Zaawansowane języki programowania. Grzegorz Dziemiński, 225112 (studia II stopnia, informatyka, zaocznie)

Refaktoryzacja na przykładzie kodu generującego piosenkę „99 bottles”, napisanego w języku Ruby.

Z racji na brak dostępu do obiektowego kodu możliwego do zrefaktoryzowania w swoim miejscu pracy, do wykonania zadania postanowiłem wykorzystać prosty program generujący piosenkę *"99 bottles"*. Został on napisany w języku Ruby, z wykorzystaniem metodyki TDD. Do stworzenia programu wykorzystano testy z książki *"99 Bottles of OOP"* autorstwa Sandi Metz. Pierwsza wersja programu, napisana przeze mnie i spełniająca wszystkie testy to v0.rb. Jako wzór do osiągnięcia posłużono się wersją "shamelessly green" z wspomnianej powyżej książki, jest ona oznaczona jako vfinal.rb. Należy zauważyć iż wersja ta nie musi być oczywiście wersją idealną i ostatecznie końcową, jak sugerowałaby nazwa "final", jednak z punktu widzenia wersji v0.rb jest jednak wersją lepszą, celem ku któremu można dążyć refaktoryzując v0.rb.

Oceniając kod obecny w wersji v0.rb oraz vfinal.rb musimy posłużyć się jakimiś wskaźnikami które pozwolą nam zdecydować która wersja jest lepsza i w jakim sensie. Na pierwszy rzut oka kod możemy ocenić odruchowo, biorąc pod uwagę to jak bardzo skomplikowany się wydaje, jak trudno go zrozumieć, albo jak dużo linii kodu zawiera lub jak dobrze jest opatrzony komentarzami. Niektóre z tych ocen są jednak subiektywne i istnieje potrzeba posłużenia się bardziej obiektywnymi wskaźnikami. Najbardziej oczywistym i popularnym zdaje się być wskaźnik SLOC (Source Lines of Code), który mówi ile linii kodu źrodłowego dany program posiada (z wyłączeniem komentarzy, pustych linii etc). Jest to prosty, jednoliczbowy wskaźnik, dzięki któremu można bardzo łatwo porównywać różne programy. Jego prostota sprawia, że posiada on jednak również wiele wad. Duża ilość linii kodu nie oznacza iż dany program jest bardziej skomplikowany, nie oznacza też koniecznie, że jest lepszy czy też, że jego wykonanie zajmuje więcej czasu i vice versa - mała ilość linii kodu nie oznacza iż dany program jest koniecznie "lżejszy" niż jego konkurent lub że jest go łatwiej zrozumieć, czasem jest wręcz przeciwnie. Wskaźnikiem SLOC należy więc posługiwać się ostrożnie.

Innym wskaźnikiem jest **ABC - Assigments, Branches and Conditions**. Sprawdza on ilość przypisywań wartości do zmiennych, odwołań do funkcji, ilość warunków w programie. Jest bardziej złożoną wersją "cyclomatic complexity" i może powiedzieć o wiele więcej niż same linie kodu źródłowego. W przypadku Ruby'ego wykorzystano oprogramowanie Flog które w dużej mierze odwzorywuje metrykę ABC. Poniżej przedstawiono porównanie wskaźników w programie w wersji v0.rb oraz vfinal.rb:

