Rapport Travail I : V&V – Processus de Test

SEYDOU BA

Université du Québec à Trois-Rivières

Cours Sujets Spéciaux en informatique III (PIF 6005)

Enseignant : William Flageol

Rapport Travail I : V&V – Processus de Test

Etape 1

Pour l'étape 1, on a considéré une classe CompteBancaire ayant trois méthodes(Verser,retirer,virer) et pour le développement, on a utilisé le framework de test unitaire JUnit de Java pour tester notre classe.

❖ Discussions sur les tests développés

Pour les tests développés , on a eu à développer quelques tests unitaires pour chacune de nos méthodes implémentées dans la classe CompteBancaire.

1. Méthode verser

   Au niveau de cette méthode, on a eu à tester deux cas de tests unitaires à savoir :

   - **Cas où le versement prend en paramètre un montant positif** : on a créé un compte bancaire contenant une somme de 5000.0 (la somme est évaluée en FCFA) et ensuite on a versé dans ce compte une somme de 1000.0 d'où le versement a réussi car la somme versée testé est positive et au final on a excepté un solde totale de 6000.0 au niveau du compte Bancaire après versement

   - **Cas où le versement prend en paramètre un montant négatif** : on a créé un compte bancaire contenant une somme de 6500.0 (la somme est évaluée en FCFA) et ensuite on a versé dans ce compte une somme de -2000.0 d'où le versement a échoué car la somme versée testé est négative et au final on a excepté

qu’ au niveau du compte bancaire on a toujours le même solde qu'à l'ouverture après l'échec du versement.

2. Méthode retirer

   Au niveau de cette méthode, on a eu à tester trois cas de tests unitaires à savoir :

   - Cas où le retrait prend en paramètre un montant de retrait inférieur au solde du compte : on a créé un compte bancaire contenant une somme de 23000.0 (la somme est évaluée en FCFA) et ensuite on a retiré dans ce compte une somme de 17500.0 d'où le retrait a réussi car le montant retiré testé est inférieur au solde contenu dans le compte bancaire et au final on a excepté un solde restant de 5500.0 au niveau du compte Bancaire après retrait.

   - Cas où le retrait prend en paramètre un montant de retrait égale au solde du compte : on a créé un compte bancaire contenant une somme de 26000.0 (la somme est évaluée en FCFA) et ensuite on a retiré dans ce compte une somme de 26000.0 d'où le retrait a réussi car le montant retiré testé se trouve dans ce compte et au final on a excepté qu’ au niveau du compte Bancaire on a plus de solde car tout l’argent a été retiré.

   - Cas où le retrait prend en paramètre un montant de retrait est supérieur au solde du compte : on a créé un compte bancaire contenant une somme de

29000.0 (la somme est évaluée en FCFA) et ensuite on a retiré dans ce compte une somme de 30000.0 d'où le retrait a échoué car le montant retiré testé est supérieur au solde se trouvant dans le compte et au final on a excepté aucun changement de solde au niveau du compte, le solde reste toujours égale à 29000.0 après l'échec du retrait.

3. Méthode virer

Au niveau de cette méthode, on a eu à tester trois cas de tests unitaires à savoir :

- Cas où le virement prend en paramètre un compte bancaire dont le solde du compte est supérieur au montant à virer : on a créé deux comptes bancaires contenant respectivement une somme de 63000.0 nommé compte 1 (la somme est évaluée en FCFA) et une somme de 43000.0 nommé compte 2.Ensuite on a effectué un virement d'un montant de 17500.0 vers le compte 2 d'où le virement a réussi car le montant viré testé est inférieur au solde contenu dans le compte 1 et au final on a excepté un solde de 35000.0 au niveau du compte 1 et un solde de 71000.0 au niveau du compte 2 après le virement effectué.

- Cas où le virement prend en paramètre un compte bancaire dont le solde du compte est inférieur au montant à virer:on a créé deux comptes bancaires contenant respectivement une somme de 63000.0 nommé compte 1 (la somme est

évaluée en FCFA) et une somme de 43000.0 nommé compte 2.Ensuite on a effectué un virement d'un montant de 88000.0 vers le compte 2 d'où le virement a échoué car le montant viré testé est supérieur au solde contenu dans le compte 1 et au final on a excepté que le solde au niveau du compte 1 reste le même qu'à l'ouverture du compte a savoir 63000.0 et de même qu'au niveau du compte 2 avec un solde de 43000.0 après l'échec du virement.

- Cas où le virement prend en paramètre un compte bancaire dont le solde du compte est nul avec un montant à retirer : on a créé deux comptes bancaires contenant respectivement aucun solde nommé compte 1 (la somme est évaluée en FCFA) et une somme de 43000.0 nommé compte 2.

