École Polytechnique de Montréal
Département de génie informatique et génie logiciel
LOG1000 – Ingénierie logicielle

Travail pratique #5

Hiver 2020

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 14 avril avant 12h00. Il y aura une pénalité de 10% par jour de retard.**
- Pour effectuer la remise sur git, veuillez indiquer un « tag » sur votre commit final, afin de bien identifier quel commit correspond à la « version finale » de votre travail. Votre tag doit être « TP5 ». N'oubliez pas de push votre tag avec **git push --follow-tags** .

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 fonctionnalités que vous leur 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)!

- Le logiciel vous 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 [/5]

La classe « Appointments » permet de gérer les informations afférentes aux différents rendez-vous. Elle possède un attribut de type « AppointmentsData ».

Examinez la classe « AppointmentsData ».

1) Identifiez le/les nom(s) des odeurs qui se cachent derrière celle-ci et expliquez pourquoi elles sont graves. **[/0.5]**

2) Identifiez le/les nom(s) des restructurations nécessaires pour enlever ce/ces odeur(s) du code. **[/0.5]**

3) Identifiez les étapes que vous allez suivre pour restructurer ce/ces odeur(s). Utilisez le même format que le tableau ci-dessous, dans lequel vous devez décomposer la restructuration globale en étapes plus simples. **[/1]**

Étape	Description

4) Restructurez le code source et expliquez le raisonnement de vos changements dans le rapport. N'oubliez pas de soumettre les modifications sur Git. **[/2]**

5) Compilez et essayez de créer un nouveau rendez-vous. Assurez-vous que ce nouveau rendez-vous soit dans la liste des 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]**

E2) Une mauvaise odeur dans les classes [/4]

Dans la structure actuelle, un « rendez-vous » est représenté comme étant un vecteur de *string*. La signification associée a chaque index de ce vecteur est définie dans le fichier Constants.hpp.

1) Est-ce la meilleure façon de définir un rendez-vous ? Justifiez votre réponse. **[/1.5]**

2) Proposez, sans coder, une restructuration et justifier en quoi celle-ci est pertinente. Cette restructuration peut-elle s'appliquer dans d'autres fichiers ? **[/1.5]**

3) Quel effet aura cette restructuration sur le fichier « Constant.hpp ». **[1]**

E3) Une mauvaise odeur dans les méthodes [/9]

a) Examinez la methode « void Appointments::display() » **/5**

1) Identifier le/les nom(s) du/des odeurs qui se cache(nt) derrière celle-ci et expliquez en quoi ce sont des odeurs graves. **[/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 ces odeurs, dans le même format que le tableau de l'exercice E1. **[/1]**
4) Restructurez le code source de cette méthode et expliquez le raisonnement de vos changements dans le rapport. Mentionnez si vous avez créé des nouvelles méthodes ou modifié des méthodes existantes autre que display(). N'oubliez pas de soumettre les modifications sur Git. **[/2]**
5) Compilez et exécutez votre logiciel. Veuillez ajouter des captures d'écrans des résultats de tests fonctionnels dans le rapport qui prouve la validité des modifications que vous avez effectués. **[/1]**

b) En examinant la méthode « int Appointments::getHoursRate()», vous remarquez probablement certaines odeurs dans le code. **/4**

1) Identifier les odeurs **[/1]**
2) Restructurez le code source de cette méthode et expliquez le raisonnement de vos changements dans le rapport. Mentionnez si vous avez créé des nouvelles méthodes ou modifié des méthodes existantes autre que getHoursRate(). N'oubliez pas de soumettre les modifications sur Git. **[/2]**
3) Compilez et exécutez votre logiciel. Veuillez ajouter des captures d'écrans des résultats de tests fonctionnels dans le rapport qui prouve la validité des modifications que vous avez effectués. **[/1]**

E3) Utilisation des variables [/5]

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 moyen, 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. Justifiez **[/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 » et recalculez les nouvelles métriques. N'oubliez pas de soumettre les modifications sur Git.. **[/2]**
5) Compilez et testez manuellement les opérations (de l'opération 0 à 3) de la méthode « main ». Veuillez ajouter des captures d'écrans de vos tests fonctionnels 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.