JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI X (2017) 200-203
Wysłane: 2017-06-29
Przyjęte: 2017-07-03

Analiza wpływu refactoringu na jakość kodu – analiza porównawcza dwóch przypadków

Mariusz Łukasik*, Marek Miłosz
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. w metodykach zwinnych jedną z technik poprawy jakości kodu jest jego refaktoryzacja. Jest to proces, wykorzystujący szereg technik, modyfikacji kodu bez zmiany jego funkcjonalności ukierunkowany na poprawę jego przejrzystości i zmniejszenie podatności na błędy. Poprawę kodu można mierzyć przy pomocy różnych metryk jakości kodu. w artykule przedstawiono analizę wpływu refaktoryzacji na statyczną jakość kodu na przykładzie open-sourcowego projektu Scuba. Jakość kodu została zmierzona w dwóch różnych punktach rozwoju oprogramowania – bezpośrednio przed i po refaktoryzacji kodu. Do pomiaru wykorzystano trzy najbardziej popularne zestawy metryk jakości kodu obiektowego i narzędzie Sonarqube. Badania wskazują jednoznacznie na istotną poprawę statycznej jakości kodu w wyniku refaktoringu.

Słowa kluczowe: refaktoryzacja; statyczna jakość kodu; metryki obiektowe

*Autor do korespondencji.

E-mail address: mrl1992@o2.pl

Analysis of the impact of refactoring on code quality – comparative analysis of two cases

Mariusz Łukasik*, Marek Miłosz
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. In agile methods, one of the techniques for improving code quality is refactoring. This is a process that employs a number of techniques, modifying the code without changing its functionality, aiming to improve its transparency and reduce vulnerability. You can measure the improvement of the code using different code quality metrics. The paper presents an analysis of the effect of refactoring on static code quality on the example of the open-source project Scuba. The quality of the code was measured at two different points of software development - right before and after refactoring the code. The three most popular sets of object code quality metrics and the Sonarqube tool were used for the measurement. The research clearly demonstrates the significant improvement of static code quality as a result of refactoring.

Keywords: refactoring, static code analysis, object metrics

*Corresponding author.

E-mail address: mrl1992@o2.pl

1. Wstęp

Zarządzanie projektami informatycznymi należy do skomplikowanych przedsięwzięć. Zawiera w sobie wiele działań i niesie ze sobą duże ryzyko niepowodzenia [1]. Osiągniecie sukcesu w obszarze IT wymaga konkretnej metodyki postępowania. Metodyki zwinne (*ang. Agile*) są jednym z przykładów metodyk wytwarzania oprogramowania, gdzie wykorzystywane są odmienne zasady i reguły postępowania, niż w powszechnie używanych metodykach tradycyjnych. Motywem stosowania metodyk zwinnych w zarządzaniu projektem jest precyzyjna i skuteczna reakcja na nowe problemy i wyzwania, pojawiające się w trakcie tworzenia oprogramowania. Dzięki elastycznemu podejściu, jakie oferują metodyki zwinne, łatwiejsze jest osiągnięcie wysokiego poziomu użyteczności biznesowej uzyskanej przez wytworzone i wdrożone oprogramowanie. Użytkownicy wytwarzanego oprogramowania w początkowym etapie prac projektowych często nie są w stanie precyzyjnie określić dokładnych wymagań i potrzeb związanych z oczekiwanym oprogramowaniem. w takiej sytuacji metodyki zwinne umożliwiają prowadzenie projektu i zakończenie go sukcesem, inaczej niż, gdyby użyto metodyk tradycyjnych, gdzie wszystkie wymagania odnośnie docelowego produktu muszą być skonkretyzowane już na początku prac projektowych [1]. Jedną z technik stosowanych w metodykach zwinnych wytwarzania oprogramowania jest refaktoryzacja kodu.

Refaktoryzacja (*ang. refactoring*) jest to proces wprowadzania zmian w strukturze projektu informatycznego nie mający wpływu na zmianę funkcjonalności ale poprawiający wydajność, koszty jego tworzenia, jakość oprogramowania [2]. Przez jakość oprogramowania rozumieć tu należy: testowalność, użyteczność, pielęgnację, zrozumienie, a także poziom osiąganej abstrakcji. Metod i zasad refaktoryzacji jest dużo i mogą być wyrażane w różnych poziomach abstrakcji. Przygotowywanie kodu na możliwe zmiany jest składnikiem idei programowania defensywnego. Jego podstawowym celem wydaje się być takie zabezpieczenie kodu, by zmiany lub nieprawidłowe wywołania w innej strefie projektu nie spowodowały szkód. Refaktoryzacja jest też możliwa na poszczególnych poziomach projektu takich jak: poziom danych, instrukcji, systemu , implementacji klasy lub interfejsu [2].

