LOG1000 - Ingénierie logicielle

Hiver 2013

Examen intra

Enseignants : Bram Adams et Julien Gascon-Samson

Date : lundi le 18 février 2013

Durée : 2 heures, de 18h30 à 20h30

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 à la fin de la page 3** (0.5 point par question).

1. Parmi les éléments suivants, lequel n'a aucun lien avec l'intégration ?

   a) SVN

   b) Makefile

   c) Git

   d) Tests unitaires

   e) Aucune de ces réponses

2. Un processus de développement itératif/évolutif travaille typiquement de façon incrémentale, c'est-à-dire :

   a) On fait l'analyse des requis, la conception du logiciel, l'implémentation et les tests l'un après l'autre, et ça génère le système complet en une fois.

   b) On fait l'analyse des requis, la conception du logiciel, l'implémentation et les tests l'un après l'autre, et ça génère déjà 80% du système en une fois. La loi de 20/80 dit qu'il ne vaut plus la peine de continuer après pour le 20% restant.

   c) On construit tout le système en une fois, et après on commence de nouveau à partir de zéro avec une autre technologie qui est beaucoup mieux.

   d) On construit une partie du système en faisant l'analyse des requis, la conception du logiciel, l'implémentation et les tests. Après, on mettra en production cette partie avant de continuer avec la même approche de construction pour la prochaine partie.

   e) Des développeurs ont tendance de copier leur code source d'autres projets, et de le modifier (évoluer) petit à petit pour le projet en cours.

3. Plusieurs utilisateurs ont rapporté que la mise à jour la plus récente du microgiciel iOS 6.1 pour téléphones intelligents "iPhone" diminuait la durée d'autonomie de la batterie. À quelle caractéristique externe de qualité logicielle fait-on référence ?

   a) Robustesse

   b) Fiabilité

   c) Efficience

   d) Utilisabilité

   e) Aucune de ces réponses

4. Parmi les éléments suivants, lequel ne devrait généralement pas être placé sous contrôle de version ?

   a) SRS

   b) Code source développé par l'équipe

   c) Tests unitaires

d) Version compilée du code source développé par l'équipe

e) Aucune de ces réponses

5. La phrase "Il coûte moins cher de corriger une erreur à la conception que de la détecter plus tard et corriger le code qui en résulte" veut dire que :

a) Les architectes gagnent moins d'argent que les développeurs.

b) Les architectes sont plus intelligents que les développeurs et plus faciles à convaincre.

c) Une erreur à la conception résulte en des choix technologiques et des activités de développement et débogage inutiles qui coûtent beaucoup plus d'argent qu'éviter ces dépenses directement au début.

d) Comme le coût majeur d'un projet sont les salaires des gens et qu'il y a beaucoup plus de développeurs que d'architectes, corriger une erreur à la conception coûte beaucoup moins cher.

e) Aucune réponse n'est correcte.

6. Parmi les éléments suivants, lequel correspond le mieux à la préparation de l'architecture logicielle à haut niveau ?

a) Diagramme de composants

b) Diagramme de classes

c) Diagramme d'états-transitions

d) Diagramme de cas d'utilisation

e) Diagramme de séquence

7. Le masquage d'information est un principe dans le développement du logiciel pour :

a) organiser la conception (design) d'un logiciel en tenant compte des sources de complexité du système et de former des composants et des interfaces autour de cette complexité.

b) éviter que plusieurs gens aient accès et la permission pour changer les mêmes composants. Ce principe est l'inverse de la construction collaborative et est utilisé typiquement pour le processus en cascade.

c) encrypter tous les artéfacts du processus de développement pour augmenter la sécurité et protéger la propriété intellectuelle.

d) ridiculiser la quantité excessive de documentation générée dans le processus de développement en cascade.

e) Aucune réponse n'est correcte.

8. Lors de la création de son navigateur Safari pour OSX et Windows, Apple a adapté l'engin de rendu KHTML provenant du navigateur Konqueror [1]. Google a, par la suite, utilisé le même engin au sein de son propre navigateur, Chrome. À quelle caractéristique interne de qualité logicielle cela correspond-il ?

a) Adaptabilité

b) Flexibilité

1. Le navigateur par défaut de l'environnement de bureau KDE sous Linux

c) Portabilité

d) Maintenabilité

e) Compréhensibilité

9. La loi de Linus Torvalds veut dire que :

a) Plus élevé le nombre de gens qui interprètent un bogue rapporté, plus bas sera la priorité assignée à un bogue, ce qui permet de trier des bogues plus rapidement.

b) Si assez de développeurs et testeurs analysent un fichier de code source, plus de bogues et plus de types différents de bogues peuvent être découverts et résolus dans une période plus courte.

c) Au contraire, plus élevé le nombre de développeurs et testeurs qui analysent un fichier de code source, plus d'opinions contradictoires apparaissent et plus difficile est la tâche de se mettre d'accord sur les vrais bogues.

d) Le nombre de développeurs est une indication de la qualité des produits d'une organisation.

e) Aucune réponse n'est correcte.

