LOG1000 - Ingénierie logicielle

Automne 2012

Examen intra

Chargé de cours : Julien Gascon-Samson

Date : jeudi le 18 octobre 2012

Durée : 2 heures, de 13h45 à 15h35

Cet examen comporte 5 questions

Pondération : 25% de la note finale

Directives :
Toute documentation permise.
Calculatrices, téléphones cellulaires, tablettes et ordinateurs non permis.
Remettez le questionnaire dûment signé.

1 Questions générales (10 points)

Pour les dix questions suivantes, choisissez la réponse qui vous semble être la bonne et indiquez votre choix **en remplissant le tableau au début de la page 3** (0,5 point par question).

1. Pour lequel des types de systèmes logiciels suivants peut-on présumer que l'activité de maintenance sera la plus longue ?
   a) Logiciel commercial
   b) Logiciel embarqué
   c) Logiciel de jeu
   d) Logiciel scientifique
   e) Logiciel patrimonial
2. Parmi les activités suivantes, laquelle constitue une activité d'encadrement ?
   a) Écriture du code
   b) Préparation des tests unitaires
   c) Gestion de projet
   d) Analyse
   e) Maintenance
3. Quel est le principal artéfact produit lors de l'activité de définition du problème ?
   a) Document de spécifications (*SRS*)
   b) Document de vision
   c) Document d'architecture
   d) Diagrammes UML
   e) Code source
4. Parmi les choix suivants, quel est l'objectif priorisé par l'activité de conception ?
   a) Faciliter la maintenance
   b) Favoriser l'extensibilité
   c) Limiter le système à ce qui a été spécifié
   d) Réduire la complexité
   e) Favoriser la réutilisabilité
5. À laquelle des activités du développement logiciel la correction d'un défaut coûte-t-elle le plus cher ?
   a) Maintenance
   b) Spécifications
   c) Implémentation
   d) Tests
   e) Architecture

6. Vous effectuez la conception d'une application mobile de réseautage social. Vous souhaitez que votre application puisse s'exécuter autant sur un *iPhone* que sur un appareil *Android*. À cette fin, au moment de la conception, vous décidez que la très grande majorité de l'application mobile sera rédigée en HTML5, CSS et Javascript. À quelle caractéristique de conception fait-on référence ?

a) Complexité minimale
b) Techniques standards
c) Extensibilité
d) Portabilité
e) Réutilisabilité

7. En tant qu'ingénieur logiciel junior, vous commercialisez un système de gestion de facturation pour les PME, écrit en Java. Un de vos clients a déployé le système sur des stations de travail Windows 7. Votre client a récemment fait l'acquisition de stations de travail MacOSX et Fedora Core (Linux). Puisqu'une machine virtuelle Java est disponible sur toutes ces plate-formes, votre client est en mesure d'utiliser votre système de gestion de facturation de manière transparente sur toutes ces plate-formes. À quelle caractéristique de qualité logicielle fait-on référence ?

a) Réutilisabilité
b) Adaptabilité
c) Utilisabilité
d) Flexibilité
e) Portabilité

8. Laquelle des activités suivantes fait partie de la construction logicielle ?

a) Analyse des besoins avec le client
b) Préparation d'un diagramme de classes
c) Rédaction du *SRS*
d) Préparation de cas d'utilisation
e) Aucune de ces réponses

9. Parmi les rôles suivants, lequel ne devrait pas assister à une réunion d'inspection formelle ?

a) Scribe
b) Modérateur
c) Gestionnaire
d) Auteur
e) Réviseur

10. Parmi les techniques suivantes, laquelle (lesquelles) peut (peuvent) être réalisée(s) uniquement par le développeur sans l'aide d'un tiers ?

a) Contrôle personnel du code
b) Tests unitaires
c) Prototypage
d) Toutes ces réponses
e) Aucune de ces réponses

Pour les questions 1 à 10, écrivez lisiblement la lettre corrrespondant à votre réponse.

1	2	3	4	5	6	7	8	9	10
e	c	b	d	a	d ou b	b	b	c	d

Pour chaque énoncé qui suit, dites s'il est vrai ou faux (0,5 point par énoncé). **Attention : pour toute mauvaise réponse, une pénalité de 0,25 s'appliquera**.