2. Metryki statycznej jakości kodu

Metryki statycznej jakości kodu są szybką i bardzo wygodną możliwością oceny jakości tworzonego oprogramowania. Znajomość ich wartości i wielkości pozwala na szacowanie złożoności, kosztu pielęgnacji oraz elastyczności systemu. Najbardziej popularne metryki obiektowe to: Martina, MOOD oraz Chidambera i Kemerera.

Metryki Chidambera i Kemerera powszechnie nazywane są zestawem metryk CK. Zostały one zaprojektowane w 1991 roku przez S.R. Chidambera i C.F. Kemerera [3]. Zawierają w sobie 6 metryk badających takie aspekty obiektowości, jakie do tej pory nie były używane w kontekście pomiarów oprogramowania: dziedziczenie, złożoność klasy, powiązania między klasami i ich spójność. Metryki te badają pojedyncze klasy i zbiory klas, stąd też służą jako narzędzie do niskopoziomowego badania jakości kodu obiektowego. Zestaw metryk CK składa się z następujących metryk [4]:

- **Depth of Inheritance Tree** (ang. *głębokość drzewa dziedziczenia, DIT*);
- **Response for a Class** *(*ang. *odpowiedź klasy, RFC;*
- **Lack of Cohesion Of Methods** (ang. *brak spójności metod,* LCOM);
- **Coupling Between Objects** (ang. *powiązania pomiędzy obiektami, CBO*);
- **Number of Children** (ang. *liczba klas pochodnych, NOC*);
- **Weighted Method per Class** (ang. *ważona liczba metod w klasie, WMC*).

Inaczej niż w przypadku metryk Chidambera i Kemerera, metryki Martina biorą pod uwagę pojęcie powiązania jednej klasy z drugą. z omówionymi tutaj metrykami wiążą się dwie zasady [5]. Są to zasada stabilne abstrakcyjne SAP (*ang. Stable Abstractions Principle*), a także stabilnych zależności SDP (*ang. Stable Dependency Principle*). w szeroko dostępnych narzędziach stosujących metryki używa się 5 następujących metryk [6]:

- **Afferent coupling (Ca)** – *(ang. powiązania do wewnątrz – dośrodkowe;*
- **Efferent coupling (Ce)** – *(ang. powiązania do zewnątrz - odśrodkowe);*
- **Abstractness (A)** – *(ang. abstrakcyjność);*
- **Instability (I)** – (*ang. niestabilność*);
- **Normalize distance from main sequence (Dn) -** (*ang. znormalizowana odległość od ciągu głównego*).

Metryki MOOD (*ang. Metrics of Object Oriented Design*) składają się z 6 metryk zaprezentowanych przez F. B. e Abreu w roku 1994 [7]. Kilka z nich zostało zmodyfikowanych i zaproponowanych w roku 1995 również przez tych samych autorów w takim kształcie jak obecnie są omawiane. Celem nadrzędnym tych metryk jest analiza stopnia użycia następujących paradygmatów obiektowych: hermetyzacji, powiązań, polimorfizmu, dziedziczenia. Przedmiotem pomiaru jest system jako całość, a nie konkretne klasy i relacje miedzy nimi lub pakietami. Metryki te dają zatem obraz jakości całego projektu, co więcej są mocno skalowalne i nie zależą od rozmiaru badanego systemu lub języka implementacji. Nie dają niestety obiektywnego wyniku w przypadku systemów, w których główną rolę grają formularze, generacja kodu i segmenty współdzielone przez cały system, w których podział odpowiedzialności do klas opiera się na różnych kryteriach [8]. Wszystkie metryki MOOD posiadają pewne cechy wspólne. Wyrażane są jako poziom wykorzystania w projekcie w stosunku do maksymalnego możliwego poziomu. z racji tego są to wartości wyrażane procentowo. Dzięki temu dają możliwość szybkiej oceny całości systemu, co jest przydatne głównie z przeznaczeniem dla poziomu zarządzania projektem.

