Université Abou Beckr Belkaid
Master 2 Génie Logiciel
Année Universitaire 2017/2018

Examen Final

—

Validation et vérification logiciel

—

Nom	
Prénom	
Numéro Etudiant	

Remarques :

- Les documents ne sont pas autorisés ainsi que les appareils électroniques (PC, Tablette, téléphone, etc).
- La première partie A est sous forme de QCM à choix multiples.
- La seconde partie B est sous forme de questions libres à répondre sur la double feuille.
- Dans la partie QCM, dans la même question une réponse fausse annule une réponse juste. Donc ne répondez que si vous êtes sure.

Partie A : Cochez la (ou les) bonne(s) réponse(s).

Questions	Réponses
1. La technique de partitionnement en classes d'équivalence consiste à :	☐ trouver des domaines sur lesquels le logiciel se comporte de façon homogène.
	☐ trouver des logiciels équivalant au logiciel à tester et à réutiliser les jeux de tests qui ont été fait pour ce logiciel.
	☐ diviser l'équipe de test en groupe de tailles et d'expériences équivalentes.
	☐ aucune réponse juste.
2. La technique des tests aux limites consiste à :	☐ Pousser aux limites les équipes de tests en les mettant fortement sous pression.
	☐ Essayer d'atteindre la limite des fonctions de maturité du logiciel.
	☐ Faire fonctionner le logiciel aux limites de ses spécifications.
	☐ Faire fonctionner le logiciel le plus longtemps possible
	☐ aucune bonne réponse.
3. Les tests en boites noires utilisent :	☐ Le code du logiciel à tester.
	☐ Les commentaires du logiciel à tester
	☐ Les spécifications du logiciel à tester.
	☐ Les commentaires de l'équipe de réalisation
	☐ aucune bonne réponse.

Questions	Réponses
1. La notion de couverture dans les tests :	☐ n'est pas utilisée.
	☐ Permet de couvrir une erreur faite lors des tests qui endommage le logiciel à tester
	☐ Sert à vérifier qu'un ancien jeu de valeurs peut être utilisé dans un nouveau contexte.
	☐ Sert à définir un objectif pour les tests.
	☐ aucune bonne réponse.

La figure suivante représente un bout de code dans un logiciel à tester :
`...`
`if (condition 1)`
`then statement 1`
`else statement 2 fi`
`if (condition 2)`
`then statement 3`
`fi`
`...`	

2. Dans le bout de code précédent, combien de cas de tests sont nécessaires pour atteindre le critère "toutes les instructions" sachant que les deux conditions sont indépendantes :	☐ Un cas de tests.
	☐ Deux cas de tests.
	☐ Trois cas de tests.
	☐ quatre cas de tests.
	☐ aucune bonne réponse.
3. L'interprétation abstraite est utilisée pour :	☐ Générer automatiquement des jeux de tests.
	☐ Prouver des propriétés sur les programmes sans les exécuter
	☐ Transformer un programme en un graphe de flot de contrôle.
	☐ Formaliser les besoins spécifiés par le client.
	☐ aucune bonne réponse.
4. Une méthode formelle de vérification représente :	☐ Une méthode mathématique qui permet de prouver qu'un programme vérifie une spécification.
	☐ Des jeux de tests utilisés pour vérifier l'absence d'erreurs
	☐ Un concept de programmation utilisé pour coder des jeux de tests.
	☐ une méthode pour formaliser les spécifications dans le but de les tester.
	☐ aucune bonne réponse.

Partie B : Exercices.

Questions	Réponses

Exercice 1 :

On veut s'assurer qu'une installation de machines en réseau fonctionne correctement. Les variables de cette installation sont l'OS des machines (Linux, Windows XP ou Mac Os X), le type de réseau utilisé (liaison Filaire ou WiFi), le type d'imprimante accessible (Canon1 HP2 ou LexMark3) et l'application utilisée (Word, Excel, Compta+, FireFox).

