Refaktoryzacje - Michał Kunda

link: https://github.com/Shanger1/Gilded-Rose

Informacje: przy tworzeniu projektu, wzorowałem się na podstawie kroków z filmu: https://www.youtube.com/watch?v=HMvue3TMgSk, przy czym metody refaktoryzacji zostały nazwane i przedstawione w dalszej części dokumentacji. Inne źródło informacji: https://www.samouczekprogramisty.pl/solid-czyli-dobre-praktyki-w-programowaniu-obiektowym/

1. Dodanie obsługi Conjured Item

```
if (!items[i].name.equals("Aged Brie")
                    && !items[i].name.equals("Backstage passes to a
TAFKAL80ETC concert")) {
                if (items[i].quality > 0) {
                    if (!items[i].name.equals("Sulfuras, Hand of
Ragnaros")) {
                        if(!items[i].name.equals("Conjured Mana Cake")){
                            items[i].quality = items[i].quality - 1;
                        } else {
                            items[i].quality = items[i].quality - 2;
                        }
                    }
                }
            }
```

Manualny test po kompilacji programu. Obsługa przedmiotu Conjured została wykonana poprawnie

2. Zmiana pętli, wydzielenie metody głównej.

Następnie zmieniłem pętle po indeksie na pętle foreach, w której wywoływana jest metoda oddzielnie dla każdego przedmiotu.

```
public void updateQuality() {
        for (Item item : items) {
            updateItem(item);
        }
    }

    public void updateItem(Item item){
        if (!item.name.equals("Aged Brie")
                && !item.name.equals("Backstage passes to a TAFKAL80ETC
```

```
concert")) {
            if (item.quality > 0) {
                if (!item.name.equals("Sulfuras, Hand of Ragnaros")) {
                    if (!item.name.equals("Conjured Mana Cake")) {
                        item.quality = item.quality - 1;
                    } else {
                        item.quality = item.quality - 2;
                    }
                }
            }
        } else {
            if (item.quality < 50) {
                item.quality = item.quality + 1;

                if (item.name.equals("Backstage passes to a TAFKAL80ETC
concert")) {
                    if (item.sellIn < 11) {
                        if (item.quality < 50) {
                            item.quality = item.quality + 1;
                        }
                    }

                    if (item.sellIn < 6) {
                        if (item.quality < 50) {
                            item.quality = item.quality + 1;
                        }
                    }
                }
            }
        }

        if (!item.name.equals("Sulfuras, Hand of Ragnaros")) {
            item.sellIn = item.sellIn - 1;
        }

        if (item.sellIn < 0) {
            if (!item.name.equals("Aged Brie")) {
                if (!item.name.equals("Backstage passes to a TAFKAL80ETC
concert")) {
                    if (item.quality > 0) {
                        if (!item.name.equals("Sulfuras, Hand of
Ragnaros")) {
                            item.quality = item.quality - 1;
                        }
                    }
                } else {
                    item.quality = item.quality - item.quality;
                }
            } else {
                if (item.quality < 50) {
                    item.quality = item.quality + 1;
                }
            }
```

```
        }
    }
```

Program został uruchomiony, logika działania nie została zmieniana więc wynik zgadza się z tym poprawnym.

