Projet d'analyse et de programmation orientée objet : Gestion de vols en planeur

Vous êtes chargés d'effectuer l'analyse et l'implémentation d'un système permettant la gestion de vols en planeur.

Enoncé du problème

Le gestionnaire d'un club de vol à voile vous demande de lui écrire une application de bureau. Cette application lui permettra de gérer les vols réalisés sur les planeurs par les pilotes du club. Elle lui permettra d'encoder les informations des pilotes et les vols qu'ils ont réalisés. Les vols sont payants. Pour pouvoir voler, un pilote doit avoir alimenté au préalable son compte pilote. Lorsqu'un pilote a effectué un virement sur le compte bancaire du club, le gestionnaire crédite le compte du pilote en utilisant l'application.
L'application ne sera installée que sur le pc du gestionnaire et ne sera pas accessible via le net. Il sera le seul à l'utiliser. Les données seront sauvées dans une base de données relationnelle présente sur son ordinateur.

a) Votre application devra permettre au gestionnaire du club d'enregistrer de nouveaux pilotes. Pour chacun d'eux, il fournira un email, un nom, un prénom, une adresse, un numéro de gsm, le solde de son compte pilote. Ces exigences fonctionnelles (« requirements ») seront à implémenter dans l'itération 2.

b) Votre application devra permettre au gestionnaire du club :
- de lister les pilotes triés sur leur nom et prénom ;
- de lister les pilotes dont le solde du compte est en négatif. Cette liste sera triée sur les soldes du plus négatif au moins négatif.
- de sélectionner un pilote dans la liste et modifier ses données.
Ces listes reprendront toutes les données des pilotes à l'exception des vols.
Ces requirements seront à implémenter dans l'itération 3.

c) Après une journée de vols, le gestionnaire du club reçoit la « planche de vol » qui détaille les données des vols réalisés ce jour-là. Chaque ligne de la planche de vol présente les informations suivantes : le nom et prénom du pilote ayant réalisé un vol, la durée du vol, le type de planeur utilisé. L'application devra permettre au gestionnaire du club d'enregistrer toutes ces données. A chaque enregistrement d'un vol, le solde du compte du pilote devra être automatiquement mis-à-jour. Le coût du vol et le nouveau solde du compte du pilote seront affichés. Le coût d'un vol se compose d'un coût fixe de 25 € (remorquage du planeur) et d'un coût variable qui dépend de la durée du vol et du type de planeur. Voici les tarifs : planeurs « bois & toile » : 17 €/h, planeurs « plastique » : 27€/h, planeurs « biplace » : 30 €/h.
Votre application devra permettre au gestionnaire du club :

- d'enregistrer des vols ;
- de lister les vols d'une journée dont on précise la date. Cette liste sera triée sur les durées de vol, du plus long au moins long.

Toutes les données de la planche de vol devront être mémorisées en DB.
Ces requirements seront à implémenter dans l'itération 4.

d) Afin de limiter votre travail :
- L'application ne devra pas permettre la suppression de pilotes ;
- Les types de planeur et leur tarif horaire, ainsi que le coût d'un remorquage seront mémorisés dans une table en DB. Un script sql exécuté sur le serveur permettra d'y insérer les données. L'application ne devra pas permettre de les modifier, ni les supprimer. Elle ne permettra pas non plus d'en ajouter de nouvelles.
- L'application ne devra pas permettre de supprimer, ni modifier l'enregistrement d'un vol.

Cet énoncé sera complété progressivement au cours des séances d'exercices suite aux questions qui seront posées. Le document « addendum au cahier des charges » disponible sur l'extranet sera complété. Vous penserez à le consulter régulièrement. Avant de poser une question au professeur, consultez ce document.

Les exigences du projet

Point de vue analyse et développement

- Votre projet sera mené de manière **itérative** et **évolutive**.
- Votre analyse débutera par une description des exigences de l'utilisateur (les « requirements ») ;
- Ensuite, vous décrirez les différents acteurs et leurs responsabilités ;
- Vous dessinerez un diagramme de use case ;
- Ensuite, vous établirez un diagramme de classes (uniquement les classes du domaine) ;
- Par itération, vous approfondirez les cas d'utilisation en fournissant une description textuelle des uses-case.
- Par itération, vous détaillerez par écrit, les tests fonctionnels.
- Enfin, vous implémenterez et testerez votre application également de façon itérative. La couche de persistance ne sera pas votre priorité. Vous commencerez par des « mock objects » pour pouvoir tester votre application rapidement.

