Gilded Rose

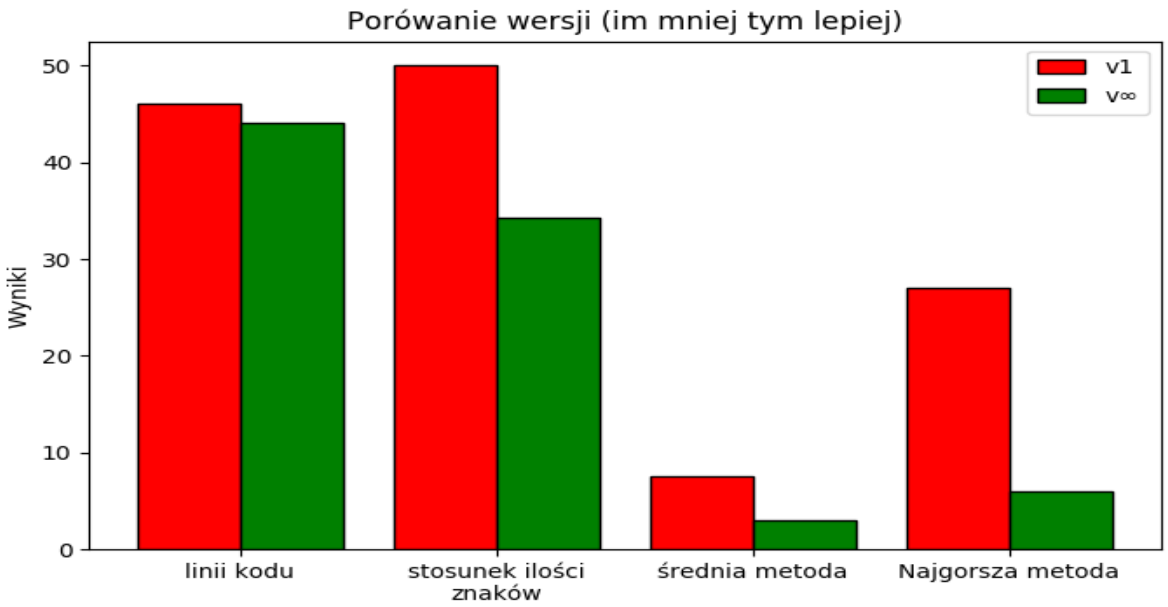
1. Gilded rose v1 vs Gilded rose v∞

Gilded rose v1 jest kodem otrzymanym od Leeroya i zmodyfikowanym przez nas, natomiast Gilded rose v∞ jest wersją do której dążymy, chcąc aby kod był czytelny i łatwo zrozumiały. Pierwsza wersja programu posiada 46 linijek kodu, a ostatnia 44. Jeśli jednak policzymy ilość znaków bez spacji, to zmniejszyła się ona z 1323 do 906, co daje spadek o 31,5%. Dla wnikliwszej analizy przyjrzymy się wynikom złożoności cyklomatycznej otrzymanej programem Radon.

```
Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
maciej@maciej-Lenovo-Y50-70:~/git/GR$ radon cc -s gilded_rose_v1.py
gilded_rose_v1.py
    M 8:4 GildedRose.update_quality - D (27)
    C 3:0 GildedRose - C (15)
    C 50:0 Item - A (2)
    M 5:4 GildedRose.__init__ - A (1)
    M 51:4 Item.__init__ - A (1)
    M 56:4 Item.__repr__ - A (1)
```

```
Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
maciej@maciej-Lenovo-Y50-70:~/git/GR$ radon cc -s gilded_rose_vinf.py
gilded_rose_vinf.py
    M 6:4 GildedRose.update_quality - B (6)
    C 1:0 GildedRose - A (5)
    M 40:4 Item.concert_update - A (5)
    C 20:0 Item - A (4)
    M 29:4 Item.update_item - A (4)
    M 3:4 GildedRose.__init__ - A (1)
    M 21:4 Item.__init__ - A (1)
    M 26:4 Item.__repr__ - A (1)
```

Na pierwszy rzut oka w naszej wersji widać dwie dodatkowe metody w klasie Item aktualizujące pojedyncze obiekty, lecz nie są one mocno skomplikowane i obydwie dostały ocenę „A" z wynikami 4 pkt i 5 pkt. Największą zmianę widać w metodzie „GildedRose.update_quality()". Zostały z niej usunięte obliczenia i przeniesione do wyżej wymienionych metod. Ułatwia to znacznie wprowadzanie do sprzedaży nowych przedmiotów, nawet z innymi właściwościami. Skutkiem tego jest duży spadek złożoności tej metody z oceny „D" i 27 pkt do oceny „B" i 6 pkt, oraz całej klasy „GildedRose" z oceny „C" i 15 pkt do oceny „A" i 5 pkt.



