M2 IM
Université Paris–Sud
Centre d'Orsay

Calcul Scientifique
D5MA2N11: Genie Logiciel Pour l'entreprise
Année 2011 – 2012

Examen

Mardi 14 Février 2012 - Durée : 3 heures

Aucun document n'est autorisé.

Mettez votre numéro d'anonymat ici : .....

Exercice - 1 *Programmation extrême et défensive*

Q-1 **:** Regrouper les pratiques XP suivantes selon les catégories :

pratique développement (DvM), **pratique développeur (DvP)** et **pratique métier (MeT)**

en mettant les symboles (DvM), (DvP) ou (MeT) devant chacune.

- Développez les standards de développement.
- Ajoutez un client à l'équipe.
- Adoptez un développement piloté par les tests.
- Pratiquez inlassablement la restructuration.
- Programmer en binôme.
- Codez et concevez avec simplicité.
- Jouez le jeu de la planification.
- Adoptez la propriété collective du code.

Q-2 **:** Détailler pour chacune des trois recommandations suivantes :

Ajoutez un client à l'équipe - Adoptez la propriété collective du code - Jouez le jeu de la planification

1. Sa justification selon XP. (*Deux arguments suffisent*)
2. Les recommandations fournies par XP pour sa mise en oeuvre.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

Exercice - 2 *Normes de Programmation et documentation automatique*

Q-1 **:** Donner deux arguments soutenant la programmation par pseudo-code (PPP).

Q-2 **:** Pour définir un tableau de nombres réels ordonnés, un individu propose l'interface suivante :

```
class TAbleau_reelsOrdonnes{
public:
 typedef enum{ordre_Croissant, DECROISSANT} TypeOrdre;
 static double EPSILON;
   TAbleau_reelsOrdonnes(const TypeOrdre& type = ordre_Croissant);
   ~TAbleau_reelsOrdonnes();
   bool est_en_ordre_croissant()const;
   int Nombre_Elements()const;
   int position_element_median()const;
   double NormeEuclidienne()const;
   double norme_Max()const;
   double moyenne_arithmetique()const;
   double MoyenneGeometrique(int puissance_geometrique)const;
   void Insere_Element(double a);
   void supprime_element(int PositionElementASupprimer);
   double element_en_tete()const;
   double element_en_queue()const;
   bool EstVide()const;
private:
    TAbleau_reelsOrdonnes(const TAbleau_reelsOrdonnes&);
    TAbleau_reelsOrdonnes& operator=(const TAbleau_reelsOrdonnes&);
    double * elements;
    int  nb_elements;
    TypeOrdre _typeOrdre;
    void __inserElementAvecPrecision(double element, double precision=EPSILON);
};
```

Q-2-1 **:** Ce code respecte-t-il une norme de programmation ?
Q-2-2 **:** Peut-on le corriger afin de respecter une certaine norme de nomenclature ?
Q-2-3 **:** Les fonctions et données membres sont-elles en accès simple pour l'utilisation ? pour la maintenance ?

Q-3 **:** Fournir une documentation **doxygen** de la modification (**Q**-2-2) de la classe ci-dessus et expliquer la démarche de sa conversion au format **.html**.

On utilisera l'espace ci-dessous pour les réponses.

Exercice - 3 *Compilation automatique et gestion de version*

Q-1 **:** Citer trois outils de gestion de version de votre connaissance.

On utilisera l'espace ci-dessous pour les réponses.

.
.
.
.

Q-2 **:** Parmi les outils cités ci-dessus, lequel préférez-vous ? Soutenez votre argumentation en précisant votre environnement de développement favori.

On utilisera l'espace ci-dessous pour les réponses.

.
.
.
.
.
.
.
.
.
.
.
.
.
.

Q-3 **:** Dans un projet de développement basé sur des *autotools*, il arrive que l'on trouve dans un répertoire des fichiers suivants : *Makefile, Makefile.am, Makefile.in, configure.ac, configure.in, config.log, configure*.

