de lui inventer un bénéfice utilisateur.

INVEST sert à relire, sans remplir six cases

Indépendante	Peut-on discuter cette évolution sans tout réaliser ?
Négociable	Le besoin laisse-t-il place à une discussion sur la solution ?
Porteuse de valeur	Le bénéfice de l'acteur est-il identifiable ?
Estimable	A-t-on assez d'informations pour envisager le travail ?
Petite	L'action est-elle assez limitée pour être réalisée et vérifiée ?
Testable	Quels exemples permettraient de l'accepter ou de la refuser ?

Ici, « gérer tous les emprunts » demanderait un découpage.

Les règles viennent de la conversation

Complément au cas fictif de bibliothèque

Un emprunt peut être renouvelé au plus deux fois. Un exemplaire réservé par un autre lecteur ne peut pas être renouvelé. Un refus ne change pas son échéance.

- Deux règles déterminent l'autorisation.
- Une garantie décrit le résultat du refus.
- Il reste à préciser de combien de jours on prolonge l'emprunt.

Une question non résolue est inscrite comme telle.

Un critère d'acceptation est un exemple observable

Cas accepté

Étant donné un emprunt actif, renouvelé une fois, sans réservation par un autre lecteur, quand le lecteur demande un renouvellement, alors le système accepte la demande et applique la durée convenue.

Cas refusé

Étant donné un emprunt déjà renouvelé deux fois, quand le lecteur demande un renouvellement, alors le système refuse et conserve l'échéance précédente.

Cette forme structurée reste lisible sans outil d'automatisation.

L'acceptation porte sur une conséquence

Test trop faible

« Quand je clique, la fonction de renouvellement est appelée. »

Cela décrit une interaction interne, sans démontrer le service rendu.

Observation utile

Le résultat annoncé correspond à la règle et les données évoluent comme prévu. Après un refus, l'échéance et le compteur sont inchangés.

Un cas d'utilisation décrit un objectif complet

Nom	Demander un renouvellement
Acteur principal	Lecteur
Déclencheur	Le lecteur souhaite garder l'exemplaire plus longtemps.
Préconditions	Le lecteur est identifié et l'emprunt est actif.
Postcondition de succès	La durée convenue s'ajoute à l'échéance et le compteur augmente d'une unité.
Garantie de refus	L'emprunt reste inchangé et le lecteur reçoit la raison du refus.

Le scénario nominal reste au niveau de l'utilisateur

1. Le lecteur sélectionne un de ses emprunts actifs.
2. Le système présente son échéance et ses conditions de renouvellement.
3. Le lecteur demande le renouvellement.
4. Le système vérifie les règles et enregistre la nouvelle échéance.
5. Le système confirme la nouvelle échéance au lecteur.

Les ouvertures de fichiers et les appels de fonctions appartiennent à une description de conception. Ils n'aident pas le lecteur à valider ce scénario.

Le contrat de fonction porte sur une décision locale

Décision isolée

« Cet emprunt est-il renouvelable ? »

Entrées Emprunt existant, actif, compteur cohérent, information de réservation connue.

Résultat Vrai si le compteur est inférieur à deux et si aucune réservation par un autre lecteur ne bloque.

Effet de bord Aucun. Décider ne renouvelle pas l'emprunt.

La fonction de décision ne réalise donc pas tout le cas d'utilisation.

La table rend les combinaisons visibles

Réservation bloquante	**Compteur**	**Autoriser**
Non	0 ou 1	Oui
Non	2	Non
Oui	0 ou 1	Non
Oui	2	Non

Le contrat exclut ici les compteurs incohérents. Un autre module garantit cet invariant, ou l'interface doit prévoir leur rejet.

Exercice : un seul exemple peut masquer une erreur

Quatre minutes, puis comparaison à deux.

1. Proposer une condition erronée qui accepte correctement le premier cas de la table.
2. Choisir un cas qui la distingue de la règle attendue.
3. Indiquer les données qui ne doivent pas changer pendant la décision.