		Vrai	Faux
11	Selon l'échelle CMMI, un processus de développement d'un système logiciel de maturité *1* sera généralement plus complexe qu'un processus de développement de maturité *5*.		X
12	Dans le contexte d'un processus logiciel, la réalisation d'une activité mène à la production d'un artéfact.	X	
13	Les tests unitaires permettent généralement d'identifier les causes et symptômes des défauts logiciels.		X
14	L'interface utilisateur d'un système logiciel constitue une spécification fonctionnelle.	X	
15	Puisque le *SRS* fait partie de l'entente contractuelle avec le client, il doit préciser les solutions techniques retenues par l'équipe de développement.		X
16	Le prototypage permet de développer à l'avance certaines portions critiques du système au sein du produit final.		X
17	Dans le contexte de développement d'une application web, la technique de conception *stratification* peut s'avérer très utile.	X	
18	L'utilisation de variables globales permet généralement d'aider à réduire la complexité.		X
19	Dans un diagramme de cas d'utilisation, le stéréotype extension ("extends") est utile pour spécifier un cas d'utilisation alternatif qui s'exécute sous certaines conditions.	X	
20	Un modèle de développement itératif/évolutif est approprié lorsque le niveau de risque est élevé.	X	

2 Réponses à développement (3 points)

Répondez de manière claire et concise aux questions suivantes.

a) (0,5 pt) Quelle est la différence entre un diagramme de cas d'utilisation et un cas d'utilisation détaillé ?

Diagramme : affiche un ensemble de cas d'utilisation sous forme graphique.
Cas détaillé : affiche les détails sous forme textuelle (déroulement alternatif, pré-post conditions, etc.)

b) (0,5 pt) Lors de la conception, nous effectuons une distinction entre complexité "essentielle" et complexité "accidentelle". En vos mots, quelle est la différence ces deux concepts ?

Complexité essentielle : complexité qui a trait à la difficulté du problème. Complexité accidentelle : complexité qui résulte des solutions choisies au problème.

c) (0,5 pt) Dans le jargon des spécifications, qu'est-ce que la *traçabilité* ?

Possibilité de tracer des liens entre une spécification, la portion de conception, d'implémentation, de tests etc. correspondant à cette spécification.

d) (0,5 pt) Quelle est la différence entre les spécifications fonctionnelles et non-fonctionnelles ?

Fonctionnelles : concerne des fonctionnalités offertes par le système. Non-fonctionnelles : concerne des aspects autres que les fonctionnalités du système (sécurité, performance, ressources, etc.).

e) (0,5 pt) Décrivez, dans vos mots, pourquoi le couplage sémantique est une mauvaise pratique.
La classe/méthode doit connaître le fonctionnement interne d'une autre classe/méthode donnée.

f) (0,5 pt) Comme nous avons eu l'occasion de le constater en cours, bien que les tests unitaires présentent une certaine efficacité, les techniques de révision et de bêta-tests sont généralement plus efficaces. En dépit de ce phénomène, les tests unitaires constituent une pratique très encouragée dans le domaine de l'ingéniérie logicielle. Expliquez brièvement pourquoi l'utilisation de tests unitaires s'avère une pratique intéressante.
Une fois programmés, ils peuvent être réexécutés périodiquement et même de façon automatisée. Il s'agit d'un investissement initial qui peut être réutilisé tout au long du développement. Ils peuvent aussi servir à détecter les erreurs de régression.

3 Modélisation UML (2 points)

a) (0,5 pt) Quelle est la différence fondamentale entre un diagramme structurel et un diagramme comportemental ?

Structurel : diagramme illustrant le système d'un point de vue statique. Comportemental : diagramme illustrant le système d'un point de vue dynamique.

b) (0,5 pt) Dans quel contexte un diagramme de déploiement s'avère-t-il utile ?

Un diagramme de déploiement s'avère utile lorsqu'un système logiciel peut être découplé en plusieurs composantes qui peuvent être déployées sur plusieurs plate-formes / machines.

c) (1 pt) Démontrez, sous forme de diagramme de classes, un court exemple illustrant votre compréhension des relations UML suivantes : héritage, agrégation, composition et dépendance. Utilisez les symboles graphiques de relations entre les classes propres à UML 2.0. N'indiquez pas les méthodes ni les attributs. Restreignez votre diagramme à un ensemble minimal de classes permettant d'illustrer chacune des quatre relations mentionnées précédemment. Les contraintes suivantes doivent être respectées :
- Un seul diagramme de classes doit illustrer l'ensemble des relations.
- Les exemples choisis doivent être différents des exemples montrés en classe.

(Correction au cas-par-cas - pour avoir les points, les 4 relations doivent être mentionnées avec des exemples différents de ce qui a été vu en cours. La bonne représentation graphique (style de flèches) doit être utilisée.)

