INF2015 – Développement de logiciels dans un environnement Agile

Examen intra – 20 février 2014 17:30 à 20:30

Nom, prénom :

Code permanent :

Répondez directement sur le questionnaire.

Question #1 – 5%

Quelle influence peut avoir le typage dynamique sur la maintenabilité d'un logiciel?

.....

.....

.....

Question #2 – 5%

Quel est le lien entre la lisibilité du code source et la maintenabilité d'un logiciel?

.....

.....

.....

Question #3 – 5%

Comment l'agilité permet-elle de renforcir la maturité d'une équipe de développeurs?

.....

.....

.....

Question #4 – 5%

Pourquoi est-il déconseillé de mettre un document Word, un document PDF ou tout autre fichier en format binaire dans un gestionnaire de sources?

.....

.....

.....

Question #5 – 10%

Qu'est-ce que l'agilité dans le développement de logiciels?

.....

.....

.....

.....

.....

.....

.....

.....

Question #6 – 10%

Considérant l'extrait de code Java suivant :

```
public static String snakeCaseIdentifierToCamelCase(String identifier) {
  char[] workString = identifier.toLowerCase().toCharArray();

  boolean firstCharacterMet = false;
  for (int charIndex = 0; charIndex < workString.length; charIndex++) {
    if (workString[charIndex] == '_' && firstCharacterMet) {
      if (charIndex < workString.length - 1) {
        workString[charIndex+1]=Character.toUpperCase(workString[charIndex+1]);
      }
    } else {
      firstCharacterMet = true;
    }
  }

  return new String(workString).replace("_", "");
}
```

Décrivez les différentes étapes de refactoring que vous feriez pour rendre cette méthode plus propre. Note : l'indentation a été réduite à 2 espaces pour l'examen, elle est à 4 espaces en réalité.

.....

.....

.....

.....

.....

.....

.....

.....

Questions à choix multiples – 60% (2% chacune)

Indiquez les réponses dans ce tableau. Un seul choix par question.

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26.
27.
28.
29.
30.

Question 1

À partir de combien de membres dans une équipe devient-il nécessaire d'utiliser un gestionnaire de sources?

a) À partir de 3 membres, pour favoriser la révision de code.
b) À partir de 2 membres, pour favoriser la révision de code.
c) Peu importe la taille de l'équipe, on devrait toujours coder sous gestion de sources.
d) Ça ne dépend pas de la taille de l'équipe, mais plutôt de la coordination des membres.

Question 2

Dans un gestionnaire de sources, qu'est-ce qu'un dépôt centralisé?

a) Un dépôt sur la machine du développeur, que les autres développeurs peuvent utiliser.
b) Un dépôt qu'on accède au travers d'un réseau et que tous les membres de l'équipe utilisent.
c) Un dépôt privé pour un développeur centralisé.
d) Un dépôt unique contenant tous les projets de l'entreprise.
e) Aucune de ces réponses.

Question 3

Quel est le principal avantage d'avoir un dépôt local avec Git?

a) Permettre de renommer un fichier et de conserver son historique.
b) Pouvoir faire des modifications dans les fichiers sources.
c) Pouvoir utiliser une interface web comme github.com.
d) Pouvoir bénéficier des avantages du gestionnaire de sources, même sans connectivité au réseau.
e) Aucune de ces réponses.

Question 4

Quel type de document **ne** devrait-on **pas** retrouver dans un gestionnaire de sources?

a) Un document d'analyse.
b) Un prototype jetable.
c) Un script de tests automatisés.
d) La documentation utilisateur.
e) Aucune de ces réponses.

Question 5

Qu'est-ce que le typage dynamique?

a) Le type doit être connu à la compilation.
b) Le type doit être respecté à l'exécution.
c) Le type peut changer à la compilation mais pas à l'exécution.
d) Le type est connu seulement à l'exécution et peut changer durant l'exécution.
e) Le type n'est jamais connu.
f) Le langage n'utilise qu'un seul type.

Question 6

Parmi ces langages de programmation, lequel est le plus expressif?

a) Groovy
b) Java
c) C++
d) C
e) Langage d'assemblage

Question 7

Quel serait l'avantage d'utiliser le langage de programmation Groovy?

a) Beaucoup de fonctionnalités en peu de lignes de code.
b) Excellente compatibilité avec Java et la JVM.
c) Meilleure productivité des programmeurs.
d) Meilleure lisibilité du code source.
e) Toutes ces réponses.

Question 8

Dans quel contexte le modèle de développement en cascade est-il un **bon** choix?

a) Lorsque les spécifications sont inconnues.
b) Lorsque les spécifications changent régulièrement.
c) Lorsque que le projet est risqué.
d) Lorsque les estimations doivent absolument être respectées.
e) Lorsque les spécifications sont connues et ne changeront pas.
f) Aucune de ces réponses.