Wśród metryk MOOD rozróżnia się:

- **Attribute Hiding Factor (AHF)** (ang. *współczynnik ukrycia atrybutów*);
- **Method Hiding Factor (MHF)** (ang. *współczynnik ukrycia metod*);
- **Attribute Inheritance Factor (AIF)** (ang. *współczynnik dziedziczenia atrybutów*);
- **Method Inheritance Factor (MIF)** (ang. *współczynnik dziedziczenia metod*);
- **Polymorphism Factor (PF)** (ang. *współczynnik polimorficzności*);
- **Coupling Factor (CF)** (ang. *współczynnik powiązań*).

3. Teza i metodyka badań

Refaktoryzacja ma między innymi służyć poprawie jakości kodu. Realnym zatem wydaję się postawienie następującej tezy: refaktoryzacja poprawia statyczną jakość kodu. w celu udowodnienia tej tezy wykonany został eksperyment. Polegał on na wyznaczeniu wartości metryk statycznej jakości kodu dla oprogramowania rozwijanego projektu w dwóch wersjach: bezpośrednio przed i po jego refaktoryzacji.

Obiektem badań był projekt Scuba. Kod pozyskany został z repozytorium projektu: https://github.com/xwangsd/Scuba. Scuba jest jedną z wersji frameworka SCUBA służącego do programowania kart inteligentnych (ang. S*mart Cards*). Autor dokonał na nim refactoryzacji polegającej na usunięciu części niepotrzebnego kodu, wydzieleniu kodu do osobnych klas itp. Wybór tego projektu uzasadniony jest ze względu na jego wielkość. Posiada on ponad 10 000 linii kodu a także ponad 200 klas.

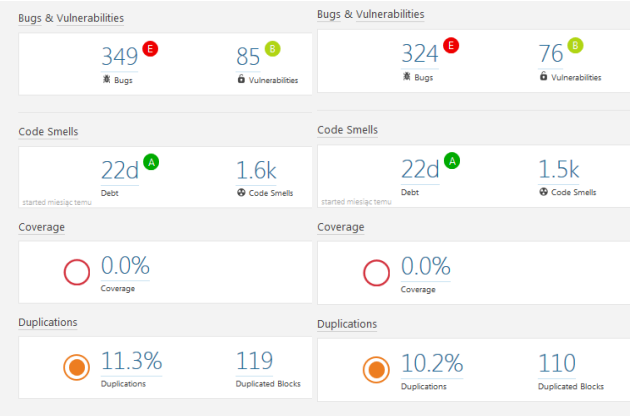
Do analizy jakości kodu użyte zostało narzędzie Sonarqube. Narzędzie to umożliwia kompleksową analizę kodu w ośmiu obszarach: niezawodność, bezpieczeństwo, łatwość utrzymania, pokrycie testami, duplikaty, parametry projektu, złożoność, problemy do rozwiązania. Wyniki w niektórych obszarach są oceniane literami od a do E, gdzie ocena (*ang. rating*) oznaczona literą a jest oceną najlepszą, a E – najgorszą.

4. Rezultaty badań

Celem badania będzie sprawdzenie tezy: ***Refaktoryzacja poprawia jakość kodu.***

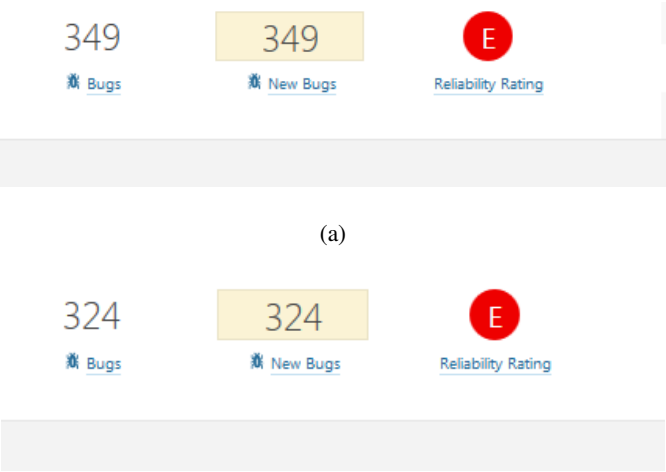
Przechodząc do szczegółowego omówienia analizy kodu podzielono je na 8 sekcji, pod kątem których badano projekt

Scuba: niezawodność, bezpieczeństwo, łatwość utrzymania, pokrycie testami, duplikaty, parametry projektu, złożoność, problemy do rozwiązania. Wyniki badań przedstawiają rys. 1-3.



