École Polytechnique de Montréal

Département de génie informatique et génie logiciel

LOG1000 – Ingénierie logicielle

Travail Pratique 5 : Restructuration de code source

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.
- **Le rapport sera remis dans un dossier nommé TP5 dans votre répertoire Git le 2 décembre avant 23h55. Il y aura une pénalité de 10% par jour de retard.**

Énoncé :

Dans cet énoncé, vous devez identifier et restructurer les mauvaises odeurs présentes dans un programme C++. Pour rappel, la restructuration de code permet de rendre celui-ci plus maintenable, plus fiable à long terme et plus lisible.

Vous avez commencé votre stage entrepreneurial d'été et vous travaillez toujours sur la même plateforme mobile permettant à des infirmières de prendre des réservations, afin de donner des soins à domicile privées. Ce matin, vous recevez un courriel qui explique que vous avez l'occasion de présenter votre jeune entreprise à des investisseurs potentiels cet après-midi. Il s'agit d'une grande opportunité pour vous! Le hic, c'est qu'il est obligatoire de présenter quelque chose lors de cette présentation et que vous n'avez toujours rien de démontrable entre vos mains. Vous prenez donc votre matin pour concevoir une preuve de concept qui vous permettra de démontrer la fonctionnalité de prise de rendez-vous. Ce faisant, puisque vous êtes pressé, vous codez très rapidement sans vous soucier de la lisibilité de votre code et de sa maintenabilité future. Aussi, vous ne faites aucune forme de validation des données. Après tout, ce n'est qu'une preuve de concept!

Votre présentation s'est très bien passé et vos auditeurs sont enthousiastes. Tellement, que plusieurs d'entre eux vous ont demandé à ce que vous développiez votre prototype, afin d'y inclure les autres fonctionnalitées que vous leurs avez promis. De retour dans l'éditeur de code, vous trouvez que ce vous avez écrit n'est pas facilement compréhensible et qu'il sera difficile d'intégrer vos nouvelles fonctionnalités à votre prototype. Vous entreprenez donc de restructurer les odeurs graves qui sont présentes dans votre code. Bien sûr, vous devrez modifier la structure interne du logiciel pour le rendre plus facile à comprendre et à modifier, évidemment SANS changer son comportement observable (par exemple, les tests doivent toujours être validés)!

- Dans le logiciel, la fonction `int main(int argc, char **argv` permet de proposer à un utilisateur les opérations possibles qu'il peut effectuer (Consulter les disponibilités d'un infirmier, Prendre rendez-vous avec un infirmier, Consulter ses rendez-vous).
- La classe Nurses regroupe le code qui est responsable de la gestion et de l'affichage des infirmiers. La classe Appointment regroupe le code qui est responsable de la gestion et de l'affichage des rendez-vous. Les fichiers Utils.hpp et Utils.cpp regroupent des structures de données et des fonctions utiles.
- Sous le dossier *data/* se trouve les différents fichiers qui contiennent les patients de votre plateforme, les infirmier et les rendez-vous enregistrés, soit respectivement patients.csv, nurses.csv et appointments.csv.

Pour vous aider dans votre restructuration de code aidez vous des diapositives du cours et/ou du lien suivant : https://refactoring.guru/

E1) Une mauvaise odeur dans les attributs [/6]

En examinant la classe « AppointmentsData », vous pouvez remarquer que la pertinence de celle-ci n'est pas très grande, puisqu'elle ne fait pas grand chose. Vous remarquez aussi que celle-ci est un attribut de la classe « Appointments » qui permet de gérer les informations afférentes aux rendez-vous.

1) Identifier le/les odeur(s) qui se cache(nt) derrière cela et expliquez pourquoi c'est grave. **[/0.5]**
2) Identifiez le/les nom.s des restructurations nécessaires pour enlever cette odeur du code. **[/0.5]**
3) Identifiez les étapes que vous allez suivre pour restructurer cette odeur. Utilisez le même format du tableau ci-dessous, dans lequel vous décomposez la restructuration globale en étapes plus simples. **[/1]**

Étape	Description

4) Restructurez le code source et copiez la/les classe(s) modifiée(s) dans le rapport (le header et le cpp files). **[/2]**
5) Compilez, essayez de créer un nouveau rendez-vous et essayez de voir si ce nouveau rendez-vous est dans la liste de rendez-vous. Veuillez ajouter des captures d'écrans des résultats de ces tests fonctionnels dans le rapport, afin de prouver la validité de vos modifications. **[/1]**
6) 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 »). **[/1]**

E2) Une mauvaise odeur dans les méthodes [/6]

En examinant la méthode « void Appointments::display() », vous vous rendez compte que cette méthode fait, l'un à la suite de l'autre, beaucoup de choses différentes et mériterait d'être un peu raccourci, afin de gagner en lisibilité.

1) Identifier le/les odeur(s) qui se cache(nt) derrière cela et expliquez pourquoi c'est grave. **[/0.5]**
2) Identifiez le/les nom.s des restructurations nécessaires pour enlever ce(s) odeur(s) du code. **[/0.5]**
3) Planifiez, étape par étape, comment restructurer cette odeur, dans le même format du tableau de l'exercice E1. **[/1]**
4) 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. **[/2]**

5) Compilez et exécutez votre logiciel. Veuillez ajouter des captures d'écrans dans le rapport qui prouve le bon fonctionnement du tout. **[/1]**
6) Faites un commit de votre code source, et une capture d'écran de ce commit dans le rapport. **[/1]**

E3) Utilisation des variables [/6]

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 « choice » et « loggedInUser » dans la méthode « main » dans « main.cpp ». Ne comptez pas les lignes vides. Les accolades et les commentaires comptent comme des lignes. **[/0.5]**
2) Interprétez les résultats, et trouvez la variable (parmi les deux citées en dessus) qui bénéficiera le plus de la restructuration. **[/0.5]**
3) Proposez des restructurations de votre fonction main pour améliorer l'utilisation de cette variable, en utilisant le même format du tableau de l'exercice E1. Notez qu'il faut tout autant minimiser le span, que la durée de vie et la portée. **[/1]**
4) Effectuez cette restructuration dans la méthode « main ». Faites une capture d'écran (ou copiez le code) de votre nouveau code source. **[/2]**
5) Compilez et testez manuellement les opérations (de l'opération 0 à 3) de la méthode « main », veuillez prendre des captures d'écrans de vos tests. **[/1]**
6) Faites un commit de votre code source, et une capture d'écran de ce commit dans le rapport. **[/1]**

E4) Tests [/2]

À la remise, assurez vous que les commandes *make* et *make test* fonctionnent correctement. Assurez-vous que tous les **tests unitaires** soient valides et effectuez des **tests fonctionnels** sur le programme, afin de vous assurer que vous n'ayez introduit aucune régression logicielle. Il n'est pas nécessaire de fournir de capture d'écran, mais le tout sera testé lors de la correction et devra être fonctionnel.

BONUS) Repérez une mauvaise odeur importante qui n'a pas été répertorié ci-haut [+1]

Essayez de trouver une mauvaise odeure grave n'importe où dans le code source et indiquez le nom de la restructuration nécessaire qui permettrait d'obtenir un code plus sain (sans toutefois restructurer le code). Veillez à trouver une mauvaise odeure qui est grave!