Ensuite on a effectué un virement d'un montant de 8000.0 vers le compte 2 d'où le virement a échoué car le solde dans le compte 1 est nul et au final on a excepté que le solde au niveau du compte 1 reste le même qu'à l'ouverture du compte à savoir 0.0 et de même qu'au niveau du compte 2 avec un solde de 43000.0 après l'échec du virement.

Rapport Travail I : V&V – Processus de Test

❖ Comparaison du code manuel et du code généré

Pour le code manuel , on a excepté aucune erreur au niveau de nos tests car ses tests ont été implémenté par soi alors qu'au niveau du code généré trois erreurs pour les tests des méthodes(Verser ,Retirer et Virer) ont été repéré car pour le code généré la création du compte prend en paramètre une valeur nulle pour le solde et qu'au niveau de notre classe , la méthode verse considère qu'un solde supérieur à 0.0 , pour la méthode retirer et virer on peut pas effectuer un virement ou un retrait avec un compte bancaire dont le solde est nul.

Pour le temps d'exécution , on note que le code manuel a pris 5 ms pour s'exécuter alors que le code généré a pris 19 ms.

Pour le code , on note que le code manuel est plus structuré que le code généré avec plus de tests unitaires a effectué.

❖ avantages et inconvénients des deux outils utilisés

Pour le code manuel , on a utilisé JUnit 5 comme outil et pour le code généré SquareTest.

- **Avantages :** Pour générer des tests unitaires, SquareTest est plus avantageux que Junit5 car il génère automatiquement les méthodes à tester et aussi crée l'instance de la classe à tester et aussi comparer les résultats actuels aux résultats attendus. On note aussi que

Rapport Travail I : V&V – Processus de Test

SquareTest est plus rapide pour effectuer des tests unitaires que JUnit 5 et aussi facile à manipuler. Pour la manipulation de nos processus , l'outil Junit 5 est plus avantageux que SquareTest car on a la main mise sur nos tests unitaires et leur implémentation.

- **Inconvénients** : on a constaté que SquareTest nous permet pas d'effectuer tous nos tests possibles et pour JUnit 5 son utilisation est un peu souffrante car on effectue tous les tests unitaires manuellement.

  Pour ce qui est du processus de nos tests , SquareTest admet un inconvénient car la plupart des tests ont échoué et pour ce qui est JUnit 5 le seul inconvénient est que le processus de test est trop long.

Rapport Travail I : V&V – Processus de Test

Etape 2

Pour cette étape , on a implémenté un exemple contenant 6 classes(IConsoleDeSortie, De, FournisseurMeteo, Heros, Ihm, Jeu) , 3 interfaces (Iconsole, IFournisseurMeteo, ILanceurDeDe) et deux classes d'énumération(Résultat , Meteo)

La classe ConsoleDeSortie implémente l'interface Iconsole : cette classe permet l'affiche de la console.

La classe Fournisseur implémente l'interface IFournisseurMeteo : qui permet de savoir le temps qu'il fait.

La classe De implémente l'interface ILanceurDeDe : qui permet de lancer un dé avec un numéro de dé aléatoire.

La classe Jeu : permet de réaliser le jeu avec une méthode Tour qui prend en paramètre le de du héro et celui du monstre tout en retournant le résultat.

La classe Ihm permet de démarrer le jeu en affichant le résultat du jeu sur la console.

Résultat est une énumération comportant Gagné et Perdu.

Meteo est une énumération comportant Soleil, Pluie et Tempête.

❖ Ordre d'intégration

Pour démarrer le jeu qui est une méthode se trouvant dans la classe Ihm, on a besoin de connaître la météo et aussi de lancer un dé.

Pour cela , on a utilisé la méthode ascendante , on a retenu l'ordre suivant :

Rapport Travail I : V&V – Processus de Test

D’abord de gérer la dépendance entre la classe Jeu , la classe FournisseurMeteo et la classe Hero pour connaître la météo et le nombre de points de vie du héros.

Ensuite , on a géré la classe Ihm qui a la dépendance avec la classe consoleDeSortie, FournisseurMétéo et De.

La démarche que nous avons suivie nous a permis de gérer la classe Ihm avec ses dépendances de gérer d’abord la classe Jeu car elle a une même dépendance que la classe Ihm c'est-à-dire la classe FournisseurMeteo et aussi pour pouvoir lancer le dé.

- Discussions sur les tests de vérification des interactions entre les classes

Pour la classe ConsoleDeSortie , on a pas effectué des tests pour cette classe car cette classe admet des méthodes qui renvoient un vide.ces méthodes permettent simplement de faire un affichage sur la console du résultat du jeu.

Pour cela , on a créé une fausse console implémentant l’interface IConsole. Dans cette classe on a utilisé la fonction StringBuilder() pour ajouter nos messages sur la console.

