LOG1000 – Ingénierie Logicielle TP5

Restructuration du code source
Hiver 2017

Objectifs :

- **Identifier** les mauvaises odeurs dans le code source (« bad smells ») nécessitant une restructuration, sur plusieurs niveaux.
- **Restructurer** le code ayant des mauvaises odeurs en utilisant des restructurations adéquates, et ce, en plusieurs phases consécutives.
- **Compiler et tester** le code restructuré après chaque phase pour valider la viabilité des nouvelles implémentations.

Notes importantes:

- Donnez des réponses courtes, claires et précises.
- Donnez votre réponse sous forme de liste quand cela est possible.
- Vérifier la lisibilité des captures d'écrans que vous allez mettre dans votre rapport.
- Les ressources qui seront consultées en ligne seront en anglais. Si vous voulez de l'aide de traduction pour certains termes techniques, n'hésitez pas à en faire part au chargé de laboratoire.

Enoncé :

Dans cet énoncé, vous devez identifier et restructurer **les mauvaises odeurs** présentes dans un programme C++.

Le groupe TVA a un composant logiciel qui permet de gérer ses émissions. Ce groupe a jugé que le code nécessite certaines restructurations, dans le but qu'il soit maintenable et fiable à long terme, et ainsi ils font appel à vous pour le restructurer.

Cette **restructuration du code** consiste à modifier la structure interne du logiciel pour le rendre plus facile à comprendre et évoluer, évidemment SANS changer son comportement observable !

Dans le logiciel, le fichier « main.cpp » permet de proposer à un utilisateur les opérations possibles qu'il peut effectuer, par exemple, ajouter et enregistrer une émission. Les émissions sont gérées par la classe « Emission » qui est définie dans le fichier «Emission.h » et implémentée dans le fichier « Emission.cpp ». Pour effectuer des tests sur le programme, donne une base de données enregistrée dans le fichier «DB.txt ».

La classe «Emission » a besoin d'être restructurée dans E1 et E2, tandis que E3 vise à restructurer la méthode « main ». Pour plus de détails sur la restructuration du code, veuillez consulter le lien suivant : http://www.professeurs.polymtl.ca/michel.gagnon/Smells/

E1) Une mauvaise odeur dans les attributs [/30]

En examinant les attributs de la classe « Emission », vous pouvez remarquer qu'elle joue deux rôles différents, et ainsi elle est faiblement cohésive, parce qu'il y a des attributs représentant des informations sur la chaîne et d'autres attributs qui représentent des informations sur une émission.

Expliquez pourquoi c'est une odeur grave. **[/2]**

Identifiez le nom de la restructuration nécessaire pour enlever cette odeur du code. **[/2]**

Identifiez les méthodes qui seront impactées par ce changement. **[/6]**

Identifiez les attributs qu'il faut modifier ou déplacer. **[/3]**

Identifiez les étapes à suivre pour restructurer cette odeur. Utilisez le même format du tableau ci-dessous, dans lequel vous décomposez la restructuration globale en étapes plus simples **[/5]**

Étape	Description

Restructurez le code source en modifiant/déplaçant les attributs de la question 4 et en modifiant les méthodes impactées (question 3). (Notez que vous avez

le droit de créer une nouvelle classe en cas de besoin). Copiez la/les classe(s) modifiée(s) dans le rapport (le fichier header et cpp). **[/6]**

Compilez et exécutez les tests unitaires fournis (/tests/…), veuillez ajouter des captures d'écrans des résultats de vos tests dans le rapport. **[/4]**

Faites un commit de votre code source, et une capture d'écran de ce commit dans le rapport (Le résultat de la commande « git add, commit, et push »). **[/2]**

E2) Une mauvaise odeur dans les méthodes [/20]

En examinant la méthode « TrouverEmission» de la classe « Emission.cpp » :

Identifiez le nom des deux odeurs graves et expliquez pourquoi ce sont des odeurs graves. **[/2]**

Planifiez, étape par étape, comment restructurer cette odeur, dans le même format du tableau de l'exercice E1. **[/6]**