3. Dodanie testów na wszystkie skrajne przypadki.

```
@Test
    public void test_quality_never_negative() {
        Item[] items = new Item[] { new Item("Elixir of the Mongoose", 0,
0) };
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertTrue(app.items[0].quality >= 0);
    }

    @Test
    public void test_quality_never_more_than_50() {
        Item[] items = new Item[] {
                new Item("Aged Brie", 5, 50), //
                new Item("Elixir of the Mongoose", 5, 50), //
                new Item("Backstage passes to a TAFKAL80ETC concert", 5,
50),
                new Item("Conjured Mana Cake", 5, 50) };
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertTrue(app.items[0].quality <= 50);
        assertTrue(app.items[1].quality <= 50);
        assertTrue(app.items[2].quality <= 50);
        assertTrue(app.items[3].quality <= 50);
    }

    @Test
    public void test_quality_downgrade_and_twice_as_fast_downgrade(){
        Item[] items = new Item[] { new Item("Elixir of the Mongoose", 1,
5) };
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertEquals(4, app.items[0].quality);
        app.updateQuality();
        assertEquals(2, app.items[0].quality);
    }

    @Test
    public void test_aged_brie_quality_increase() {
        Item[] items = new Item[] { new Item("Aged Brie", 5, 5) };
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertEquals(6, app.items[0].quality);
        app.updateQuality();
        assertEquals(7, app.items[0].quality);
```

```
    }

    @Test
    public void test_sulfuras() {
        Item[] items = new Item[] { new Item("Sulfuras, Hand of Ragnaros",
5, 5) };
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertEquals(5, app.items[0].quality);
        app.updateQuality();
        assertEquals(5, app.items[0].quality);
    }

    @Test
    public void test_special_event() {
        Item[] items = new Item[] { new Item("Backstage passes to a
TAFKAL80ETC concert", 11, 3) };
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertEquals(4, app.items[0].quality);
        app.updateQuality();
        assertEquals(6, app.items[0].quality);
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        assertEquals(17, app.items[0].quality);
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        assertEquals(0, app.items[0].quality);
    }
```

Pozwoli to uniknąć błędów w kodzie przy refaktoryzacji.

4. Odwracanie if, aby unikać negatywnych warunków, upraszczanie operacji, wydzielenie metod.

Do wykonania tego kroku wykorzystałem wbudowaną pomoc w IntelliJ do odwracania i wydzielania instrukcji warunkowych oraz usuwania tzw. martwego kodu. Dzięki takim zmianom nasz kod jest o wiele czytelniejszy. Nie trzeba poświęcać zbyt wiele czasu na poznanie warunków if.

W czasie refaktoryzacji wykonywane zostały testy napisane w poprzednim punkcie.

```
public void increaseItemQuality(Item item, int value){
        if(item.quality < 50) {
            item.quality += value;
        }
    }
```

```
    public void decreaseItemQuality(Item item, int value){
        if(item.quality > 0) {
            item.quality -= value;
        }
    }

    public void updateItem(Item item){
        if (item.name.equals("Aged Brie")) {
            increaseItemQuality(item,1);
        } else if (item.name.equals("Backstage passes to a TAFKAL80ETC
concert")) {
            increaseItemQuality(item, 1);
            if (item.sellIn < 11) {
                increaseItemQuality(item, 1);

            }
            if (item.sellIn < 6) {
                increaseItemQuality(item, 1);

            }

        } else {
            if (item.quality > 0) {
                if (!item.name.equals("Sulfuras, Hand of Ragnaros")) {
                    if (item.name.equals("Conjured Mana Cake")) {
                        decreaseItemQuality(item, 2);
                    } else {
                        decreaseItemQuality(item, 1);
                    }
                }
            }
        }

        if (!item.name.equals("Sulfuras, Hand of Ragnaros")) {
            item.sellIn -= 1;
        }

        if (item.sellIn < 0) {
            if (!item.name.equals("Aged Brie")) {
                if (!item.name.equals("Backstage passes to a TAFKAL80ETC
concert")) {
                    if (item.quality > 0) {
                        if (!item.name.equals("Sulfuras, Hand of
Ragnaros")) {
                            decreaseItemQuality(item, 1);
                        }
                    }
                } else {
                    item.quality = 0;
                }
            } else {
                increaseItemQuality(item, 1);
            }
```

```
        }
    }
```

Dzięki testom udało się wykryć błąd, który popełniłem przy warunkach "Backstage passes"

5. W tym kroku chciałbym użyć zasady feature envy do umieszczenia metod w klasach i wydzielenia typów przedmiotów na oddzielne klasy.