(a) (b)

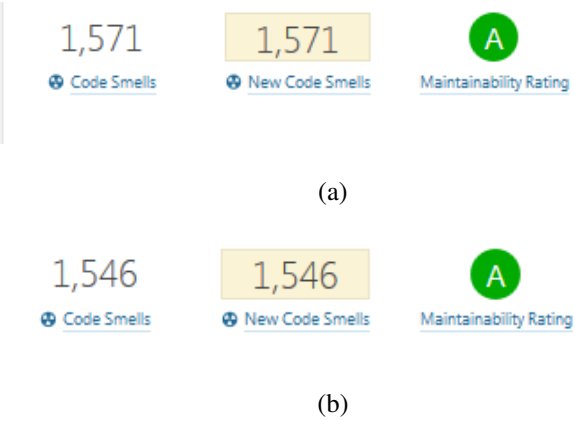
Rys. 1. Ogólna analiza jakości kodu projektu przed (a) i po (b) refaktoryzacji



(a)

(b)

Rys. 2. Widok dotyczący niezawodności projektu w panelu Sonarqube przed (a) i po (b) refaktoryzacji



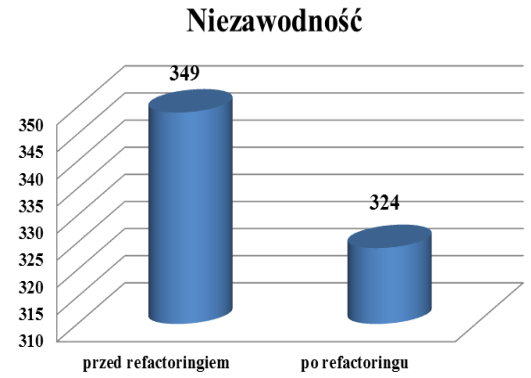
(a)

(b)

Rys. 3. Widok dotyczący łatwości utrzymania projektu w panelu Sonarqube przed (a) i po (b) refaktoryzacji

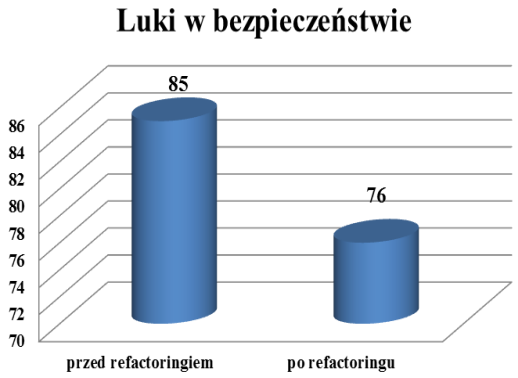
5. Analiza porównawcza

W niniejszym rozdziale dokonano analizy porównawczej wraz z dyskusją. Analiza porównawcza w formie graficznej podsumowuje badania. Na wykresach (rys. 4-8) przedstawiono zmiany w jakości kodu po dokonaniu refaktoryzacji.



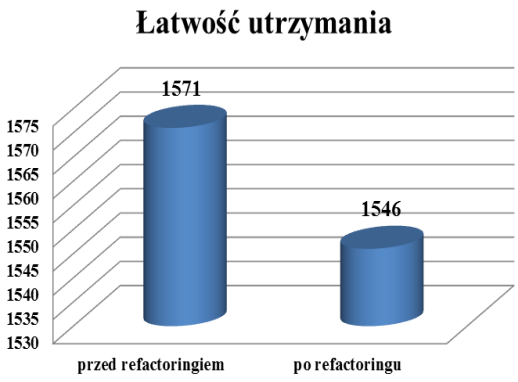
Rys. 4. Porównanie liczby błędów w kodzie

Niezawodność kodu mierzona liczbą błędów spadła po refaktoryzacji z 349 do 324 (rys. 4).



Rys. 5. Porównanie liczby luk w bezpieczeństwie

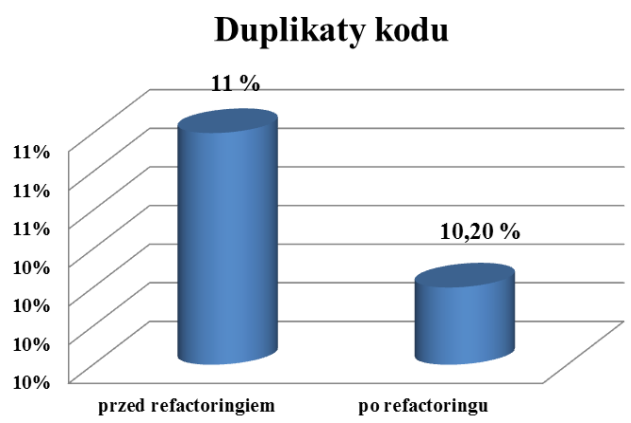
Liczba luk w bezpieczeństwie także spadła z 85 do 76 (rys.5).



