Examen de Génie Logiciel – 1ère session
M1 informatique

Responsable : C. Roudet

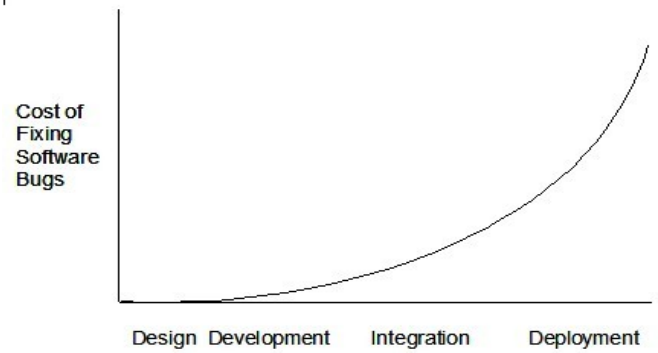
Documents de cours/TD/TP autorisés (livres interdits)
Calculatrices autorisées - Téléphones et ordinateurs portables interdits

22 mai 2015. Durée : 1h

Exercice 1 : Question d'ordre général - CHOISIR l'une des 2 questions (3 pts/10)

La qualité de la rédaction sera prise en compte ! FAITES DES PHRASES - 15 lignes MAXI !

1) Commenter le graphique ci-dessous et décrire les solutions qui ont été proposées par le Génie Logiciel pour en tenir compte.



2) Une étude de *Forrester Research* (publiée début janvier 2010) montre que les entreprises adoptent de plus en plus les méthodes "agiles" de développement (comme eXtreme Programming), qui sont censées être plus pragmatiques que les méthodes traditionnelles en cascade ou en V.
Qu'est-ce qui selon vous, pousse les entreprises à passer à ce modèle de développement de logiciels ? Quelles sont à votre avis les difficultés qu'elles peuvent rencontrer lors du passage d'une méthode traditionnelle à une méthode "agile" ? Justifiez vos réponses en les illustrant par des exemple.

Exercice 2 : Design Patterns - CHOISIR l'une des 2 questions (2 pts/10)

1) On dispose de données, soit dans une base de données, soit dans un fichier XML. Une classe `Analyseur` doit les lire avant avant de les traiter, les traitements étant différents dans les deux cas. Après traitement, les résultats doivent être écrits sous une forme identique dans un fichier XML. Donner le diagramme de classes UML correspondant, conforme au pattern **Template Method**, en précisant (grâce à un commentaire UML), le contenu de la méthode principale « templateMethod() ».

2) On veut définir une classe `Validateur` capable de valider différentes saisies (sous forme de `String`), par exemple la saisie de valeurs entières ou d'adresses mail. Donner le diagramme de classes UML correspondant, conforme au pattern **Strategy**, en utilisant des commentaires UML pour détailler au besoin certaines méthodes ou constructeurs.

../..

Exercice 3 : Graphe de flot de contrôle et tests unitaires en Java (5 pts/10)

Soit le programme en langage Java suivant :

```
 /* Classe d'analyse des chaînes de caractères, ne contenant qu'une
  * seule méthode statique booléenne. Cette dernière indique si une
  * chaîne de caractères "s" est (ou non) un palindrome.
  */
1  public class AnalyseChaines {
2
3   public static boolean palindrome(String s){
4      if (s.equals(null))                                          //A
5          return false;                                            //B
6     else
7     {
8       final int TAILLE = s.length();                             //C
9       int i = 0, j = TAILLE-1;                                   //D
10      while(i < TAILLE/2 && s.charAt(i) == s.charAt(j)){         //E
11         i++ ;                                                   //F
12         j-- ;                                                   //G
13      }
14      return (i >= TAILLE/2);                                    //H
15    }
16  }    // Attention « TAILLE/2 »  est une division entière : retourne la partie entière
17 }
```

- **Rappel** (*tiré de Wikipédia*) : un **palindrome**, aussi appelé **palindrome de lettres**, est une figure de style désignant un texte ou un mot dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche, comme dans la phrase « Esope reste ici et se repose » ou encore « La mariee ira mal ». On considère qu'une chaîne de caractères vide ou ayant une seule lettre sont des palindromes.

- Extrait de l'API Java :

`java.lang`

Class **String**

...

Methods :

Type	Method and Description
char	charAt (int index) Returns the char value at the specified index. An index ranges from 0 to length()-1. The first char value of the sequence is at index 0. **Throws:** IndexOutOfBoundsException - if the index argument is negative or not less than the length of this string.
int	length() Returns the length of this string. The length is equal to the number of Unicode code units in the string.

1) Dessiner le **graphe de flot de contrôle** correspondant à la méthode « palindrome » (en précisant pour chaque état du graphe à quelle ligne du programme Java il est associé) et donner sa **forme algébrique**.

2) Trouver les données de test (DT) minimales pour **couvrir toutes les instructions** (couverture de **tous les nœuds** du graphe de flot de contrôle). Pour chaque DT, indiquer le résultat attendu (« true » ou « false »).

3) Ces DT assurent-elles la **couverture de tous les arcs du graphe** ? Sinon ajouter de nouvelles DT pour couvrir **tous les arcs** (et préciser toujours le résultat attendu pour ces DT).

4) Ajouter au besoin des DT pour assurer :
- qu'on exécute la fonction **sans passer** dans la boucle while,
- qu'on exécute la fonction en passant **une fois** dans la boucle while,
- qu'on exécute la fonction en passant **deux fois** dans la boucle while.

5) La **ligne n°10** du programme contient une **conditionnelle composée** (deux expressions booléennes reliées par un **ET logique**). Ainsi, pour que les **tests** soient **complets**, il faut évaluer toutes les possibilités de cette conditionnelle, répertoriées dans le tableau ci-dessous :

ET logique	**Vrai**	**Faux**
Vrai	Vrai	Faux
Faux	Faux	Faux

Ajouter au besoin des DT (en précisant le résultat attendu), pour que les tests soient complets.

Important : si la première expression booléenne d'une conditionnelle composée (dont les expressions sont reliés par des **ET logiques**) est évaluée à faux, Java n'évaluera pas les expressions suivantes et retournera faux pour la conditionnelle composée.

6) Écrire maintenant, dans une **classe de test JUnit** nommée **`TestAnalyseChaines`** (en précisant si vous utilisez JUnit 3 ou 4), des **méthodes de test** pour tester la méthode « palindrome », pour **2 des DT** que vous venez d'identifier (**au choix**).

Remarque : inutile d'écrire les « import » en début de classe, nécessaires pour JUnit !