- Lesquels de ces fichiers sont régénérés pendant la compilation ?
- Serait-il judicieux d'avoir une copie de ces fichier dans le dépôt SVN ?
- Enumérez certains fichiers dont la présence dans un dépôt SVN n'est pas nécessaire.

On utilisera l'espace ci-dessous pour les réponses.

.
.
.
.
.
.
.
.
.

Q-4 **:** Que vous évoque la notion de *licence* dans un code. Comment peut-on en savoir le contenu dans un projet basé sur les *autotools*. Comment mentionner la licence dans un fichier d'un projet en développement ?

On utilisera l'espace ci-dessous pour les réponses.

.
.
.
.
.
.
.
.

Exercice - 4 *Dépendance cyclique et structuration par composants*

Q-1 : Que signifie pour vous les sigles **CCD, ACCD, NCCD** ?

Q-2 : On considère les deux conceptions suivantes :

```cpp
//Fichier Sommet.hpp de A
#include "Point.hpp"
class Sommet
:public Point{
int m_couleur;
public:
// ...
};
//---------------------------
//Fichier Triangle.hpp de A
#include "Sommet.hpp"
class Triangle{
 Sommet m_sommets[3];
  public:
   //...
};
```

```cpp
//Fichier Sommet.hpp de B
#include "Point.hpp"
class Sommet{
 Point* m_point; //sans allocation
 int m_couleur;
 public:
  //...
};
//--------------------------------------
//Fichier Triangle.hpp de B
class Sommet;
class Triangle{
 Sommet* m_sommets[3];//sans allocation
 public:
  //...
};
```

Q-2-1 : Calculer leur CCD respective et dire lequel mettra moins de temps à compiler.

Q-2-2 : L'inclusion du fichier `Point.hpp` dans la conception de droite est-elle nécessaire ? Justifier et corriger si nécessaire.

On utilisera l'espace ci-dessous pour les réponses.

.
.
.
.
.
.
.
.

Q-3 : Énumérez trois techniques susceptibles de réduire le **temps de compilation** en C++ : vous identifierez

- Le nom qui lui est associé.
- La justification de son efficacité.
- Un exemple de code exhibant son utilisation.

On utilisera l'espace ci-dessous pour les réponses.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

Q-4 : Une des classes **B** et **C** ci-dessous (dans lesquelles les pointillés signifient qu'on a expressément omis certains détails) produit la dépendance cyclique. Identifier la, justifiez le et apportez-y une correction.

Corriger directement ici.

```
#include "A.hpp"

class B
{
 //...

private:
A* m_a;
};
```

Corriger directement ici.

```
#include "A.hpp"

class C
{
// ...

private:
A* m_a;
};
```

Corriger directement ici.

```
#include "B.hpp"
#include "C.hpp"
class A
{
 //...;
 private:
 B * m_b;
 C   m_c;
};
```

On utilisera l'espace ci-dessous pour la justification.

.
.
.
.
.
.

Q-5 : Illustrer sur un exemple une mise en oeuvre de la technique d'**escalation** et de **démolition** dans un processus de réduction de temps de compilation. (Utiliser la page suivante pour la réponse)

Q-6 : Vous disposer d'un code de résolution d'équations différentielles ordinaires ainsi qu'un fichier de test

```
class EDOSolver{
 Etat m_etat; // toutes les données
 public:
 //...
 int run(){
 //code qui résout l'edo et
 //stocke tout le nécessaire dans m_etat
          }
};
```

```
#include "EDOSolver.hpp"

int main(int argc, char**argv)
{
 //... quelques préparations ici
   EDOSolver solver(/*les paramètres*/);
   solver.run();
   return 0;
}
```

Un exemple de structure **Etat** serait par exemple

```
 struct Etat{
 int m_dimension;
 int m_iteration_courante;
 int m_nb_echecs;//dans la recherche du pas
 /*Et aussi
  une structure pour stocker: les INSTANTS - les PAS DE TEMPS - et les SOLUTIONS
  ... et bien d'autres choses ...
 */
};
```

Le jour du test d'intégration, votre patron vous demande d'exécuter le code et d'afficher l'évolution des itérations. Cela vous amène à modifier la fonction bf *run*, en y ajoutant quelques lignes de code, de sorte à satisfaire les attentes de votre Patron. Content de ce qu'il voit, votre Patron vous demande d'afficher aussi la solution toutes les 10 itérations, ce qui vous amène encore une fois à recompiler la classe **EDOSolver**.