Écrire le résultat attendu avant de choisir une syntaxe C.

Correction : faire varier une condition à la fois

Erreur possible

Autoriser dès que le compteur est inférieur à deux, en oubliant la réservation.

Contre-exemple

Compteur égal à un, réservation bloquante présente. La règle exige un refus, la condition incomplète accepte.

Tester aussi le seuil exact : 1 est admissible, 2 ne l'est plus. La décision ne change ni compteur, ni échéance, ni réservation.

Un prototype permet de lire le contrat C

```c
typedef struct {
    int nb_renouvellements;
    int reservation_bloquante;
} emprunt_t;

int renouvelable(const emprunt_t *e);
```

- `emprunt_t` : un type composé de deux champs.
- `e` : l'adresse d'un emprunt, et non une copie de ses champs.
- `const` : cette fonction ne peut pas modifier les champs à travers `e`.

Objet, adresse et accès au champ

```c
emprunt_t pret = { .nb_renouvellements = 1,
                   .reservation_bloquante = 0 };
const emprunt_t *p = &pret;
/* Ces deux lectures donnent la meme valeur. */
int a = pret.nb_renouvellements;
int b = p->nb_renouvellements;
```

`pret.champ` part d'un objet. `p->champ` part d'une adresse valide. Cela équivaut à `(*p).champ`. Les parenthèses comptent.

L'absence doit avoir une signification définie

```c
int renouvelable(const emprunt_t *e) {
    if (e == NULL) return 0;
    return e->nb_renouvellements < 2
        && !e->reservation_bloquante;
}
```

On enrichit ici le contrat : une adresse absente donne un refus. La garde précède toute lecture d'un champ. `NULL` ne représente ni un emprunt vide ni une structure remplie de zéros.

Un test met en relation contrat et valeurs

```c
#include <assert.h>
emprunt_t e = { .nb_renouvellements = 1,
                .reservation_bloquante = 0 };
assert(renouvelable(&e) == 1);
e.reservation_bloquante = 1;
assert(renouvelable(&e) == 0);
assert(renouvelable(NULL) == 0);
```

Question : quel cas de la table manque encore ?

Un test met en relation contrat et valeurs

```c
#include <assert.h>
emprunt_t e = { .nb_renouvellements = 1,
                .reservation_bloquante = 0 };
assert(renouvelable(&e) == 1);
e.reservation_bloquante = 1;
assert(renouvelable(&e) == 0);
assert(renouvelable(NULL) == 0);
```

Question : quel cas de la table manque encore ? **Réponse :** le seuil de deux renouvellements sans réservation. Un test réussi ne prouve que ce qu'il observe.

La modification commence par une prédiction

1. Lire le contrat et choisir un cas qui pourrait échouer.
2. Prédire ce que calcule la version actuelle.
3. Exécuter le test et expliquer l'écart.
4. Modifier la fonction, puis rejouer les autres cas.

Question de revue

Pourquoi cette condition correspond-elle à la table ? La réponse doit mentionner les entrées et le résultat, pas seulement « le test passe ».

La méthode à conserver

Story	Qui veut faire quoi, pour quel bénéfice ?
Critère	Dans cette situation, quelle conséquence observe-t-on ?
Cas d'utilisation	Quelles interactions permettent d'atteindre l'objectif ?
Contrat local	Quelles entrées, quel résultat, quels effets autorisés ?
Table de décision	Quelles combinaisons doivent produire des résultats différents ?

En C : distinguer l'objet de son adresse, expliquer `const`, vérifier l'absence avant l'accès au champ.

Transposition au TD2 et au TP2

☞ Votre dossier : BioSeq

Vous écrirez deux stories et le scénario nominal d'une sauvegarde. Un complément au dossier précisera une règle locale à formaliser par une table et un contrat.

Le TP utilisera cette table

Vous auditerez une condition dans le code fourni, retrouverez son appel et compléterez les tests. La copie des fichiers et les chemins sont déjà pris en charge.