Questions :

1 Si l'on souhaite tester toutes les possibilités et si le test d'une configuration prend 10 min, combien de temps faut-il consacrer aux tests ?

2 Donner les jeux de tests permettant de tester au moins chaque valeur de chaque paramètre (critère « All Singles »)

3 Donner les jeux de tests correspondant à l'application de la méthode « All Pairs » (ou « Pairwise »). Justifiez votre réponse.

Exercice 2 :

On dispose d'une interface Java et de 5 classes qui implémentent cette interface. Voici le code Java correspondant :

```
1  public interface Fruit {
2  public boolean isSeedless(); //renvoie vrai si le fruit n'a
3  pas de noyau
4  }
5  public class Banane implements Fruit {
6  public boolean isSeedless() {return true;}
7  public String toString() {return "Banane";}
8  }
9  public class Ananas implements Fruit {
10 public boolean isSeedless() {return true;}
11 public String toString() {return "Ananas";}
12 }
13 public class Fraise implements Fruit {
14 public boolean isSeedless() {return true;}
15 public String toString() {return "Fraise";}
16 }
17 public class Cerise implements Fruit {
18 public boolean isSeedless() {return false;}
19 public String toString() {return "Cerise";}
20 }
21 public class Abricot implements Fruit {
22 public boolean isSeedless() {return false;}
23 public String toString() {return "Abricot";}
   }
```

Questions :

1 Créer une classe de test **BananeTest**, pour tester les cas suivants :
- tester qu'une banane n'a pas de noyau.
- tester l'affichage d'une banane (méthode toString()).
- tester l'égalité entre deux banane (méthode equals()).

Questions	Réponses

2 On souhaite représenter un cocktail de fruits, comme un fruit particulier. Créer une classe **Cocktail** qui implémente l'interface **Fruit** et contient les méthodes suivantes :
- méthode **String toString()** : qui retourne une représentation textuelle des fruits qui le composent (Représentation textuelle de votre choix).
- méthode **boolean isSeedless()** : qui retourne vrai si aucun des fruits contenus dans le cocktail ne contient de noyau.
- méthode **void add(Fruit f)** : qui ajoute le fruit « f » au cocktail (à noter qu'il est possible d'ajouter autant de fruits qu'on veut et même un cocktail dans une cocktail).
- Indication : vous pouvez utiliser le conteneur **java.util.ArrayList** pour représenter la liste de fruits composant la Cocktail.

3 Créer une classe de test **CocktailTest** pour tester la classe Cocktail, puis :
- créer dans cette classe une variable appelée **iles** correspondant à un Cocktail des iles (Cocktail composé d'Ananas et de Banane),
- on suppose que les variables suivantes sont déjà créées (il n'est pas demandé de les créer, mais on pourra s'en servir dans la classe) :
  - **vide** : correspond à une Cocktail vide.
  - **bananes** : correspond à une Cocktail de bananes.
  - **cerises** : correspond à une Cocktail de cerises.
  - **rouges** : correspond à une Cocktail aux fruits rouges (Fraise, Cerise).
  - **noyaux** : correspond à une Cocktail de fruits à noyaux (Cerise, Abricot).
- enfin créer une variable **laTotale** correspondant à un Cocktail contenant les 5 fruits.

4 Créer maintenant dans la classe **CocktailTest** des méthodes de test pour réaliser les tests suivants :
- pour chaque Cocktail crée, écrire le code permettant de tester si le Cocktail contient ou non des noyaux (méthode isSeedless()).
- tester ensuite l'affichage : uniquement pour les Cocktails **iles** et **vide** (méthode toString()).
- tester enfin la méthode **void add(Fruit f)** en vous servant de l'approche aux limites et des classes d'équivalences. Justifiez le choix des tests réalisés (vous pourrez vous servir d'un tableau pour recenser tous les cas particuliers à tester).

Bon courage et bonne continuation.