Maciej Kieżel
GitHub: mkiezel/Gilded_rose

2. Refaktoryzacja

2.1 v1 → v3

Refaktoryzacja zaczęła się od usunięcia powielonego kodu. Usunęliśmy warunki minimalnego i maksymalnego „item.quality", które były zwielokrotnione w kodzie, a dodaliśmy je na końcu. Dzięki temu zabiegowi zmniejszyliśmy znacząco ilość warunków if oraz maksymalny stopień ich zagnieżdżenia zmalał o jeden. Biorąc pod uwagę, że „Sulfuras" jest itemem legendarnym na początku metody umieściliśmy linijkę omijającą jedno przejście pętli gdy chodzi o ten przedmiot. Umożliwiło to usunięcie wszyskich warunków związanych z tym przedmiotem, który nie zmienia swoich wartości „sell_in" i „quality". Na koniec usunęliśmy niepotrzebne zagnieżdżenie dotyczące koncertów.

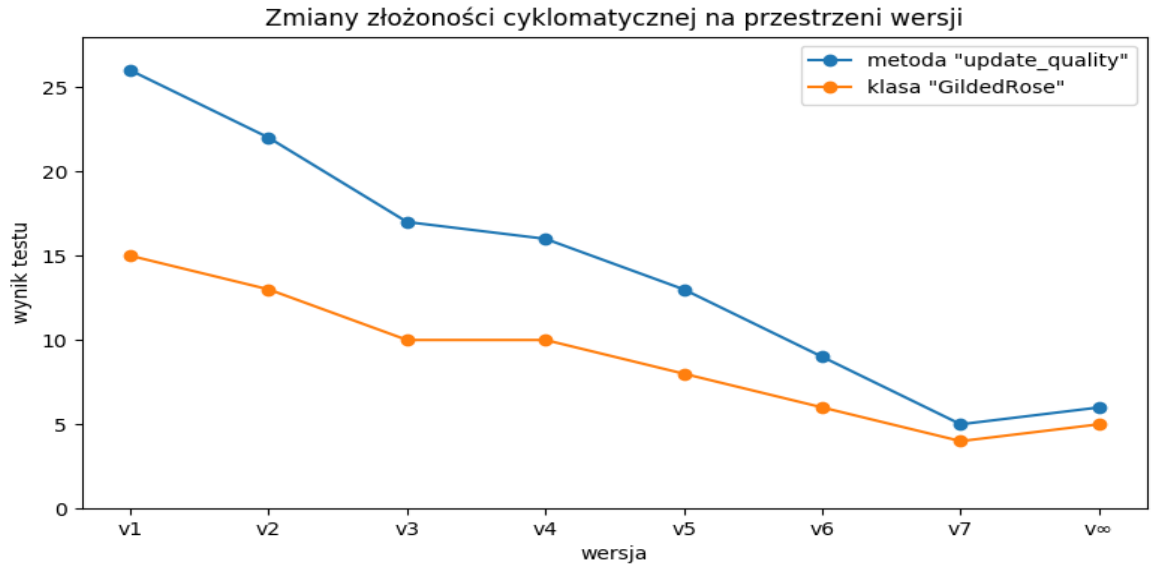
Dzięki pracy nad kodem złożoność dla metody „GildedRose.update_quality()" zmalała z 26 pkt do 17 pkt co pozwoliło zmienić jej ocenę z „D" na „C". Podobnie zachowała się klasa „GildedRose" której wynik z początkowych 15 pkt obniżył się do 10 pkt, czyli maksymalnej wartości dla oceny „B".

2.2 v3 → v4

Ciekawy zabieg wykonaliśmy w przejściu z wersji v3 do v4. Krok ten obejmował zmianę w kodzie wszystkich warunków przeczących (!=) na warunki równości (==). Nie zmieniło to złożoności cyklomatycznej klasy, a złożoność metody spadła tylko o 1 ptk, jednak kod stał się znacznie bardziej czytelny.

2.3 v4 → v7

W kolejnych wersjach program zaczął stawać się bardziej obiektowy. Dodaliśmy metody aktualizujące koncert i sprawdzające granice „quality". Jako ostatnią zmianę dodaliśmy metodę, która umożliwiła usunąć zdublowaną część kodu odpowiadającą za „przeterminowanie" produktów. Kod stał się prostszy do zrozumienia i łatwiejszy do przyszłej rozbudowy.