Powstało 5 nowych klas dziedziczących po klasie Item, każda z nich ma własną implementację metody upgrade wyliczającą quality. Legendarny przedmiot wciąż tworzony jest przez klasę Item, która ma pustą implementację metody upgrade. **Przy tym kroku tworzenie przedmiotów w testach musiały zostać poprawione**. Klasa z metodą updateQuality prezentuje się teraz zupełnie inaczej, a cały program działa tak samo. W projekcie można zauważyć poprawnie wykorzystane techniki programowania obiektowego. Dzięki wprowadzonym zmianom, kod ten jest bardzo łatwo rozszerzalny. Jeśli w przyszłości byłaby potrzeba dodania nowej kategorii przedmiotu, której implementacja metody zmieniającej jakość przedmiotu będzie się różnić, wystarczy dodać nową klasę.

Główna klasa z metodą updateQuality:

```
class GildedRose {
    Item[] items;

    public GildedRose(Item[] items) {
        this.items = items;
    }

    public void updateQuality() {
        for (Item item : items) {
            item.upgrade();
        }
    }
}
```

Przykład jednego z typu przedmiotów.

```
public class SpecialEvent extends Item {

    public SpecialEvent(String name, int sellIn, int quality) {
        super(name, sellIn, quality);
    }

    public void upgrade() {
        setSellIn(getSellIn() - 1);
        increaseItemQuality();
        if(getSellIn() < 10){
            increaseItemQuality();
        }
        if(getSellIn() < 5){
```

```
            increaseItemQuality();
        }
        if(getSellIn() < 0){
            setQuality(0);
        }
    }

}
```

Po zmianach w klasie z testami kod prezentuje się w taki sposób:

```
public class GildedRoseTest {

    private Item[] items;

    @Before
    public void init_list(){
        items = new Item[] {
                new NormalItem("Elixir of the Mongoose", 0, 0),
                new NormalItem("Elixir of the Mongoose", 1, 5),
                new AgedBrie("Aged Brie", 5, 5),
                new Item("Sulfuras, Hand of Ragnaros", 5, 5),
                new SpecialEvent("Backstage passes to a TAFKAL80ETC
concert", 11, 3)
        };
    }

    @Test
    public void test_quality_never_negative() {
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertTrue(app.items[0].quality >= 0);
    }

    @Test
    public void test_quality_never_more_than_50() {
        Item[] allItems = new Item[] {
                new AgedBrie("Aged Brie", 5, 50), //
                new NormalItem("Elixir of the Mongoose", 5, 50), //
                new SpecialEvent("Backstage passes to a TAFKAL80ETC
concert", 5, 50),
                new ConjuredItem("Conjured Mana Cake", 5, 50) };
        GildedRose app = new GildedRose(allItems);
        app.updateQuality();
        assertTrue(app.items[0].quality <= 50);
        assertTrue(app.items[1].quality <= 50);
        assertTrue(app.items[2].quality <= 50);
        assertTrue(app.items[3].quality <= 50);
    }

    @Test
    public void test_quality_downgrade_and_twice_as_fast_downgrade(){
```

```
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertEquals(4, app.items[1].quality);
        app.updateQuality();
        assertEquals(2, app.items[1].quality);
    }

    @Test
    public void test_aged_brie_quality_increase() {
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertEquals(6, app.items[2].quality);
        app.updateQuality();
        assertEquals(7, app.items[2].quality);
    }

    @Test
    public void test_sulfuras() {
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertEquals(5, app.items[3].quality);
        app.updateQuality();
        assertEquals(5, app.items[3].quality);
    }

    @Test
    public void test_special_event() {
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        assertEquals(4, app.items[4].quality);
        app.updateQuality();
        assertEquals(6, app.items[4].quality);
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        assertEquals(17, app.items[4].quality);
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        app.updateQuality();
        assertEquals(0, app.items[4].quality);
    }

}
```

5. Ocena kodu.

Moje repository na GitHub zostało przeanalizowane przez program CodeClimate. Na głównym panelu możemy zauważyć podsumowanie z którego wynika, że:

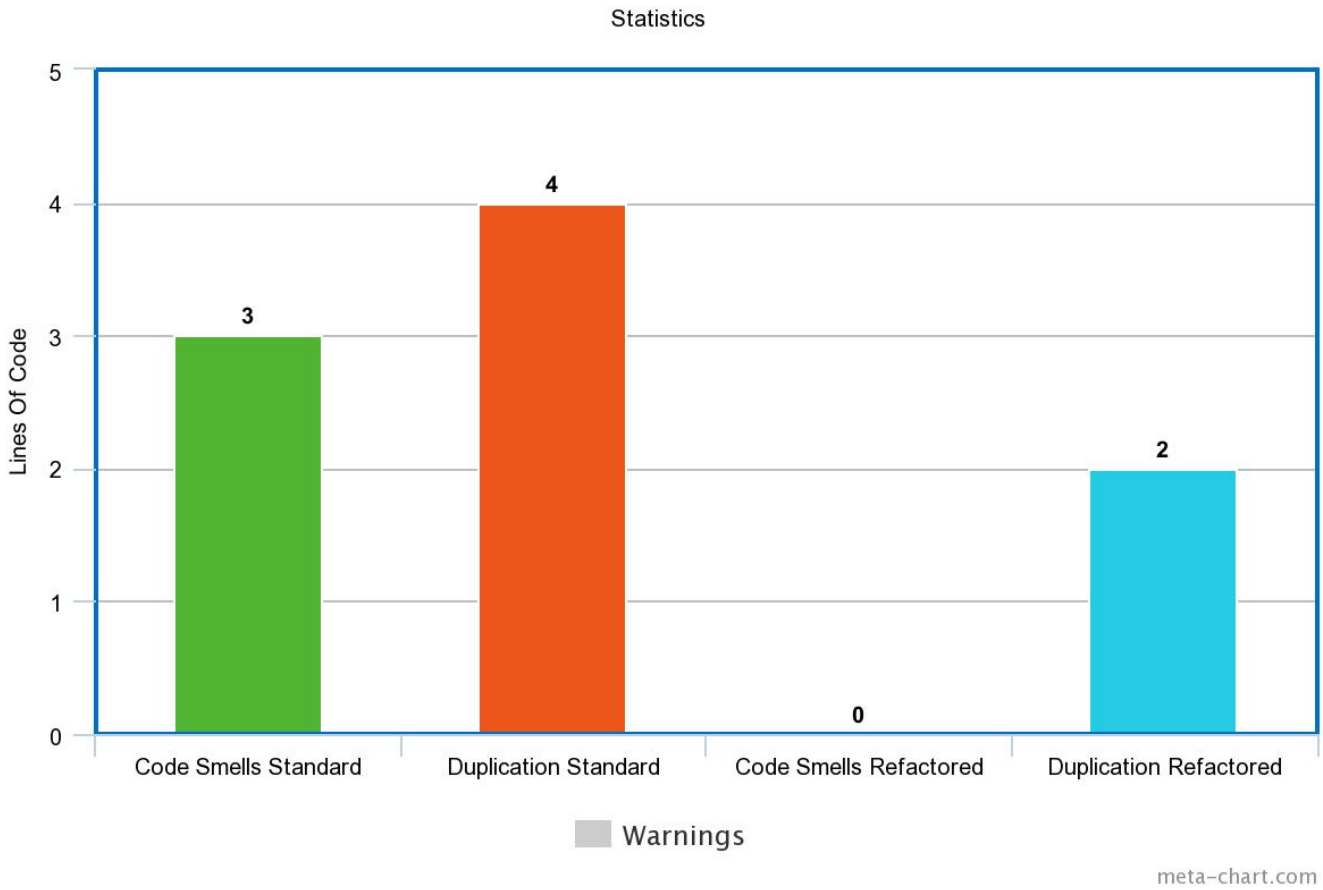
- Maintainability otrzymało ocenę: B
- Znalezionych "Code Smells": 0
- Duplikaty: 2 (są to dwie różne deklaracje konstruktora, które wykryto jako duplikat)
- Inne problemy: 0

Możemy je porównać do projektu w wersji początkowej:

- Maintainability otrzymało ocenę: F
- Znalezionych "Code Smells": 3
- Duplikaty: 4 (są to dwie różne deklaracje konstruktora, które wykryto jako duplikat)
- Inne problemy: 0

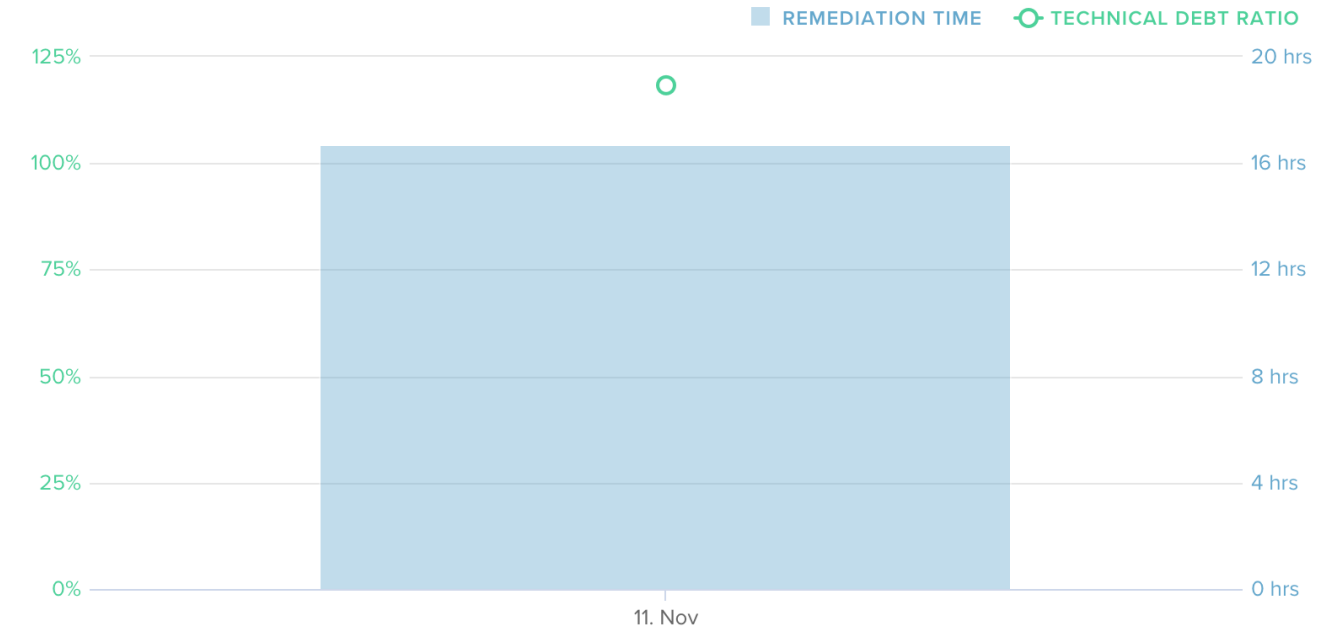
Statystyki projektów:

Porównałem problemy początkowego projektu z problemami po refaktoryzacji i utworzyłem diagram słupkowy, który porównuje code smells projektu przed i po refaktoryzacji oraz duplikacje kodu przed i po refaktoryzacji:

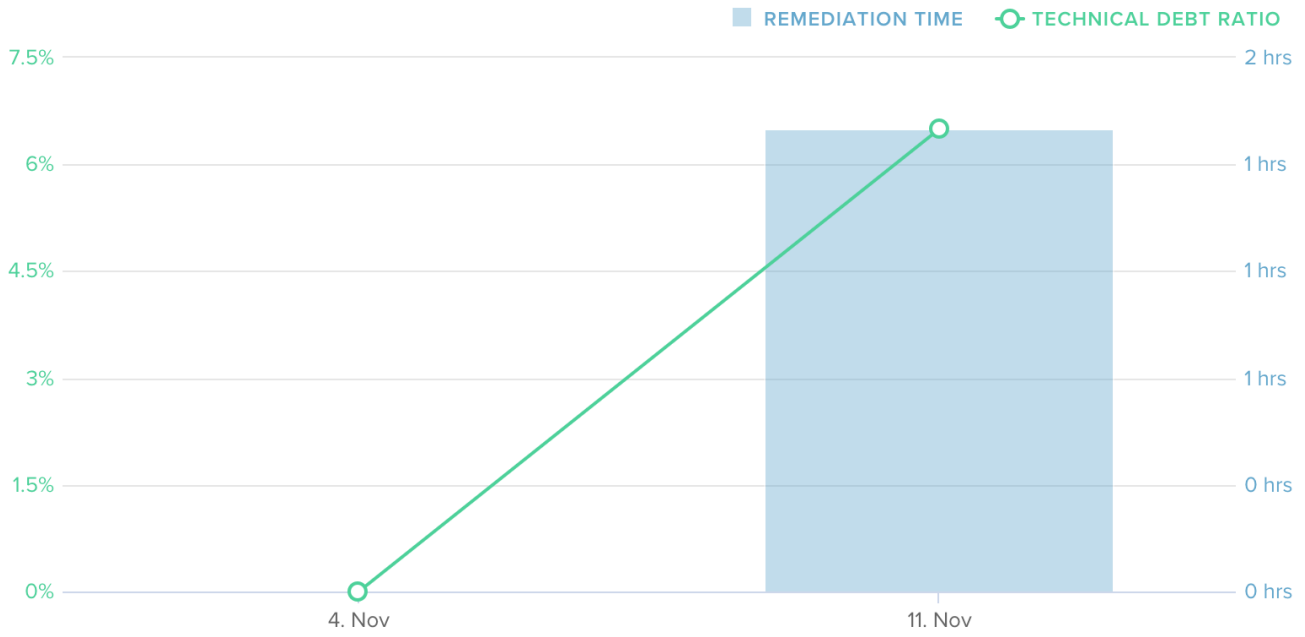


Więcej diagramów pobranych z Code Climate:

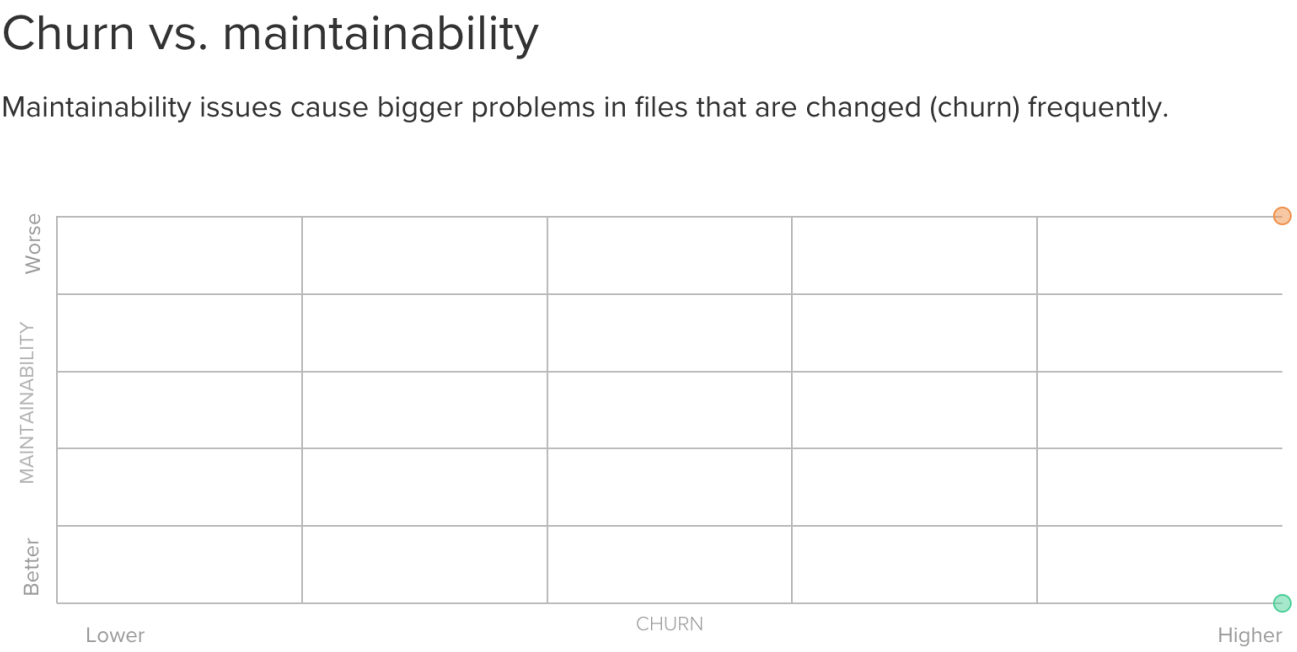
Technical Debt przed refaktoryzacją:



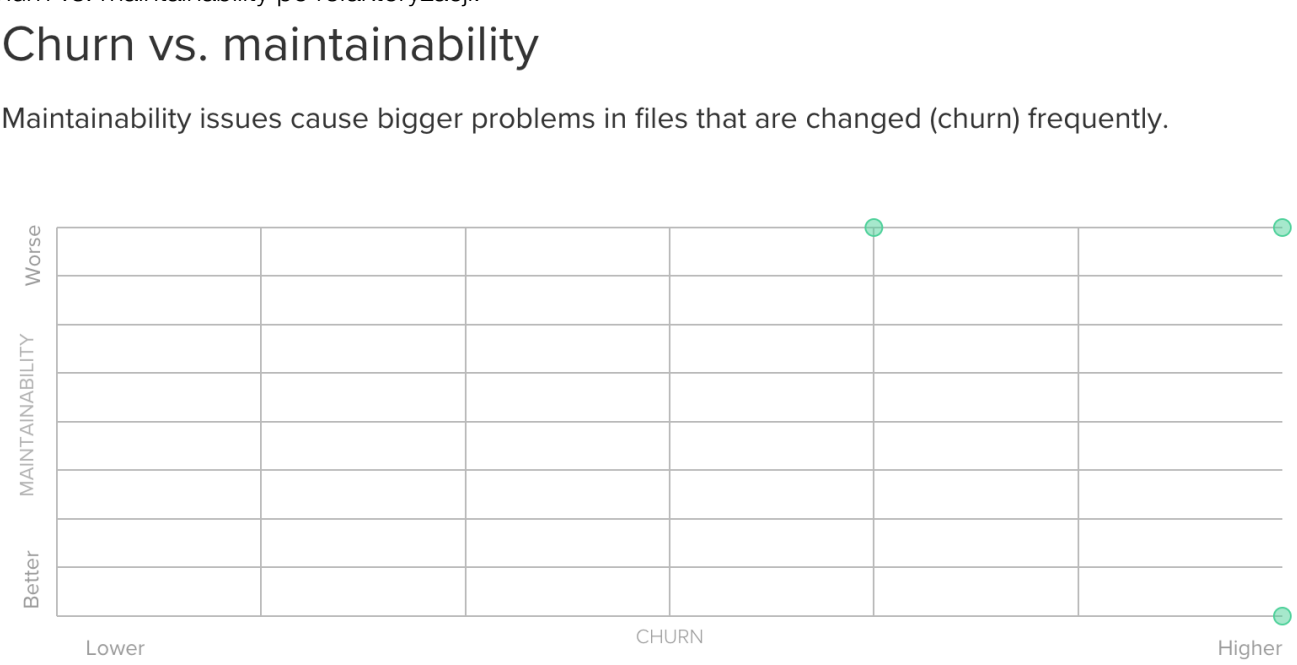
Technical Debt po refaktoryzacji:



Churn vs. maintainability przed refaktoryzacją:



Churn vs. maintainability po refaktoryzacji:



Po przeanalizowaniu wyników można stwierdzić, że refaktoryzacja kodu przebiegła pomyślnie.