Sachant que votre Patron reviendra ultérieurement vous demander d'afficher d'autres statistiques pendant la résolution, il vous semble naturel d'apporter une modification de sorte à ne plus devoir compiler à nouveau votre classe **EDOSolver**. Et même mieux de sorte à laisser à tout client de **EDOSolver** le soin d'apporter la statistique souhaitée.

- Comment procéderiez-vous si l'on suppose que toutes les attentes ultérieures de devront pas modifier la structure de la classe embarquée **Etat**.
- Pourriez-vous aussi anticiper une demande qui nécessiterait la modification de la la structure de l'objet stocké dans **Etat** ?

On utilisera l'espace ci-dessous pour les réponses à **Q-5**

On utilisera l'espace ci-dessous pour les réponses à **Q-6**

Exercice - 5 *Outils d'ajout de fonctionnalités : patron de conception ou design patterns*

Q-1 **:** On dispose d'une classe **`Maillage`** que l'on souhaite étendre dans le futur sans le compiler de nouveau.

Conception proposée par l' architecte **A**

```
//Class Maillage  de A
class Maillage{
public:
template <typename TypeManip>
void transform(const TypeManip& unManip){
 unManip.execute(*this);
}
};
```

```
//Inserteur de Points de A
class InserteurDePoints{

public:

void execute(Maillage&) const { "code ici" }

};
```

Conception proposée par l'architecte **B**

```
//Class Maillage  de B
class ManipMaillage;
class Maillage{
ManipMaillage* m_manipMaillage;
public:
void chargeManip(ManipMaillage* unManip){
 unManip->chargeCible(*this);
 m_manipMaillage = unManip;
}
void transform(){
     m_manipMaillage->execute();
  }
};
```

```
//manipulateur de base de B
class ManipMaillage{
Maillage* m_maillageCible;
public:
void chargeCible(Maillage& unMaillage);
virtual void execute() = 0;
};
```

```
//Inserteur de Points de B
class InserteurDePoints
   : public ManipMaillage{
public:
void execute(){ "code ici" }
};
```

- Ces conceptions sont-elles susceptibles de résoudre le problème ? .....
- A quels motifs de conception vous renvoient-elles ? .....
- Lequel préfériez-vous ? ..... Pourquoi ? .....
- Expliquer comment ajouter une opération pour l'écriture du maillage dans un fichier, avec chaque conception :

Avec la conception A

```
.
.
.
.
.
.
.
.
```

Avec la conception B

```
.
.
.
.
.
.
.
.
```

Q-2 **:** Compléter la table suivante, en fournissant le pattern correspondant à la description donnée

DESCRIPTION	PATTERN CORRESPONDANT
Enveloppe un objet et fournit une interface différente pour y accéder	Adaptateur
Permet de notifier les changements d'état à des objets	
Les sous-classes décident de la façon d'implémenter les étapes d'un algorithme	
Les sous-classes décident quelles sont les classes à créer	
Garantit qu'un objet et un seul est créé	
Encapsule les comportements interchangeables et utilise la délégation pour décider lequel utiliser	
Les clients traitent des collections d'objets et les objets individuels de la même manière	
Encapsule des comportements basés sur des états et utilise la délégation pour permuter ces comportements	
Fournit un moyen de parcourir une collection d'objets sans exposer son implémentation	Itérateur
Simplifie l'interface d'un ensemble de classes	
Enveloppe un objet pour fournir un nouveau comportement	
Permet à un client de créer des familles d'objets sans spécifier leurs classes concrètes	
Enveloppe un objet et en contrôle l'accès	
Encapsule une requête sous forme d'objet	Commande

On utilisera l'espace ci-dessous pour les réponses