Maciej Kieżel
GitHub: mkiezel/Gilded_rose

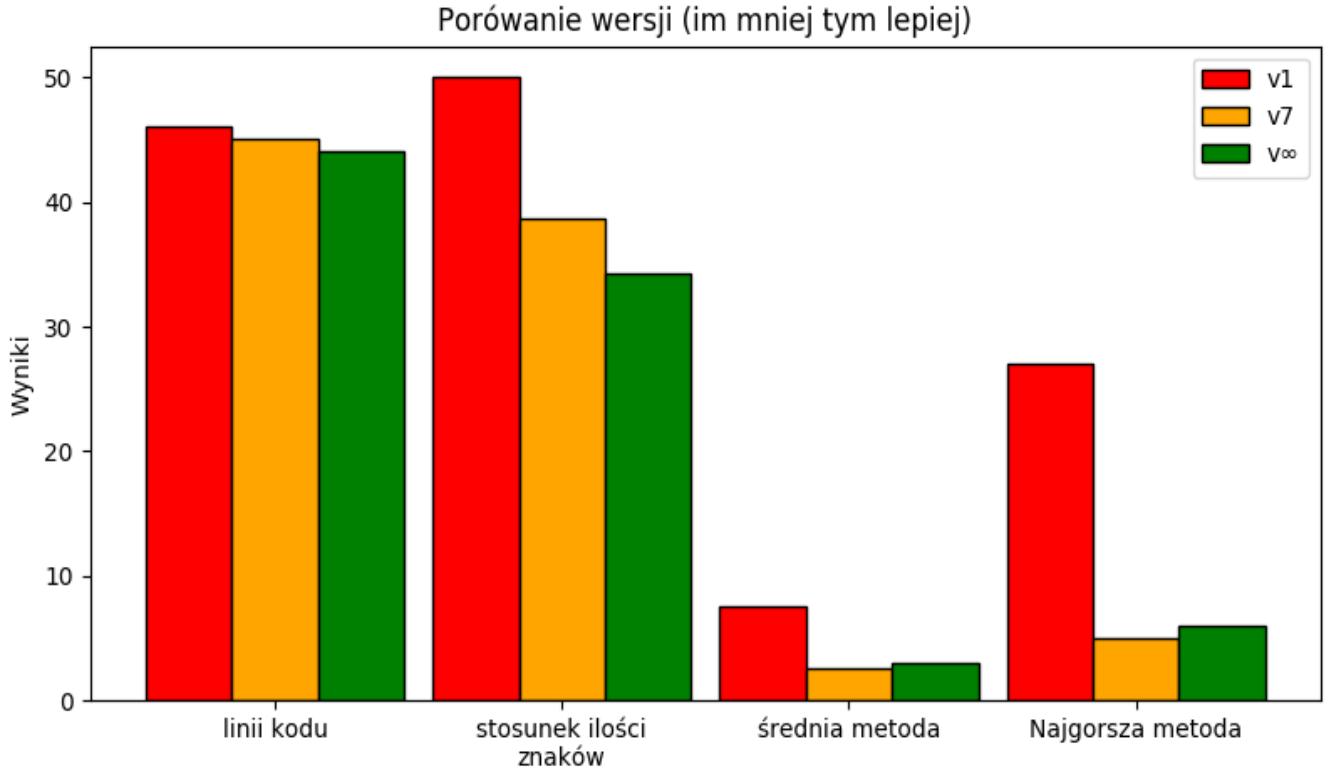
3. Testy

Jako testu użyliśmy programu napisanego w pythonie, który za każdym razem importował dwie wersje kodu. Wersję Leeroya i drugą, podaną przez użytkownika. Następnie program porównywał wyniki dla wszystkich dostępnych przedmiotów przy różnych ilościach dni. Wszystkie wersje od v1 do v7, razem z wersją v∞ przeszły wszystkie testy.

4. Podsumowanie

Podczas refaktoryzacji udało nam się zmniejszyć złożoność klasy „GildedRose" do złożoności niższej niż w wersji ustalonej jako v∞. Metoda „update_quality" uzyskała wynik 4 pkt, czyli o 1 pkt mniej niż wersja ustalona za najlepszą. Biorąc pod uwagę średni wynik metody wersja v7 również wyprzedza wersję v∞, jednak spowodowane jest to większą ilością metod. Łatwość dodawania do programu kolejnych pozycji znacznie spadła, tak samo jak ułatwiona została możliwość przyszłego rozwoju programu.

Uważam jednak, że wersja v∞ wygrywa konfrontację z naszym kodem po refaktoryzacji. W wersji „idealnej" mamy mniej metod i są one bardziej intuicyjne w użytku. Różnica w ilości kodu jest również na korzyść v∞. Za największy atut w tym wypadku uznaję możliwość dodawania przedmiotów po zapoznaniu się tylko z metodą „update_quality()" , a nie całym kodem.



Maciej Kieżel
GitHub: mkiezel/Gilded_rose