Pour la classe Jeu on a effectué plusieurs tests :

- **Cas où l'héro gagne le monstre avec un lancer de dé supérieur à celui du monstre** : on sait que cette classe admet un dépendance avec la classe Hero et la classe FournisseurMeteo.

Rapport Travail I : V&V – Processus de Test

D’abord on a réalisé un Mock sur la classe FournisseurMeteo c'est-à- dire en créant une fausse base de données de fournisseur de météo.Ensuite , on a créé une instance de la classe Jeu contenant cette fausse base de données.

Enfin on a effectué le lancer des dés du héros et celui du monstre en donnant un dé numérotée à 6 pour l'héro et 2 pour le monstre.

La comparaison du résultat prouve que le monstre est battu et l'héro ayant obtenu un point et ayant toujours maintenu son nombre de points de vie (15).

- **Cas où l'héro gagne le monstre avec un lancer de dé égale à celui du monstre:** D’abord on a réalisé un Mock sur la classe FournisseurMeteo c'est-à-dire en créant une fausse base de données de fournisseur de météo.Ensuite , on a créé une instance de la classe Jeu contenant cette fausse base de données.

Enfin on a effectué le lancer des dés du héros et celui du monstre en donnant un dé numérotée à 4 pour l'héro et pour le monstre.

La comparaison du résultat prouve que le monstre est battu et l'héro ayant obtenu un autre point égale à 2 cette fois-ci et ayant toujours maintenu son nombre de points de vie (15).

- **Cas où l'héro perd contre le monstre avec un lancer de dé inférieur à celui du monstre et avec du soleil comme météo:**

on a appelé la méthode QuelTempsFaitIl contenu la classe FournisseurMeteo pour obtenir la météo.Après cela, on a créé une instance de la classe Jeu contenant fournisseurMeteo qui a été mocke.Enfin on a effectué le lancer des dés du héros et celui du monstre en donnant un dé numérotée à 1 pour l'héro et 4 pour le monstre.

La comparaison du résultat prouve que le monstre a gagné et l'héro n'a pas obtenu de points cette fois-ci et a perdu des nombre de points de vie calculé en fonction du temps qu'il fait. Pour cette exemple la météo est Soleil , donc le résultat prouve que le nombre de points de vie est égale à 12 car on fera l'opération suivante : on prend le de du montre qu'on soustrait avec le de du hero et ensuite on enlève du nombre de points de vie.

- Cas où l'héro perd contre le monstre avec un lancer de dé inférieur à celui du monstre et avec de la pluie comme météo:

on a appelé la méthode QuelTempsFaitIl contenu la classe FournisseurMeteo pour obtenir la météo.Après cela, on a créé une instance de la classe Jeu contenant fournisseurMeteo qui a été mocke.Enfin on a effectué le lancer des dés du héros et celui du monstre en donnant un dé numérotée à 1 pour l'héro et 4 pour le monstre.

Rapport Travail I : V&V – Processus de Test

La comparaison du résultat prouve que le monstre a gagné et l'héro n'a pas obtenu de points cette fois-ci et a perdu des nombre de points de vie calculé en fonction du temps qu'il fait. Pour cette exemple la météo est Pluie , donc le résultat prouve que le nombre de points de vie est égale à 9 car on fera l'opération suivante : on prend le de du montre qu'on soustrait avec le de du héro et ensuite on enlève du nombre de points de vie.

Cas où l'héro perd contre le monstre avec un lancer de dé inférieur à celui du monstre et avec de la tempete comme météo:

on a appelé la méthode QuelTempsFaitIl contenu la classe FournisseurMeteo pour obtenir la météo.Après cela, on a créé une instance de la classe Jeu contenant fournisseurMeteo qui a été mocke.Enfin on a effectué le lancer des dés du héros et celui du monstre en donnant un dé numérotée à 1 pour l'héro et 4 pour le monstre.

La comparaison du résultat prouve que le monstre a gagné et l'héro n'a pas obtenu de points cette fois-ci et a perdu des nombre de points de vie calculé en fonction du temps qu'il fait. Pour cette exemple la météo est Tempete , donc le résultat prouve que le nombre de points de vie est égale à 3 car on fera l'opération

suivante : on prend le de du montre qu'on soustrait avec le de du héro et ensuite on enlève du nombre de points de vie.

Pour la classe Ihm on a effectué plusieurs tests :

- **jeuGagne** : on sait que cette classe admet un dépendance avec la classe consoleDeSortie, la classe De et la classe FournisseurMeteo.

  D'abord on a réalisé un Mock sur la classe FournisseurMeteo et la classe De c'est-à- dire en créant une fausse base de données pour la classe fournisseur de météo et la classe De.Ensuite , on a appelé la méthode QuelTempsFaitIl contenu la classe FournisseurMeteo pour obtenir la météo et on a appelé la méthode Lance contenu la classe De pour effectuer un ensemble de lancers.