10. Extreme programming est ...

a) un sport provenant du japon où on essaie de programmer aussi longtemps que possible afin de gagner des prix de grandes valeurs, comme un XBox 360 avec Kinect ou un iPad 4.

b) la discipline de programmation des téléphones intelligents et d'autres appareils embarqués (par exemple des RFIDs).

c) une implémentation spécifique d'une approche agile.

d) une implémentation spécifique d'une approche en cascade.

e) une collection d'idées de développement à partir desquelles l'approche agile est une implémentation spécifique.

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

1	2	3	4	5	6	7	8	9	10

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	Le document SRS joue un rôle primordial lorsque l'on travaille selon une approche Agile, puisqu'il permet d'aider à prioriser les scénarios à réaliser pour la prochaine itération.		
12	Une approche séquentielle est généralement appliquable si les spécifications sont peu sujettes à changement.		
13	Le SRS est produit pendant la construction logicielle.		
14	Un cas d'utilisation est l'un des éléments du langage de modélisation UML.		
15	Le document de vision sert à détailler les spécifications.		
16	Il faut d'abord développer des tests, parce que ça permet aux testeurs de travailler sur d'autres projets pendant le développement.		
17	Une librairie comme le Facebook Java API 2.1 utilisée par notre projet est un artéfact important qui doit être versionné.		
18	Le modèle en cascade est optimisé pour des projets web comme Facebook et eBay.		
19	Une architecture avec un fan-out élevé est très réutilisable.		
20	Des patrons de conception font partie d'une librairie qui peut être téléchargée et intégrée dans un logiciel pour améliorer la conception.		

2 Questions variées (3 points)

Répondez de manière claire et concise aux questions suivantes.

a) (0.25 pt) Quelle est la différence entre une activité de développement et une activité d'encadrement ?

b) (0.5 pt) Considérez l'architecture logicielle illustrée à la figure 1. S'agit-il d'un bon ou d'un mauvais exemple d'architecture ? Justifiez votre réponse.

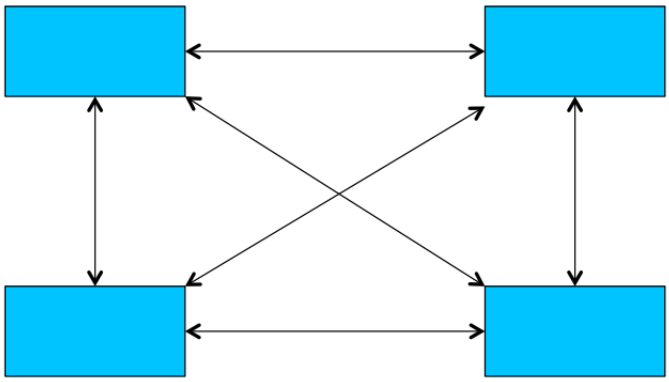

FIGURE 1 – Architecture logicielle

c) (1 pt) Il est généralement démontré que les bêta-tests à grand volume constituent une bonne pratique d'assurance qualité. Il y a toutefois certains inconvénients. Quels sont les deux aspects principaux négatifs avec cette approche d'assurance qualité ?

d) (0.75 pt) Séparer l'architecture logicielle d'un système en plusieurs couches est, la plupart du temps, une bonne pratique. Un cas classique d'un modèle multi-couches est le modèle à trois couches, souvent utilisé pour le développement d'applications web. Décrivez très brièvement le rôle de chacune des couches d'un tel modèle.

e) (0.5 pt) Nommez et expliquez quatre métriques (mesures) *différentes* reliées à l'assurance-qualité du code (implémentation) (0.25 pt seront attribués par deux métriques acceptables mentionnées).

3 Gestion de configuration (2 points)

Répondez de manière claire et concise aux questions suivantes.

a) (0.75 pt) Dans le contexte d'un système de gestion de configuration, il existe deux approches pour gérer le fait que plusieurs développeurs travailleront sur le même ensemble de fichiers.

(a) Mentionnez ces deux approches. (0.25 pt)

(b) Mentionnez un avantage et un inconvénient pour chacune de ces deux approches. (0.5 pt)

b) (0.75 pt) Supposez un entrepôt SVN contenant un fichier `system.conf` placé sous contrôle de version. Deux collaborateurs ont accès à une copie locale de l'entrepôt et y travaillent : Pierre et Luc. Voici la séquence de commandes exécutées par chacun d'entre eux.