Point de vue technique

- Ce programme doit être implémenté en **Java** en utilisant l'éditeur **Eclipse** ;

- La persistance se fera en **base de données** que vous implémenterez sous **PostgreSQL**. Cette base de données doit être simple, càd sans aucun trigger, ni procédure ;
- Vous utiliserez un serveur de gestion de version pour le développement de votre projet en équipe. **BitBucket** sera utilisé. Créez-y un compte et ajoutez les membres de votre équipe, ainsi que le professeur (olivole@gmail.com). Vous mettrez en œuvre la méthodologie « **GitFlow** » vue en première année.
- L'**IHM** du programme doit être simple, user friendly, ergonomique, … (en **Swing**). Les fonctionnalités seront accessibles via une barre de menus et une « tool bar ».
- Votre conception sera le reflet des méthodes rigoureuses apprises durant cette année. Votre découpe visera l'indépendance des couches ;
- Un accent particulier sera mis sur l'aspect des tests. Vous automatiserez vos tests unitaires et d'intégration en utilisant **TestNG**. Pensez à créer des suites de tests décrites dans des fichiers **xml**.

Echéancier

En théorie …

Ce projet est développé de manière itérative et évolutive. Nous le découpons donc en une série d'étapes.

Chaque itération se compose des phases suivantes : analyse, développements, tests.

En pratique …

Notre projet se divise en 4 itérations représentant chacune un mini projet. A la fin de chaque itération (sauf itération 1), le projet est exécutable et les tests peuvent être exécutés. Chacune de ces itérations est détaillée dans la suite du document. Sauf pour l'itération 1, une itération se déroule comme suit :

1. Correction des éléments de l'itération précédente ;
2. Implémentation des éléments d'analyse corrigés de l'itération précédente ;
3. Tests unitaires, d'intégration et fonctionnels[1];
4. Analyse des éléments à implémenter dans l'itération suivante (sauf pour la dernière itération).

[1] Les exigences concernant les tests fonctionnels sont décrites à la page 10.

Concrètement :

- Remise de l'itération 1 : le 28/3 avant la pause ;
- Remise de l'itération 2 : le 18/4 avant la pause ;
- Remise de l'itération 3 : le 9/5 avant la pause ;
- Remise de l'itération 4 : le 16/5 (dossier final).

Itération 1 : analyse globale

Cette itération a pour premier objectif de clarifier l'énoncé. Elle permettra ensuite de détailler les différentes fonctionnalités attendues par le client. Elle s'attardera ensuite à l'analyse des fonctionnalités à implémenter durant l'itération 2.

Voici les étapes de cette itération :

1. Procéder à une analyse globale :
   - Choix d'un nom pour la nouvelle application ;
   - Définition du système (phrases courtes permettant de définir clairement l'application à développer) ;
   - Glossaire des terminologies ;
   - Détails des acteurs ainsi que leurs responsabilités respectives ;
2. Décrire et énumérer les exigences du client (les « requirements ») ;
3. Décrire et énumérer les cas d'utilisation par itération ;
4. Élaborer le diagramme des cas d'utilisation ;
5. Distribuer les rôles de chaque membre du groupe pour les itérations futures (en veillant bien à répartir le travail équitablement). Nous insistons pour que chaque membre du groupe travaille sur des couches différentes de votre application à chaque itération ;
6. Présenter le modèle complet (scénario principal inclus ainsi que les alternatives) des cas d'utilisation relatifs à l'itération suivante ;
7. Fournir un diagramme de classe de niveau conceptuel des classes du domaine ne reprenant que les attributs, pas les méthodes, ni les constructeurs ;
8. Fournir un diagramme de composants décrivant l'architecture du système. Il doit reprendre les packages de l'application.
9. Présenter un schéma relationnel (modèle logique) de votre base de données.

Itération 2

Au terme de cette itération, le gestionnaire du club doit pouvoir enregistrer de nouveaux pilotes. Pour chacun d'eux, il fournira un email, un nom, un prénom, une adresse, un numéro de gsm, le solde de son compte pilote.

Voici les étapes de cette itération :

Analyse :

1. **Corriger le rapport ;**
2. Compléter, si nécessaire, vos diagrammes de classe et de composants décrivant l'architecture du système ;
3. Ecrire les tests fonctionnels de cette itération[2] ;
4. Présenter le modèle complet (scénario principal inclus ainsi que toutes les alternatives) des cas d'utilisation relatifs à l'itération suivante ;

Implémentation:

5. Implémenter le cas d'utilisation cité ci-dessus (en début de page).
6. Pour la couche de persistance, commencez par un ou plusieurs « mock objects » qui renvoie(nt) des données intéressantes. Quand tous les tests passeront, vous développerez l'implémentation des DAOs qui interagissent avec la DB.
7. Tester votre implémentation (tests unitaires, d'intégration et fonctionnels).