Rys. 6. Porównanie łatwości utrzymania kodu

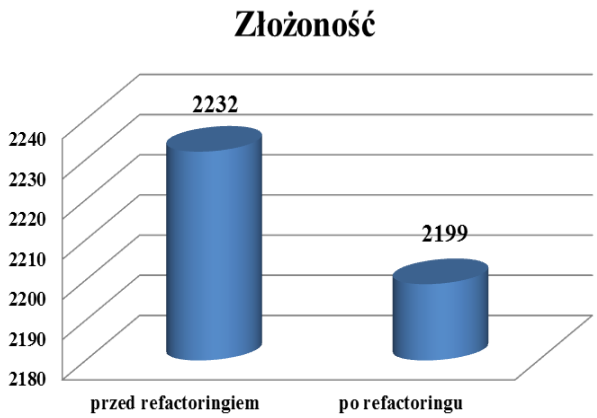
Analizując łatwość utrzymania projektu można dostrzec, że spadła ilość miejsc w kodzie wymagających zmiany

implementacji jako potencjalne miejsca przyszłych problemów z 1571 do 1546 (rys. 6).



Rys. 7. Porównanie ilości udziału duplikatów w kodzie

Stosunek kodu zduplikowanego do kodu unikalnego spadł z 11% do 10,2%. Jeśli chodzi o liczbę linii kodu to spadła z 1947 do 1795 (rys. 7).



Rys. 8. Porównanie złożoności kodu

Co ważne projekt stał się także mniej złożony. Ocena złożoności spadła z 2232 do 2199 pkt. (rys. 8).

6. Podsumowanie

W artykule dokonano przeglądu statycznych metryk analizy kodu. w części badawczej metryki zostały wykorzystane do badania projektu o nazwie Scuba pod kątem wpływu refaktoryzacji na jakość kodu. Omówiono również wyniki analizy a także przedstawiono wnioski.

Artykuł ten może być źródłem nauki metryk statycznej analizy kodu w kontekście praktycznym. Używanie metryk często wymaga doświadczenia i wyrobienia własnych doświadczeń. Metryki stanowią cenne wsparcie w procesie refaktoryzacji. Otwierają możliwość oceny jakości kodu na wielu poziomach abstrakcji i badania parametrów, które przekładają się na ogólną ocenę niezawodności i funkcjonalności systemu informatycznego.

Podsumowując warto skomentować postawioną wcześniej tezę, że refactoring wpływa na jakość kodu. Z analizy dokonanej za pomocą Sonarqube wynika jednoznacznie, że tak. Potwierdzają to m.in. pomiary złożoności projektu czy ilości duplikatów w kodzie ale także ocena niezawodności i łatwości utrzymania. Eksperyment badawczy wykorzystywał jeden projekt. By statystycznie udowodnić poprawność tezy należałoby zbadać większą liczbę projektów.

Literatura

[1] Mirosława Lasek, Aleksandra Adamus, *Informatyka ekonomiczna,* 2014.
[2] M. Miłosz, M. Borys, M. Plechawska-Wójcik, *Metodyki zwinne wytwarzania oprogramowania,* Politechnika Lubelska, 2011.
[3] S.R. Chidamber, C.F. Kemerer, a *metrics suite for object-oriented design,* IEEE Transactions on Software Engineering, No 6, Vol. 20, p. 476-493, 1994.
[4] Dr. Linda Rosenberg, *Applying and interpreting OO Metrics*, http://www.literateprogramming.com/ooapply.pdf.
[5] B. Henderson-Sellers, *Object-Oriented Metrics, measures of Complexity*, Prentice Hall, 1996.
[6] Robert Martin, *OO Design Quality metrics*. October 28, 1994.
[7] S.H. Kan, *Metryki i modele w inżynierii jakości oprogramowania,* PWN 2006, s. 355-383.
[8] F. B. Abreu, *The MOOD Metrics Set*. ECOOP Workshop on Metrics, 1995