Décrire les exceptions et les rendre testables

Analyse : de l'exigence à la spécification

Ressource R3.03

BUT Informatique, début de deuxième année

Version du 24 septembre 2026

Ce qui peut mal se passer appartient au contrat

Question du cours

Après un incident, que peut-on encore promettre à l'utilisateur et comment le vérifier sans attendre la panne réelle ?

Nous reprenons le renouvellement d'un emprunt et sa notification.

- Une exception précise un autre déroulement du scénario.
- Un test doit contrôler l'incident et observer ses conséquences.
- Un appel indirect permet de remplacer une opération dans un test.

Déroulé de la séance

#	Partie	Durée
1	Relecture du scénario nominal	10 min
2	Exceptions et garanties après échec	20 min
3	Niveaux de test	15 min
4	Pointeurs de fonctions et doublure	15 min
5	Exercice de trace et correction	10 min
6	Synthèse et passage au TD	5 min

Chaque étape suggère des incidents

Étape du renouvellement	**Question d'exception**
Retrouver l'emprunt	L'emprunt existe-t-il encore et appartient-il au lecteur ?
Vérifier les règles	Une réservation bloque-t-elle le renouvellement ?
Enregistrer l'échéance	Que se passe-t-il si l'enregistrement échoue ?
Confirmer le résultat	Que se passe-t-il si l'envoi de la notification échoue ?

Une violation de règle métier et une panne technique peuvent demander des résultats différents.

Un flux alternatif est rattaché à une étape

Exemple : réservation bloquante à l'étape 3

3a. Le système détecte une réservation par un autre lecteur.
3a1. Il refuse le renouvellement et en donne la raison.
3a2. Il conserve le compteur et l'échéance. Le scénario se termine.

- Le point de départ est explicite.
- Les actions et l'état final sont observables.
- On sait si le scénario reprend ou se termine.

La panne ne dicte pas toute seule la décision métier

L'échéance a été enregistrée, mais le courriel échoue

Faut-il annuler le renouvellement, recommencer l'envoi ou conserver le renouvellement avec un avertissement ?

Discussion, trois minutes : pour chaque choix, quelle conséquence pour le lecteur et pour le personnel ? La bibliothèque doit trancher. Une API qui renvoie `-1` ne choisit pas cette politique à sa place.

Une décision validée fournit deux garanties distinctes

Décision retenue dans notre cas fictif

Le renouvellement reste acquis après un échec de notification. Le système signale que l'envoi a échoué. Il ne prétend pas que le lecteur a reçu le message.

- État métier : nouvelle échéance conservée.
- État de notification : échec visible.

La possibilité d'une nouvelle tentative d'envoi doit avoir son propre contrat, notamment pour éviter les envois multiples.

Une postcondition d'échec décrit aussi les effets interdits

Incident	**Garantie à tester**
Refus de renouvellement	Aucun changement d'échéance ou de compteur.
Échec d'enregistrement	Aucune confirmation mensongère de réussite.
Échec de notification	Renouvellement conservé, échec d'envoi signalé.

« Le programme ne plante pas » ne décrit aucune de ces garanties.

Les trois niveaux répondent à trois questions

Niveau	**Question**	**Exemple**
Unitaire	La règle locale calcule-t-elle le bon résultat ?	Refus au deuxième renouvellement déjà consommé.
Intégration	Les composants coopèrent-ils correctement ?	Lecture, modification et relecture d'un emprunt dans un fichier temporaire.
Acceptation	Le service satisfait-il le besoin exprimé ?	Un lecteur obtient une nouvelle échéance conforme à sa demande autorisée.

Le niveau dépend de ce que l'on cherche à prouver

- Beaucoup de règles peuvent se tester avec quelques valeurs, sans entrée-sortie.
- Certaines garanties exigent un vrai fichier ou plusieurs composants.
- Quelques scénarios complets vérifient la cohérence du service rendu.

Limite d'une doublure

Un test avec un faux service d'envoi ne prouve pas que le serveur réel accepte le message. Il prouve le comportement du composant face à la réponse choisie.

Un test a des entrées contrôlées et une observation

Préparer Un emprunt renouvelable, une ancienne échéance et un service d'envoi configuré pour échouer.

Agir Demander le renouvellement.

Vérifier L'échéance a changé, le résultat métier est acquis et l'échec de notification est visible.

Un test doit pouvoir échouer

Vérifier seulement que « la fonction a renvoyé une valeur » laisse passer presque toutes les erreurs intéressantes.