[2] Les exigences concernant les tests fonctionnels sont décrites à la page 10.

Itération 3

Au terme de cette itération, le gestionnaire du club doit pouvoir :
- lister les pilotes triés sur leur nom et prénom ;
- lister les pilotes dont le solde du compte est en négatif. Cette liste sera triée sur les soldes du plus négatif au moins négatif.
- sélectionner un pilote dans la liste et modifier ses données.

Ces listes reprendront toutes les données des pilotes à l'exception des vols.

Voici les étapes de cette itération :

Analyse :

1. **Corriger le rapport ;**
2. Compléter, si nécessaire, vos diagrammes de classe et de composant décrivant l'architecture du système ;
3. Ecrire les tests fonctionnels de cette itération[3] ;
4. Présenter le modèle complet (scénario principal inclus ainsi que toutes les alternatives) des cas d'utilisation relatifs à l'itération suivante.

Implémentation :

5. Implémenter les cas d'utilisation cités ci-dessus (en début de page).
6. Pour la couche de persistance, commencez par un ou plusieurs « mock objects » qui renvoie(nt) des données intéressantes. Quand tous les tests passeront, vous développerez l'implémentation des DAOs qui interagissent avec la DB.
7. Tester votre implémentation (tests unitaires, d'intégration et fonctionnels).

[3] Les exigences concernant les tests fonctionnels sont décrites à la page 10

Itération 4

Au terme de cette itération, le gestionnaire du club doit pouvoir :
- enregistrer des vols ;
- lister les vols d'une journée dont on précise la date. Cette liste sera triée sur les durées de vol, du plus long au moins long.

Toutes les données de la planche de vol devront être mémorisées en DB.

Voici les étapes de cette itération :

Analyse :

1. **Corriger le rapport et le code ;**
2. Compléter, si nécessaire, vos diagrammes de classe et de composant décrivant l'architecture du système ;
3. Ecrire les tests fonctionnels de cette itération ;
4. Décrivez les tests et suites de tests réalisés et leurs résultats. (Voir plus loin)
5. Ajouter une introduction et une conclusion dans votre rapport.
6. Détailler les problèmes rencontrés au point de vue organisationnel et technique et leur(s) solution(s)
7. Présenter l'état de votre application (niveau technique) :
   - Qu'est-ce qui marche ?
   - Qu'est-ce qui ne fonctionne pas ? Pourquoi ?
   - Quelles sont les éventuelles améliorations à apporter à votre projet (point de vue design de l'application) ?
   - Donner votre avis sur le projet (niveau organisationnel).
   - Quels sont vos impressions sur ce projet et vos regrets ?
   - Qu'est-ce qui a été le plus difficile à gérer dans ce projet ?
   - Qu'est-ce qui pourrait améliorer la qualité de ce travail ?
   - Évaluez chacun le temps de travail que vous avez investi dans ce projet.
   - Remarques, conseils ...

Implémentation:

1. Implémenter les cas d'utilisation cités ci-avant (en début de page).
2. Tester votre implémentation (tests unitaires, d'intégration et fonctionnels).

Les exigences du rapport

En ce qui concerne le rapport, chaque itération débouchera sur la livraison d'une série de documents (en fonction de ce qui est demandé dans l'itération) :

- Le document de spécification reprenant tout ce qui décrit ce que le client attend de l'application :
  - Une introduction expliquant la demande du client
  - Glossaire
  - Détail des acteurs ainsi que leurs responsabilités respectives ;
  - Liste numérotée des exigences fonctionnelles (« requirements »)
  - Liste numérotée des cas d'utilisation (+référence(s) aux requirements à l'origine du uses case)
  - Diagramme de Uses Cases
  - Description des Uses Cases (+référence(s) aux requirements à l'origine du uses case).
  - Diagramme de classes « conceptuel » de la couche métier
- Le document d'architecture reprenant tous les diagrammes relatifs au design de l'application :
  - Diagramme de composants décrivant l'architecture (reprenant les packages)
  - Schéma relationnel (modèle logique) de votre base de données.
- Le document de vérification et de validation de votre application :

  Décrivez les tests et suites de tests réalisés et leurs résultats. (Voir plus loin)
- A chaque itération, pour chacun de ces documents vous nous remettrez la nouvelle version contenant les corrections de l'itération précédente ainsi que les ajouts dus à la nouvelle itération.

