# Bartosz Ratajczyk, 238194

Refaktoryzacja Gilded Rose w języku Java.

### 1.Testy

Po pierwsze, napisałem testy, pokrywające 100% refaktoryzowanego kodu, by za pomocą re-testów, upewniać się, że program nadal działa poprawnie i dodatkowo umożliwiające sprawdzanie gdzie są nieużywane bloki programu(np. if'y, które są zawsze fałszywe)

#### 2. Ocena

Czy Gilded rose potrzebuje refaktoryzacji?

Czysty kod powinien spełniać kilka wymagań:

- 1.Być czytelny- Dobrze napisany kod nie potrzebuje komentarzy. Same nazwy zmiennych czy funkcji powinny być jednoznaczne i oczywiste, nawet dla kogoś, kto nigdy wcześniej nie widział danego kodu.
- 2. Nie powinien zawierać duplikacji- Dobrze napisany kod powinien być prosty do zaktualizowania, wprowadzane zmiany w zduplikowanych mechanikach są żmudne i trudne.
- 3. Być krótki i możliwie elementarny- w dobrym kodzie nie powinno być "Klasy- Boga" ani "Funkcji-Boga", czyli takiej klasy lub funkcji, która zajmuje się wieloma mechanikami na raz.
- 4. Powinien przechodzić testy i spełniać założenia- Co z tego, że mamy piękny kod, który nie robi tego, czego od niego oczekujemy.

Gilded Rose dumnie pluje w twarz wszystkim powyższym wymaganiom. Kod jest pełen niezrozumiałych i nieczytelnych zagnieżdżeń w których bardzo trudno się połapać i nie do końca wiadomo, który fragment kodu, za co odpowiada. Zawiera kilka niepotrzebnych duplikacji. Posiada funkcję-Boga, która jest niepotrzebnie przydługawa. Oraz (co najgorsze) nie posiada żadnych testów.

Reasumując Gilded rose, jest śmerdzącym kodem, który z pewnością na proces refaktoryzacji zasługuje.

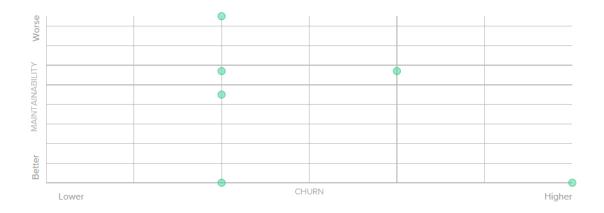
Przed refaktoryzacją kod był oceniany bardzo nisko (codacy, code climate. bettercode), przed refaktoryzacją było 5 code smells, po już tylko dwa (Cognitive Complexity of 6 i 6, gdzie dopuszczalne jest 5, ale nie miałem pomysłu, w jaki sposób bardziej to uprościć).

Plik Gilded Rose w CodeClimate miał <u>Maintainability</u> ocenione na F, po refaktoryzacji ocena zmieniła się na A

В	src/main/java/main/java/com/gildedrose/SulfurasItem.java			
В	src/Test/java/test/java/com/gildedrose/TexttestFixture.java			
A	src/main/java/main/java/com/gildedrose/BackstageItem.java			
A	src/main/java/main/java/com/gildedrose/Item.java			
A	.idea/libraries/Mavenjunit_junit_4_12.xml			
A	.idea/encodings.xml			
A	src/main/java/main/java/com/gildedrose/GildedRose.java			
A	.idea//Mavencom_approvaltests_approvaltests_4_0_2.xml			
A	.idea/vcs.xml			
A	.idea//Mavencom_approvaltests_approvaltests_util_4_0_2.xml			
src/main/java//java/com/gildedrose/ConjuredManaCakeItem.java				
src/ma	in/java/main/java/com/gildedrose/AgedBrieltem.java	18	A 55 mins	
src/main/java/main/java/com/gildedrose/BackstageItem.java		26	A 1 hr	
src/main/java/main/java/com/gildedrose/ConjuredManaCakeltem.java		23	A 45 mins	
src/main/java/main/java/com/gildedrose/GildedRose.java		12	A 0 mins	
src/main/java/main/java/com/gildedrose/ltem.java		42	A 55 mins	
src/main/java/main/java/com/gildedrose/SulfurasItem.java		9	A 0 mins	

# Churn vs. maintainability

Maintainability issues cause bigger problems in files that are changed (churn) frequently.



## 3. Przebieg refaktoryzacji

- -napisanie testów i zaznaczenie pokrycia kodu
- -sprzątanie zagnieżdżonych if'ów za pomocą wyciągania poszczególnych warunków, w zależności od poszczególnego przedmiotu.(wszystkie warunki dot. Sulfuras. Zostały wyciągnięte do wspólnego przypadku (if(name==sulfuras)...) ).
- -przeniesienie updateQuality do klasy Item (w celu późniejszego zastosowania polimorfizmu)
- -użycie polimorfizmu, stworzenie podklas Item, definiujących typ obiektu i nadpisujących metodę update quality
- -zadeklarowanie zachowań przedmiotów przy podczas mijania czasu bezpośrednio w podklasach
- -Dodanie żądanej funkcjonalności w dodatkowej podklasie.(conjured mana cake).