Question 9

Comment le modèle de développement itératif permet-il de gérer le risque dans un projet de développement de logiciel?

a) En analysant tous les risques dans la première itération.
b) En développant les fonctionnalités les plus risquées dans les premières itérations.
c) En développant les fonctionnalités les plus risquées dans les dernières itérations.
d) En ajoutant des fonctionnalités lors de chaque itération.
e) En présentant régulièrement le logiciel au client.
f) Aucune de ces réponses.

Question 10

Selon le modèle de développement en cascade, quand devons-nous effectuer des tests sur le logiciel produit?

a) À chaque itération.
b) Après la réalisation.
c) Pendant la maintenance.
d) Après l'analyse organique.
e) Jamais.
f) Après l'analyse des besoins.
g) Après le déploiement.

Question 11

Selon le modèle de développement itératif, quand pourrait-on déployer chez le client le logiciel produit?

a) À la fin du projet.
b) Lorsque toutes les fonctionnalités élémentaires sont terminées.
c) Lorsque le budget a été complètement consommé.
d) Après chaque itération.
e) Toutes ces réponses.

Question 12

Comment le modèle de développement itératif permet-il d'obtenir une meilleure satisfaction du client?

a) Après chaque itération, on déploie le logiciel chez le client pour le tester.
b) Après chaque itération, on présente le logiciel au client pour recueillir son feedback.
c) Après chaque itération, on demande de nouvelles spécifications.
d) Avant chaque itération, on demande au client de définir ses attentes.
e) Aucune de ces réponses.

Question 13

Qu'est-ce qu'un écosystème de développement?

a) L'ensemble des développeurs de l'équipe.
b) L'ensemble des développeurs de l'entreprise.
c) L'ensemble des règles de codification.
d) L'ensemble des outils des développeurs.
e) L'environnement physique des développeurs.

Question 14

Parmi les éléments suivants, lequel **n'est pas** un élément de style dans le code source?

a) La position des accolades.
b) La nomenclature.
c) Le format des commentaires.
d) L'ordre de déclaration des méthodes et des variables.
e) La position des caractères d'espacement.
f) Aucune de ces réponses.

Question 15

Pourquoi est-il important d'avoir un style uniforme dans le code source?

a) Améliorer la performance du logiciel.
b) Améliorer la vitesse d'écriture du code.
c) Améliorer la lisibilité du code source.
d) Améliorer le sentiment d'appartenance des développeurs dans l'équipe.
e) Toutes ces réponses.

Question 16

En Java, on recommande d'utiliser la casse :

a) PascalCase pour les noms de classes.
b) camelCase pour les noms de variables.
c) UPPER_SNAKE_CASE pour les noms de constantes.
d) camelCase pour les noms de méthodes.
e) Toutes ces réponses.

Question 17

Selon le chapitre 2 de «Coder proprement», laquelle des affirmations suivantes **n'est pas** recommandée lorsqu'on doit choisir un nom significatif :

a) Choisir des noms prononçables.
b) Choisir des noms compatibles avec une recherche.
c) Choisir une codification pertinente et succincte, comme la notation hongroise.
d) Choisir des noms de méthodes qui commencent par un verbe.
e) Aucune de ces réponses.

Question 18

Selon le chapitre 3 de «Coder proprement», laquelle des affirmations suivantes **n'est pas** recommandée concernant le nombre d'arguments à passer à une fonction :

a) Une fonction sans argument, c'est l'idéal.
b) Les fonctions à trois arguments doivent être évitées autant que possible.
c) Une fonction ne devrait jamais avoir plus de trois arguments.
d) Plus il y a d'arguments, plus la fonction est difficile à tester.
e) Le nombre d'argument peut être élevé, tant que chaque argument possède un nom significatif et une documentation adéquate.
f) Aucune de ces réponses.

Question 19

Selon le chapitre 3 de «Coder proprement», quelle affirmation représente l'opinion de l'auteur au sujet de l'instruction *return* :

a) Une fonction ne devrait avoir qu'une seule instruction *return*.
b) Une fonction peut avoir plusieurs instructions *return*, en autant que chaque instruction *return* possède un commentaire explicatif.
c) Si la fonction est courte, avoir plusieurs instructions *return* n'est pas un problème.
d) Si la fonction est longue, c'est préférable d'avoir plusieurs instructions *return*.
e) Aucune de ces réponses.