#	Pierre	Luc
1	`svn update`	
2	`cat "Ubuntu" > system.conf`	
3		`svn update`
4		`cat "Fedora" > system.conf`
5	`svn update`	
6	`svn commit -m "Ubuntu OS"`	
7		`svn update`
8		`svn commit -m "Fedora OS"`

Notez que la commande `cat "bla" > file.txt` remplace le contenu de fichier file.txt par "bla".

Suite à l'exécution de ces commandes :

– Quel problème surviendra et qui le rencontrera ? (0.25 pt)

– Quelles actions sont nécessaires pour régler ce problème ? Ne donnez que des explications, pas de commandes. (0.5 pt)

c) (0.5 pt) Nous avons vu les systèmes de gestion de version SVN et Git en classe. Quelle est la différence architecturale fondamentale de ces deux systèmes ? Vous pouvez utiliser des dessins pour clarifier votre réponse.

4 La construction collaborative et les approches agiles (3 points)

Répondez de manière claire et concise aux questions suivantes.

a) (1 pt) Votre organisation doit implémenter le logiciel pour contrôler un nouveau type de rasoir. Le document de vision mentionne quatre fonctionnalités révolutionnaires, dont les fonctionnalités A et B sont absolument nécessaires, la fonctionnalité C est “nice to have”, et la fonctionnalité D est optionnelle. Expliquez en mots et avec un calendrier comment une approche agile produirait le logiciel. En plus, expliquez ce qui se passe si, après 8 semaines, la fonctionallité B est annulée et remplacée par une nouvelle fonctionnalité E.

b) (1 pt) Qu’est-ce que l’intégration et pourquoi est-ce qu’il faut la faire fréquemment ?

c) (1 pt) Une équipe agile permet à tout le monde de changer n'importe quel fichier. Pourquoi est-ce que l'on est si confiant que ces changements ne tomberont pas mal ? Donnez quatre moyens.

5 Makefile (2 points)

Pendant un des TPs, vous travaillez avec un de vos collègues sur un robot programmé en C. Comme le père de votre collègue travaille pour la ville de Montréal dans le domaine de la construction, votre collègue s'est engagé volontairement pour écrire le Makefile du programme C :

```
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic
LDFLAGS=
EXEC=hello
OUTIL=generez_hello_h.sh
MOT_DE_PASSE_SECRET=bramadams

all: $(EXEC)

hello: hello.o main.o
        $(CC) -o hello hello.o main.o $(LDFLAGS)

hello.o: hello.c
        $(CC) -o hello.o -c hello.c $(CFLAGS)

main.o: main.c hello.h
        $(CC) -o main.o -c main.c $(CFLAGS)

hello.h: hello_there.h
        $(OUTIL) $(MOT_DE_PASSE_SECRET) > hello.h

clean:
        rm -rf *.o

mrproper: clean
        rm -rf $(EXEC)
```

Malheureusement, avant la date limite, votre collègue doit quitter le pays avec sa famille pour quelques mois. Comme il a fait beaucoup de travail, vous êtes d'accord de finir le projet vous-même.

Dix minutes avant la date limite, vous réalisez qu'il y a un bogue grave dans le code qui rend le robot aggressif dès qu'il voit un canard jaune. Comme le chargé de lab a un canard jaune sur son bureau et exécute toujours les projets avant de les noter, vous paniquez à l'idée que le chargé ne sera plus capable de vous donner vos notes. Alors, vous commencez à naviguer les fichiers. D'abord, vous changez le fichier $hello.h$. Vous modifiez, en particulier, la valeur de la variable globale $ennemi$, qui est "$CANARD$", pour "". Après, vous corrigez une faute d'orthographe dans un commentaire du fichier $hello\_there.h$.

Comme il ne reste que quelques minutes avant que la page de soumission sur Moodle ferme, vous compilez rapidement le projet avec :

```
make clean
make
```

... et vous soumettez l'exécutable $hello$ avec le code source.

a) (0.5 pt) Est-ce que le chargé de cours sera blessé par votre robot ? Pourquoi (pas) ?

b) (1.0 pt) Quels fichiers sont supprimés ou régénérés lors de l'exécution des commandes `make clean` et `make` ?

c) (0.5 pt) Qu'est-ce qui se serait passé si vous aviez utilisé les commandes suivantes après vos changements (au lieu de `make clean` et `make`) :

```
gcc -o hello.o -c hello.c -W -Wall -ansi -pedantic
gcc -o hello hello.o main.o
```