4 Makefile (3 points)

Vous développez une application Java en deux modules : *application* et *librairie*. **Le module *application*, qui contient le *main* (similaire au *main* en C++) fait appel au module *librairie*.** Votre tâche est d'écrire un Makefile complet permettant la compilation de l'application Java en respectant les directives données.

En Java, un fichier source porte l'extension *.java*. Dans le cas présent, nous avons deux fichiers source : *application.java* et *librairie.java*. Lorsque le compilateur est exécuté, un fichier *.class* est généré pour un fichier *.java* correspondant. Ce fichier est équivalent aux fichiers *.o* intermédiaires générés par le compilateur C++. Nous pouvons exécuter l'application au moyen des fichiers *.class* compilés. Nous avons aussi la possibilité de générer un fichier *.jar* (Java Archive) qui contient notamment l'ensemble des fichiers *.class*. L'exécution peut aussi être réalisée au moyen du fichier *.jar*. Les principales commandes sont :

- Compilation de *fichier.java* en *fichier.class* : `javac fichier.java`
- Création d'un fichier *.jar* à partir d'un ensemble de fichiers *.class* : `jar cfe programme.jar main main.class fichier1.class fichier2.class fichier3.class`
  a) L'argument `cfe` spécifie une combinaison de drapeaux qui indiquent que l'on souhaite créer une archive en spécifiant son nom et le point d'entrée (méthode *main*)
  b) L'argument suivant spécifie le fichier archive *.jar* à générer
  c) L'argument en italique, *main*, spécifie le fichier qui possède la méthode main, sans extension. Dans la ligne d'appel de la commande *jar* démontrée en exemple, il s'agit de `main.class` (préalablement compilé à partir de `main.java`) qui contient une méthode *main*.
  d) Les arguments subséquents (`main.class`, `fichier1.class`, `fichier2.class`, et `fichier3.class`) spécifient l'ensemble des fichiers compilés qui doivent être inclus dans l'archive *.jar*

a) (2 pts) Votre tâche est de rédiger un *Makefile* complet possédant la même structure que les Makefile vus en cours. Ce Makefile doit permettre à l'utilisateur de :
   - Compiler chaque fichier *.java* en un fichier *.class* (une cible par fichier, comme en C++)
   - Générer un fichier *.jar* à partir de l'ensemble des fichiers *.class*
   - Compiler l'ensemble des fichiers *.java* **et** générer le fichier *.jar* (cible par défaut)
   - Nettoyer l'ensemble des fichiers de compilation produits

```
all: application.jar #facultatif si application.jar est la première cible

application.jar: application.class librairie.class
   jar cfe application.jar application application.class librairie.class

application.class: application.java librairie.java #ou librairie.class
   javac application.java

librairie.class: librairie.java
   javac librairie.java

clean:
   rm -f *.class
   rm -f application.jar #ou *.jar
```

b) (1 pt) Dans le contexte général d'un Makefile, lorsque l'utilisateur exécute *make* en spécifiant le nom d'une cible et en assumant que cette cible ait des dépendances, comment l'utilitaire *make* s'y prend-il pour déterminer si la cible doit être reconstruite (2 éléments de réponse) ?

1) Si les fichiers dépendants sont plus récents que la cible.

2) Si la cible n'existe pas.

5 SVN (2 points)

a) (1 point) Vous avez un entrepôt SVN contenant les fichiers suivants : *application.cpp* et *readme.txt*. Vous décidez de créer un répertoire *source* et d'y déplacer le fichier *application.cpp*. Les modifications doivent être appliquées à l'entrepôt. On suppose que la copie locale est à jour. Quelle(s) commande(s) complète(s) est-il nécessaire d'exécuter ?

```
mkdir source
svn add source svn rename application.cpp source/application.cpp
svn commit -m 'Déplacement du fichier'
```

b) (0,5 pt) Un autre utilisateur a modifié le fichier *readme.txt*. Vous souhaitez afficher les modifications afin de le comparer avec votre version. Quelle(s) commande(s) faut-il exécuter ?

```
svn update
svn diff readme.txt
```

c) (0,5 pt) Vous mettez à jour l'entrepôt (`svn update`) et SVN vous indique qu'il y a un conflit au sein du fichier *readme.txt*. Vous modifiez alors manuellement le fichier *readme.txt* afin de supprimer le conflit. Quelle(s) commande(s) SVN faut-il exécuter par la suite ?

```
svn resolved readme.txt
svn commit -m 'Résolu conflit'
```