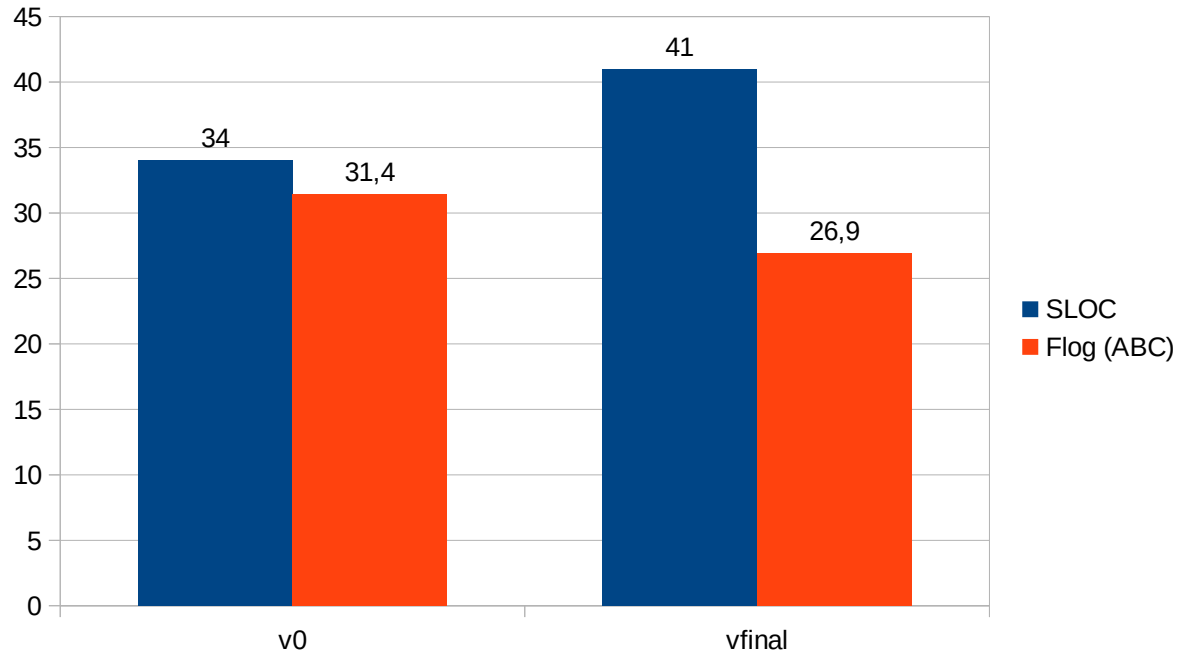
v0.rb:

```
FLOG
      31.4: flog total
      10.5: flog/method average
      25.1: Bottles#verse                    v0.rb:2-22


SLOC
      {:total=>34, :empty_lines=>3, :single_comment=>8}}
```

vfinal.rb:

```
FLOG
      26.9: flog total
      6.7: flog/method average
      19.3: BeerSong#verse                   vfinal.rb:14-38


SLOC
      {:total=>41, :empty_lines=>6, :single_comment=>3}}
```

Jak widać, wersja v0 mimo iż posiada mniej linii kodu (**34**) od wersji vfinal (**41**), to według metryki ABC (jej implementacji w postaci *Flog*), jest bardziej skomplikowana gdyż uzyskała wynik "**31.4**" w porównaniu do "**26.9**" przy vfinal.rb. Już teraz widać iż dłuższy kod nie oznacza iż jest on gorszy, trudniejszy do zrozumienia lub zbyt skomplikowany. Poniżej przedstawiono wyniki w postaci wykresu.



v1.rb

W celu refaktoryzacji kodu v0.rb, w pierwszej kolejności postanowiłem sprawdzić "**code smell**" które mogą znajdować się w v0.rb. W tym celu skorzystałem z narzędzia do Ruby'ego które pomaga je wynajdować. Narzędzie nazywa się "*reek*" (z angielskiego "smród", nazwiązanie do "smell"). Najprostszym do naprawienia i najbardziej oczywistym "smellem" była nazwa zmiennej "*i*" której nazwa nie mówi wiele, dlatego też została ona zmieniona na nazwę bardziej komunikatywną "*verse_number*", która wskazuje już iż pod zmienną kryje się numer akapitu piosenki (od 99 do 0). Inny banalny code smell to brak komentarza opisującego działanie klasy "Bottles". Ciekawszymi code smellami są te wskazane poniżej:

```
  [4, 9]:DuplicateMethodCall: Bottles#verse calls '"#{i} bottles of beer on the wall, "
+ ... "Take one down and pass it around, "' 2 times
[https://github.com/troessner/reek/blob/v5.5.0/docs/Duplicate-Method-Call.md]

  [4, 9]:DuplicateMethodCall: Bottles#verse calls '"#{i} bottles of beer on the wall, "
+; "#{i} bottles of beer.\n"' 2 times
[https://github.com/troessner/reek/blob/v5.5.0/docs/Duplicate-Method-Call.md]

  [7, 12]:DuplicateMethodCall: Bottles#verse calls 'i - 1' 2 times
[https://github.com/troessner/reek/blob/v5.5.0/docs/Duplicate-Method-Call.md]
```

Informują one o duplikacji jakiegoś fragmentu kodu i sugerują, by jej zapobiec, na przykład definiując coś w kodzie jeden raz, jeśli jest to używane wielokrotnie. Jest to tak zwana zasada **DRY**, skrót od angielskiego *Don't Repeat Yourself*, czyli *Nie Powtarzaj Się*. Duplikowanie w kodzie mogą być zarówno dane, jak i określona logika działania programu. W tym wypadku chodzi jedynie o duplikację danych (linijki piosenki), dlatego też by pozbyć się tych code smelli, pewne rzeczy zdefiniowałem jako zmienne które są następnie wykorzystywane w metodzie verse. W przypadku duplikacji logiki, należałoby zdefiniować metody, funkcje, które określoną logikę zawierają i które następnie mogą być wykorzystywane wielokrotnie.