Question 20

Selon le chapitre 4 de «Coder proprement», quelle approche devrions-nous adopter par rapport aux commentaires dans le code source?

a) Ne jamais écrire des commentaires dans le code.
b) Toujours écrire des commentaires dans le code.
c) Si le code nécessite un commentaire, c'est souvent un signe qu'on devrait réécrire le code.
d) Si le code nécessite un commentaire, on rédige le commentaire pour rendre le code plus clair.
e) Aucune de ces réponses.

Question 21

Selon le chapitre 4 de «Coder proprement», que devons-nous faire si nous rencontrons du code en commentaire?

a) On ne doit pas y toucher, il a été mis en commentaire pour une raison valable.
b) On doit le faire évoluer comme le code qui l'entoure. Le jour où il sera décommenté, il fonctionnera toujours.
c) On doit retrouver le développeur qui l'a mis en commentaire et lui demander pourquoi ce code a été commenté.
d) On doit retirer ce code, nous pourrons toujours le retrouver dans le gestionnaire de sources aux besoins.
e) Aucune de ces réponses.

Question 22

Selon le chapitre 7 de «Coder proprement», comment devons-nous gérer les erreurs dans un programme Java de façon à conserver le code propre?

a) Avec des codes de retour clairs et bien documentés.
b) Retourner null en cas d'erreur.
c) Avec des exceptions.
d) On utilise les trois techniques ci-dessus dépendamment du contexte de la fonction.
e) Aucune de ces réponses.

Question 23

Qu'est-ce que le refactoring?

a) Modifier du code existant, sans modifier la fonctionnalité.
b) Réécrire du code pour le rendre plus lisible.
c) Retravailler le code pour en améliorer la maintenabilité.
d) Refaire la conception dans le but de la simplifier.
e) Toutes ces réponses.

Question 24

Pourquoi voudrions-nous faire du refactoring?

a) Pour rendre le code plus facile à modifier.
b) Pour mieux gérer le risque.
c) Pour que les développeurs aient une bonne confiance en eux.
d) Parce que l'équipe d'assurance-qualité l'exige.
e) Toutes ces réponses.

Question 25

À quel moment devrions-nous faire du refactoring?

a) En fin de projet, s'il reste du temps.
b) Après le déploiement en production.
c) Après chaque itération.
d) Un peu chaque jour.
e) Toutes ces réponses.

Question 26

Que devons-nous faire si nous découvrons un bogue dans le code actuel pendant que nous effectuons une activité de refactoring?

a) Corriger le bogue en même temps que le refactoring.
b) Terminer le refactoring et corriger le bogue ensuite.
c) Laisser le bogue en place et le documenter pour une correction ultérieure.
d) Annuler l'activité de refactoring pour corriger le bogue car la correction aura peut-être un impact significatif sur le code, ce qui rendrait notre refactoring inutile.
e) Aucune de ces réponses.

Question 27

Vous faites parti d'une équipe de trois développeurs et vous avez toujours suivi le même style instinctivement. Un nouveau développeur arrive dans l'équipe et vous réalisez que son style est différent. Comment allez-vous adresser ce problème?

a) Ce n'est pas un problème. Le nouveau développeur va sûrement finir par faire comme les autres.
b) L'équipe demande au nouveau développeur de suivre le style déjà en place dans le code.
c) L'équipe en profite pour se rencontrer et élaborer ensemble une norme de codification.
d) L'équipe rejette le nouveau développeur.
e) Toutes ces réponses.

Question 28

Comment peut-on faire du refactoring de façon sécuritaire si nous **n'avons pas** de tests unitaires dans le projet?

a) Écrire des tests unitaires sur les parties touchées par le refactoring.
b) Faire des petites modifications peu risquées à l'aide des outils de refactoring automatique de NetBeans.
c) Faire plusieurs petits changements et tester manuellement l'application après chaque changement.
d) Travailler minutieusement et faire vérifier son travail par un collègue expérimenté.
e) Toutes ces réponses.

Question 29

Parmi les choix suivants, lequel **n'est pas** une partie prenante lors d'un développement de logiciel?

a) Les développeurs.
b) Les utilisateurs.
c) Les commanditaires.
d) L'équipe d'assurance-qualité.
e) Aucune de ces réponses.

Question 30

L'utilisation d'un gestionnaire de sources est un prérequis à l'agilité. Pourquoi?

a) Parce que ça permet aux développeurs d'annuler rapidement une modification erronée et de remettre le dépôt dans un état stable.
b) Parce que toutes les équipes de développeurs utilisent ce genre d'outil.
c) Parce que le refactoring nécessite l'utilisation d'un gestionnaire de sources.
d) Parce que le développement itératif nécessite l'utilisation d'un gestionnaire de sources.
e) Toutes ces réponses.