Une adresse de fonction peut changer l'opération appelée

```c
int ajouter(int a, int b) { return a + b; }
int retrancher(int a, int b) { return a - b; }

int (*operation)(int, int) = ajouter;
int x = operation(7, 3);
operation = retrancher;
int y = operation(7, 3);
```

x vaut 10 et y vaut 4. La variable contient une adresse de fonction compatible avec la signature annoncée.

Les parenthèses changent la déclaration

```c
int (*operation)(int, int); /* pointeur vers une fonction */
int *operation2(int, int);  /* fonction renvoyant int *   */
```

Lecture à partir du nom : `operation` est un pointeur vers une fonction qui prend deux entiers et renvoie un entier.

Avant l'appel

Le pointeur doit désigner une fonction valide de type compatible. Une déclaration seule ne l'initialise pas.

Une structure peut porter le contrat d'une opération

```c
typedef struct {
    void *ctx;
    int (*envoyer)(void *ctx, const char *message);
} notification_t;

int avertir(const notification_t *n) {
    return n->envoyer(n->ctx, "Nouvelle echeance");
}
```

`n` pointe vers la structure. `n->envoyer` désigne une fonction. `n->ctx` transporte l'état de son implémentation. Contrat : 0 signifie envoi réussi, -1 signifie échec.

Le contexte transporte un état de test

```c
typedef struct { int echec; int appels; } simulation_t;

int envoyer_simule(void *ctx, const char *message) {
    simulation_t *s = ctx;
    (void)message;
    s->appels++;
    return s->echec ? -1 : 0;
}
```

Le montage doit fournir l'adresse d'un `simulation_t` vivant pendant l'appel. `void *` ne permet pas de vérifier automatiquement ce contrat de type et de durée de vie.

Le montage choisit l'implémentation

```
simulation_t s = { .echec = 1, .appels = 0 };
notification_t n = {
    .ctx = &s,
    .envoyer = envoyer_simule
};
int code = avertir(&n);
assert(code == -1);
assert(s.appels == 1);
```

Aucun courriel ne part. L'appel traverse pourtant la même interface que dans le programme qui utilise un adaptateur réel.

Les doublures ont des usages différents

Réponse programmée, ou stub Fournir une valeur choisie, par exemple un échec d'envoi.

Observation des appels Compter ou enregistrer les interactions pour les vérifier. Une doublure avec attentes peut jouer le rôle de mock.

Implémentation simplifiée, ou fake Reproduire assez de fonctionnement pour un test, par exemple un stockage en mémoire.

Notre simulation combine une réponse programmée et un compteur. Elle ne simule pas tout un serveur. Le test reste responsable des assertions.

Pourquoi isoler l'opération ?

Une dépendance passée en paramètre

Le service reçoit le contrat de notification. Le montage choisit l'adaptateur réel ou la doublure.

- Le service n'a pas besoin de connaître le protocole d'envoi.
- Le test peut provoquer une panne à chaque exécution.
- Le changement de réponse ne demande pas de modifier la règle métier.

C'est une application locale de l'inversion des dépendances. Elle a un coût de lecture : le chemin de l'appel doit être documenté.

Exercice : suivre l'état après deux appels

```
simulation_t s = { .echec = 1, .appels = 0 };
notification_t n = { .ctx = &s, .envoyer = envoyer_simule };
int a = avertir(&n);
s.echec = 0;
int b = avertir(&n);
```

Quatre minutes : valeurs de `a`, `b` et `s.appels` ? Quelle fonction s'exécute ? Quelles adresses sont des adresses de données ? Que changerait l'affectation d'une autre fonction compatible à `n.envoyer` ?

Correction : distinguer l'état et le comportement

- `a = -1`, `b = 0`, `s.appels = 2`.
- Les deux appels exécutent `envoyer_simule`.
- `&n` et `&s` sont des adresses de données.
- Le champ `envoyer` contient une adresse de fonction.

Modifier `s.echec` change l'état lu par la même fonction. Modifier `n.envoyer` peut choisir une autre fonction. Ce sont deux opérations différentes.

Des assertions d'état et d'absence se complètent

Assertion	**Ce qu'elle vérifie**
Échéance égale à la valeur attendue	La conséquence métier.
Code de notification égal à échec	La visibilité du résultat technique.
Compteur d'envois égal à zéro après refus	L'absence d'un effet interdit.

Éviter de figer des détails d'appel sans lien avec le contrat. Sinon une amélioration interne peut casser un test alors que le service reste correct.

Transposition au TD3 et au TP3

☞ Votre dossier : BioSeq

Vous étudierez trois incidents. Pour chacun : étape concernée, résultat attendu, effets interdits et test.

Le TP utilisera vos flux alternatifs

Vous suivrez un appel à travers le port fourni, changerez une opération dans un montage de test et corrigerez une décision de repli. La doublure est déjà écrite.