  Pour que notre exemple nous renvoie une réponse gagnante , on a effectué 17 lancers sur lesquels, l'héro gagne 9 fois et perd 8 fois pour pouvoir sortir du jeu car on avait une boucle infini qui effectue toujours lorsque le nombre de points de vie du héro est toujours supérieur à 0.Enfin on a démarré le jeu et à la sortie on a pu afficher la console.

- **jeuPerdu** : on sait que cette classe admet un dépendance avec la classe consoleDeSortie, la classe De et la classe FournisseurMeteo.

  D'abord on a réalisé un Mock sur la classe FournisseurMeteo et la classe De c'est-à- dire en créant une fausse base de données pour la classe fournisseur de

Rapport Travail I : V&V – Processus de Test

météo et la classe De.Ensuite , on a appelé la méthode QuelTempsFaitIl contenu la classe FournisseurMeteo pour obtenir la météo et on a appelé la méthode Lance contenu la classe De pour effectuer un ensemble de lancers.

Pour que notre exemple nous renvoie une réponse perdue , on a effectué 17 lancers sur lesquels, l'héro gagne 7 fois et perd 10 fois.Enfin on a démarré le jeu et à la sortie on a pu afficher la console.

Rapport Travail I : V&V – Processus de Test

Etape 3

- Modification mineure

Pour les modifications mineures , on a fait un retrait d'attribut , retrait de méthode , ajout d'une méthode.

Pour le retrait d'attribut : on a fait le retrait de l'attribut heros dans la classe Jeu.

Après le retrait de l'attribut on a constaté que les tests jeu et ihm ne marchaient plus car l'attribut a un impact sur ses tests.

Pour le retrait de la méthode : on a fait le retrait de la methode GagneLecombat dans la classe Jeu. Après cela on a constaté que nos tests précédents ne marchaient plus car cette méthode a un impact fort sur ces tests.

En résumé, lorsqu'on exécute des tests après ses modifications on constate une erreur nous invoquant des erreurs dans la classe Jeu.

Ajout d'une méthode : on a ajouté une methode EstTermine dans la classe Jeu. Donc cela nous amène à effectuer un petit changement dans la classe Ihm pour que la méthode demarre en appelant cette méthode.

Après cela , on a constaté qu'aucune erreur n'est évoquée dans nos tests. Nos tests sont bien exécutés sans aucune erreur.

Rapport Travail I : V&V – Processus de Test

NB: on a fait des commentaires sur nos modifications mineures sur le code pour l'étape 2 où vous verrez les modifications apportées.

on a constaté aussi pour le test sur la classe Jeu , que nos classes sont couvertes à 100% et que 75 % de nos méthodes dans cette classe ont été couverts.Donc cela montre que toutes nos méthodes n'ont pas été bien couvertes.

Pour notre classe Ihm , toutes nos méthodes et classes ont été bien couvertes à 100%.

Donc cela montre que toutes nos méthodes et classes ont été bien utilisées.

- Modification majeure

Pour les modifications majeures , on a ajouté 3 nouveaux classes(FabriqueMonstres, Monstre, MonstreCostaud) et 2 nouvelles interfaces(IFabriqueMonstres,IMonstre).

La classe FabriqueMonstres implémente l'interface IFabriqueMonstres.

La classe Monstre et MonstreCostaud implémentent l'interface IMonstre.

Pour les modifications majeures, on a pu étendre la classe Jeu et la classe Ihm.

La classe Jeu et la classe Ihm ont chacune une dépendance dans la classe FabriqueMonstres. L'etend de ces classes ont eu des influences sur nos tests précédents car on a eu à refaire quelques modifications sur nos tests précédents pour pouvoir avoir la dépendance entre ces classes.

Rapport Travail I : V&V – Processus de Test

Les modifications majeures ont bien eu un impact grandiose sur nos tests précédents car on pu revoir tous nos tests pour pouvoir refaire bien les tests car on a eu de nouvelles dépendances.

Les modifications mineures n’ont pas eu une grande impact sur nos tests initiaux car les changements ont été mineures alors que les modifications majeures ont eu un grand impact sur nos tests initiaux car on a la tâche d'effectuer nos tests car il y avait de nouvelles dépendances à mettre car nos classes ont eu de nouvelles dépendances.
Cela implique que les modifications mineures n'ont touché que sur 20% de nos codes alors que les modifications majeures ont touché 80% de nos codes.

Rapport Travail I : V&V – Processus de Test

Références

oo05exerc1-cresolu-java-xxx.pdf (unisciel.fr)

Entrainez-vous à utiliser les tests pour détecter un problème dans une application - Testez votre application C# - OpenClassrooms