Po tak podstawowej refaktoryzacji, należałoby zweryfikować jak obecnie prezentują się wskaźniki w przypadku wersji v1.rb.

SLOC wersji v1.rb:

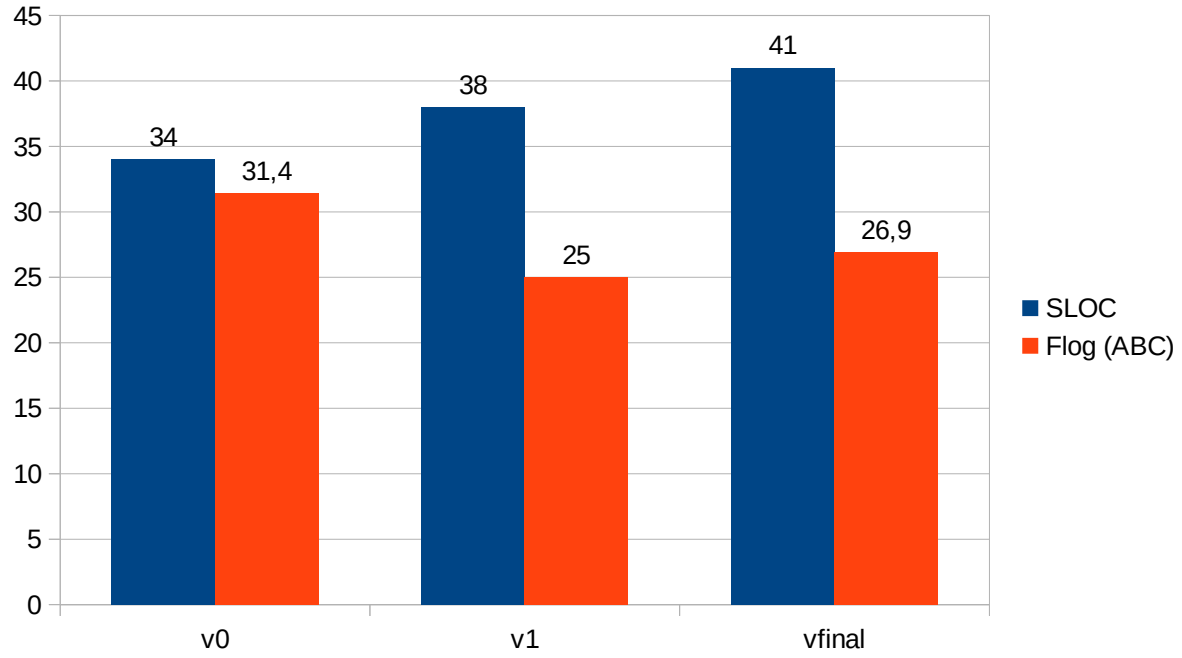
```
{:total=>38, :empty_lines=>3, :single_comment=>10}}
```

jak widać wzrosła ilość linii kodu, zbliżając nas do wartośći vfinal.rb. Korelacja linii kodu z jej lepszym wynikiem ABC w przypadku wersji v0 i vfinal sugerowałaby, że większa ilość linii zapewnie również lepszy wynik w metryce ABC, choć nie jest to pewne. Jak jednak wskazuje flog, tak się właśnie stało:

FLOG wersji v1.rb:

```
25.0: flog total
8.3: flog/method average
18.6: Bottles#verse                    v1.rb:3-26
```

Wynik we flogu przekroczył nawet wynik wersji vfinal. Wynosi on **25**, podczas gdy vfinal posiada wynik **26.9**.



Spójrzmy jednak jaka jest czytelność kodu na pierwszy rzut oka. W przypadku wersji *v0.rb* jasne jest na przykład, że piosenka posiada cztery rodzaje wersów:

```
if i > 2
  "#{i} bottles of beer on the wall, " +
  "#{i} bottles of beer.\n" +
  "Take one down and pass it around, " +
  "#{i - 1} bottles of beer on the wall.\n"
elsif i == 2
  "#{i} bottles of beer on the wall, " +
  "#{i} bottles of beer.\n" +
  "Take one down and pass it around, " +
  "#{i - 1} bottle of beer on the wall.\n"
elsif i == 1
  "#{i} bottle of beer on the wall, " +
  "#{i} bottle of beer.\n" +
  "Take it down and pass it around, " +
  "no more bottles of beer on the wall.\n"
else
  "No more bottles of beer on the wall, " +
  "no more bottles of beer.\n" +
  "Go to the store and buy some more, " +
  "99 bottles of beer on the wall.\n"
```

