COURS

Écrivez du code back-end en .NET C#

P1C2 - Repérez ce qui rend un code difficile à maintenir

Vérifiez votre travail

1. La divergence. `GetBalance` arrondit le nombre de mois au mois supérieur avec `Math.Ceiling` ; `Approve` l'arrondit au mois inférieur avec `Math.Floor`. Pour Leïla, l'affichage compte 8 mois, soit 16,64 jours de droits et un solde de 6,64 jours. L'approbation n'en compte que 7, soit 14,56 jours et un solde de 4,56 jours, insuffisant pour les 5 jours demandés. L'écart vaut exactement un mois de droits, 2,08 jours.

Avez-vous aussi remarqué l'alternance ? Une copie écrit `25 + 5`, l'autre `30`. Le résultat est aujourd'hui identique, mais rien ne garantit que les deux écritures évolueront ensemble. C'est une divergence en germe.

2. Le calcul correct. La règle compte les mois complets : c'est `Math.Floor`, donc `Approve`, qui la respecte. Le bug vient de l'affichage, qui promet un solde que la salariée n'a pas. Leïla n'avait donc pas droit à ses 5 jours : le refus était juste, c'est l'affichage qui était faux.

3 et 4. Le diagnostic. Voici une proposition :

Problème	Critère	Conséquence pour l'ajout des stages	Type
`Approve` recherche, calcule, enregistre, notifie et répond	Responsabilité unique	La règle du stage doit être ajoutée dans une méthode qui gère aussi la persistance et la notification	Organisation
Le calcul des droits est copié dans deux méthodes et les copies ont divergé	DRY	La règle du stage doit être écrite deux fois, avec le risque d'une nouvelle divergence	Organisation
`HamacDbContext` et `NotificationService` sont créés avec `new`	Couplage	Impossible de tester la règle du stage sans base de données ni notification	Organisation

Vous avez pu retenir d'autres problèmes : la condition sur le type de contrat, la branche `else` qui attribue silencieusement zéro jour, les noms imprécis, les chaînes écrites en dur ou le filtrage après `ToList()`. Ils sont valables si vous les justifiez par un critère et une conséquence concrète. Les trois derniers sont des problèmes locaux.

5. L'ordre de traitement. Commencez par rassembler le calcul des droits en un seul endroit : c'est ce qui supprime la cause du bug. Allégez ensuite le contrôleur en déplaçant les règles métier dans un service, puis remplacez la condition sur le type de contrat. La création des dépendances sera confiée au framework au chapitre 4.

La distinction attendue dans votre diagnostic est la suivante : corriger l'arrondi traite le **symptôme** ; rassembler la règle en un seul endroit traite la **cause**. C'est cet argument qui permettra à Inès de justifier le temps consacré au refactoring.

Erreurs fréquentes

- **Se limiter à remplacer** `Math.Ceiling` **par** `Math.Floor` : le bug disparaît, mais la règle reste dupliquée. La prochaine évolution, comme l'ajout des stages, pourra recréer le même écart.
- **Choisir soi-même la bonne règle** : sans la confirmation de Samuel, vous auriez pu « corriger » l'approbation pour l'aligner sur l'affichage, et accorder des jours que les salariés n'ont pas.
- **Décrire un problème sans critère ni conséquence** : « ce code est mal écrit » ne permet pas à Inès de prendre une décision ; « la règle est dupliquée, donc le stage devra être ajouté deux fois » le permet.
- **Modifier le code pendant le diagnostic** : vous perdez votre point de comparaison, et les tests ne vous disent plus si le comportement a changé.

Point de contrôle : votre diagnostic est rédigé et argumenté. Le code n'a pas été modifié et les tests passent toujours.