Pour chaque élément d'analyse (diagramme ou texte) ou de programmation (classe), un **auteur** sera **clairement** spécifié. Un validateur/vérificateur peut éventuellement être mentionné. Il s'agit effectivement d'un travail de groupe **et** individuel (voir section Evaluation : travail en groupe – travail individuel).

Le rapport est à remettre dans une **farde à rabats**. Personnalisez cette farde afin que l'on puisse l'identifier aisément comme étant celle de votre groupe !

Vous veillerez toujours à y inclure :

- Une pagination des feuilles (numéro de page/nombre total de pages)

- Une table des matières (référençant les numéros de page) ;
- Les références aux auteurs des diagrammes ou texte.
- Les différentes itérations séparées de façon distincte (intercalaire par exemple)

Remarque : votre rapport peut contenir d'autres diagrammes d'analyse (que ceux exigés) et que vous jugez intéressants.

Les exigences du code Java

Vos classes Java doivent respecter les principes de programmation appris à l'école. N'oubliez pas de nettoyer vos classes : pas trop de commentaires, pas d'affichage de débuggging, ...

Chaque classe commencera par les tags @author suivi du nom de l'auteur de cette classe et @version.

Eventuellement, si un validateur a vérifié le code, il devra alors être mentionné explicitement en commentaire (//validateur = __).

Les exigences des tests

Les tests se feront à 3 niveaux :

- Tests unitaires et d'intégration ;
- Tests fonctionnels

Les tests unitaires et d'intégration devront être automatisés à l'aide de programmes de tests de type **TestNG**.

Vous pouvez utiliser l'approche Test Driven Development. Càd que vous commencerez par écrire quelques cas de tests simples **avant d'écrire le code de vos méthodes** ! Ces cas de tests vous permettront d'avoir une idée plus claire de ce que votre méthode doit faire. Considérez-les comme la spécification de vos méthodes.

Vous ne devez pas tester toutes les méthodes de votre code. Vous ne devez pas faire de tests exhaustifs, mais avoir quelques cas intéressants (cas nominal + quelques cas particuliers).

Vu l'ampleur de votre projet, il n'est pas possible de tester toutes les méthodes de toutes les classes de votre projet en un temps raisonnable.

Pour chaque niveau de tests, identifiez les éléments cruciaux et les plus susceptibles de contenir des bugs.

Pour chaque niveau de test, vous devez identifier ce qui est testé.

Vous devez donc nous livrer les éléments suivants :

Pour les tests unitaires et d'intégration :

La liste des classes et dans celles-ci les méthodes testées.

Le code imprimé des classes de tests TestNG avec à l'intérieur de celles-ci des commentaires explicitant les cas de tests.

Le code imprimé des « mock objects » utilisés. (Le cas échéant)

Pour les tests fonctionnels

Vous devez décrire les tests fonctionnels sous forme d'un tableau. Chaque test sera identifié. Il précisera le(s) identifiant(s) des uses cases concerné(s) par ce test. Vous décrirez le scénario du test. Il faut préciser par exemple quels boutons pousser, ce qu'il faut introduire comme données, … et préciser les résultats attendus.

Les tests doivent être des tests « positifs », mais aussi « négatifs ». Vous devez tester aussi les alternatives précisées dans les descriptions textuelles des use cases.

Ces tests doivent être très détaillés. Une personne n'ayant participé ni à l'analyse, ni à l'implémentation doit être capable de les réaliser en lisant simplement votre document. Prévoyez une colonne vide intitulée « Résultat obtenu ». Le testeur la remplira au fur et à mesure des tests qu'il réalise. Ce document vous servira ensuite pour débugger votre application.

Evaluation : travail en groupe - travail individuel

Vous réalisez ce projet en groupe. Ce qui ne signifie pas un cerveau (une plume) pour trois mais des partenaires qui **collaborent**. C'est pour cela que vous diviserez le travail au sein du groupe pour chaque itération en veillant à la cohérence et à la qualité et quantité des productions individuelles.

A cette fin, chaque partie d'analyse (diagramme ou texte) sera conçue par une personne et vérifiée/validée éventuellement par une autre. Chaque classe débutera par la spécification de son auteur. La validation prendra le plus souvent la forme d'une relecture consciencieuse. Pour le code, la validation prendra la forme de tests.

Attention, vous serez évalué en fonction de vos présences. Càd si vous êtes présent à 80% des séances, vous aurez une note rabotée à 80%. Évidemment, toute absence valable et non répétée sera prise en compte. Nous insistons sur ce point, car l'évaluation dépend notamment du travail que vous fournirez effectivement en séance.

Bon travail.