pierwszy to wers od 99 do 3. Wszystkie one są takie same, różnią się jedynie cyfrą oznaczającą liczbę butelek. Drugi rodzaj to wers gdy zostały 2 butelki. Jest on inny, gdyż po zabraniu jednej butelki mamy informację iż "1 butelka została na ścianie", w liczbie pojedynczej, a nie jak w przypadku wersów 99 do 3, w liczbie mnogiej. Trzeci rodzaj wersu to gdy została tylko jedna butelka, oraz ostatni, gdy nie ma już butelek. Widać to w powyższym kodzie wyraźnie z powodu przejrzystej składni if/elsif/end. W obecnej wersji, która posiada lepszy wynik w metryce ABC, nie jest to jednak tak oczywiste:

```
if verse_number >= 2
    "#{verse_number} #{container_before_taken} of beer on the wall, " +
    "#{verse_number} #{container_before_taken} of beer.\n" +
    "#{one_down}" +
    "#{one_less} #{container_after_taken} of beer on the wall.\n"
  elsif verse_number == 1
    "#{verse_number} bottle of beer on the wall, " +
    "#{verse_number} bottle of beer.\n" +
    "Take it down and pass it around, " +
    "no more bottles of beer on the wall.\n"
  else
    "No more bottles of beer on the wall, " +
    "no more bottles of beer.\n" +
    "Go to the store and buy some more, " +
    "99 bottles of beer on the wall.\n"
  end
```

jak widać połączono sekcje kodu odpowiedzialnego za wersy 99-3 z sekcją kodu odpowiedzialną za wers 2 w jeden warunek "*if verse_number >= 2*". Pozwoliło to pozbyć się duplikacji zgodnie z zasadą DRY. Podczas refaktoryzacji należy brać pod uwagę iż wszystko ma swój koszt, również uniwersalna zasada DRY posiada koszt. Uzyskujemy korzyść braku duplikacji i mniejszej ilości kodu, elegancji kodu, kosztem możliwie zmniejszonej czytelności - w tym wypadku zmniejszona jest czytelność logiki piosenki "99 bottles", nie widzimy na pierwszy rzut oka iż istnieją 4 rodzaje wersów. Widzimy za to nadal iż wersje 99-3 oraz 2 są bardzo podobne, różnią się jedynie liczbą pojedynczą od bottles („*bottle*”).

v2.rb

W wersji v1.rb w niektórych miejscach zamiast wpisanego "na sztywno" zwrotu "*bottles*" oraz "*bottle*" posiadamy zmienne "*bottles_before_taken*" oraz "*bottles_after_taken*". Obecnie w wersji v2 po refaktoryzacji zmieniono nazwy tych zmiennych na takie zawierające zwrot "*container*" zamiast "*bottles*". Dlaczego nazwałem te zmienne "*container*"? By kod był łatwiejszy w zmianie, był w przyszłości łatwiejszy do modyfikacji. Jeśli ktoś zadecyduje iż chce by piosenka nie mówiła o butelkach piwa, ale np. puszkach, to wystarczy, że zmieni ciąg znaków widoczny pod tymi zmiennymi, a nazwa zmiennych nadal będzie aktualna. "Container" oznacza ogólnie "pojemnik", a czy tym pojemnikiem będzie puszka czy butelka, to już jest nieistotne dla zmiennej. Oprócz tego dodano też zmienną "beverage" (z ang. "napój") pod którą można podstawić coś innego niż "beer". To również prezentuje mechanizm korzyść/koszt podczas refaktoryzacji - zmniejszyłem być może czytelność ("**understanding**") kodu, zwiększając jednak jego "edytowalność" ("**changeability**"), prostotę modyfikacji.