Restructurez le code source de cette méthode. Copiez dans le rapport le nouveau code de la méthode, ainsi que d'autres méthodes si vous en créez des nouvelles ou si vous modifiez d'autres méthodes dans cette restructuration. **[/6]**

Compilez et exécutez les tests unitaires fournis (/tests/…), veuillez ajouter des captures d'écrans des résultats de vos tests dans le rapport. **[/4]**

Faites un commit de votre code source, et une capture d'écran de ce commit dans le rapport. **[/2]**

E3) Utilisation des variables [/22]

Afin d'améliorer la compréhension et la lisibilité du code, il est important de minimiser **le span, la durée de vie** et **la portée** des différentes variables :

1) Calculez le span, la durée de vie et la portée des variables « DBFile», «choix » et « emission» dans la méthode « main » dans « main.cpp ». Les lignes vides ne comptent pas ! **[/6]**

Interprétez les résultats, et trouvez la variable (parmi les trois citées en dessus) qui bénéficiera le plus de la restructuration. **[/2]**

Proposez des restructurations pour améliorer l'utilisation de cette variable, en utilisant le même format du tableau de l'exercice E1. **[/4]**

Effectuez cette restructuration dans la méthode « main ». Faites une capture d'écran (ou copiez le code) de votre nouveau code source. **[/4]**

Compilez et testez manuellement (en exécutant le programme sur la ligne de commandes) les opérations (de l'opération 0 à 4) de la méthode « main », veuillez prendre des captures d'écrans de vos tests. **[/4]**

Faites un commit de votre code source, et une capture d'écran de ce commit dans le rapport. **[/2]**

E4) Questions de cours [/10]

i. Citez deux avantages de l'utilisation des APIs. Quel problème un développeur peut avoir en utilisant une API non stable ? [/1]

ii. Citez deux librairies qui fournissent une API de bonne qualité. [/1]

iii. Comme la plupart de vous ont déjà touché à la programmation orienté objet, citez des erreurs que vous avez faites. Et comment allez vous les éviter après la lecture de “les APIs et les classes” ? [/1]
iv. Quelles sont les critères qu’on doit prendre en considération lors de l’implémentation d’une méthode ? [/1]
v. Pour faciliter la lecture de votre code, quels sont les pratiques que vous ne devez pas oublier ? [/1]
vi. Quelle est la différence entre une “Cohésion fonctionnelle” et une “Cohésion de communication” ? [/1]
vii. Définissez “patron de conception”, “anti-patron de conception” et “mauvaise odeur du code” et le lien entre eux ? [/2]
viii. (BONUS) Nous sommes rendus à la fin des TPs du LOG1000. Comment avez vous trouvez les TPs ? et d'après vous quels sont les changements qu’on doit apporter aux TPs ? [/2]

ATTENTION

Légende utilisée au cours de ce laboratoire :

- **Le texte en gras** représente des éléments de cours qui doivent être compris pour répondre aux exercices
- Les lignes précédées d’une lettre minuscule (1.) représentent la progression conseillée dans l’exercice
- Le texte en vert représente les questions auxquelles vous devrez répondre textuellement dans vos rapports
- Le texte en rouge représente des consignes à suivre pour assurer le bon fonctionnement des exercices. **Un non-respect de ces consignes entraînera des pertes de points sévères**.

- Le texte bleu souligné représente des liens vers les ressources disponibles sur Moodle. Il suffit de Ctrl+clic sur ce texte pour y accéder.

Rédaction du rapport :

- Votre rapport sera un **DOCUMENT PDF** contenant les captures d'écran et les réponses aux questions demandées.
- Le rapport sera remis dans un dossier nommé TP5 dans votre répertoire Git. N'oubliez pas de vérifier après la remise si votre rapport est bel et bien visible sur le serveur et pas seulement sur votre copie locale.(git ls-files)
- Jusqu'à **10%** de la note peut être enlevé pour la qualité du français, et la présentation du rapport.
-
-
- **DATE DE REMISE :**
- **Groupe 2 : 27 novembre 2017 à 12h30**
- **Groupe 1 : 4 décembre 2017 à 12h30**
-