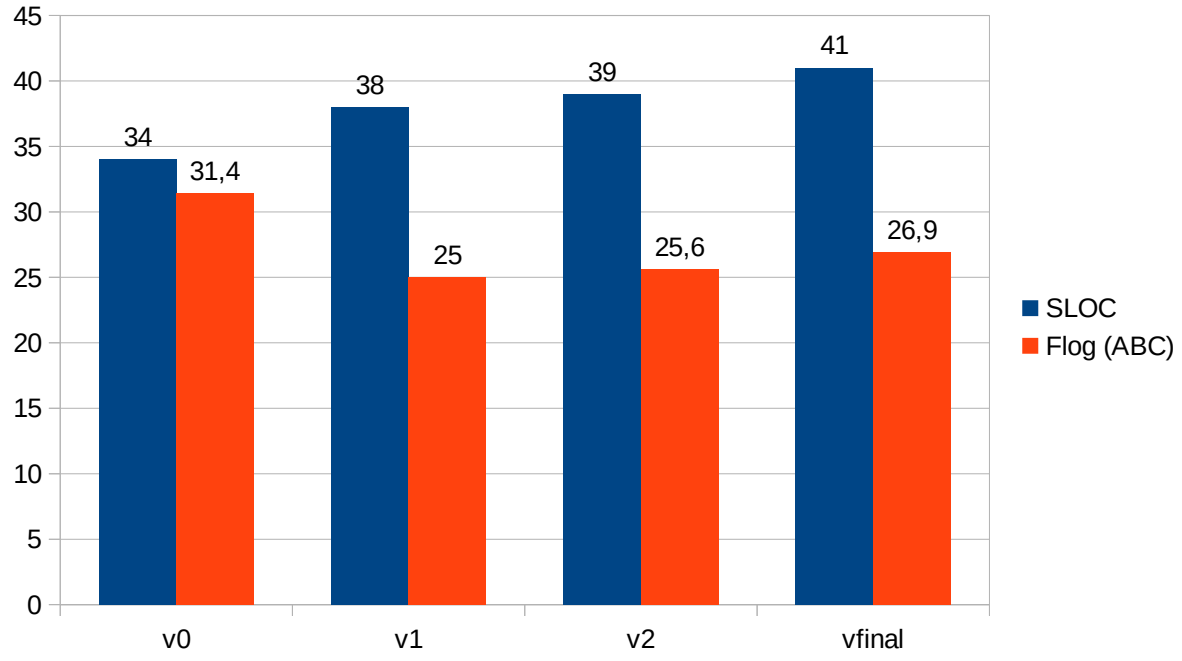
Zakładając iż zmiany nie będą wymagane w kodzie już po jego stworzeniu i wdrożeniu (*deployu*), wersja vfinal lub v1 mogłyby być odpowiednie. Jeśli jednak przewidujemy iż w przyszłości ktoś może kod chcieć rozwinąć lub zmienić, to wersja v2.rb jest lepsza. Umożliwia np. stworzenie piosenki w wersji "dla dzieci" i podstawienie w odpowiednie zmienne "*can/cans*" zamiast "*bottle/bottles*" oraz "*juice*" zamiast "*beer*". Spójrzmy na wskaźniki ABC oraz SLOC obecnej wersji kodu:

SLOC wersji v2:

```
{:total=>39, :empty_lines=>2, :single_comment=>25}}
```

FLOG wersji v2:

```
25.6: flog total
8.5: flog/method average
19.3: Bottles#verse                    v2.rb:3-28
```



Wynik flog jest gorszy niż wersji *v1*, ale minimalnie i jest nadal lepszy niż *vfinal*. Kod zawiera tylko jedną linię więcej niż wersja *v1*, ale nadal mniej niż *vfinal*. Uważam, iż ta wersja kodu uzyskuje pożadany balans pomiędzy jego czytelnością a łatwością edycji kodu. Nie widać na pierwszy rzut oka jak wyglądają kolejne wersje piosenki, jednak po chwili bardzo łatwo to zauważyć. W zamian zyskujemy możliwość dostosowania kodu i rozbudowania go, czego nie można powiedzieć o wersji "shameless green" z książki Sandi Metz (vfinal.rb). Kod nie stosuje też zaaplikowanych na siłę sztuczek które zmniejszałyby metrykę SLOC ale bardzo zmniejszałyby zrozumienie kodu. Sama w sobie mała ilość linii kodu nie może być celem samym w sobie, gdyż podążając za tak płytką metryką możemy szybko stworzyć skutki uboczne.

Dalsza refaktoryzacja mogłaby ulepszyć elegancję kodu w wersji v2, jednocześnie dbając by był on łatwy w zmianie/rozwoju lub by było to łatwiejsze, np. poprzez rozbicie metody verses na mniejsze metody. Jak zauważa Sandi Metz należy pamiętać by nie tylko pojedyncze metody były łatwe do zrozumienia, ale cały kod nie sprawiał wrażenie zbędnie skomplikowanego, tylko i wyłącznie w celu osiągnięcia lepszego wyniku w jakiejś metryce (SLOC).