JCSI 1 (2016) 54-59
Journal of Computer Sciences Institute

Wysłane: 2016-09-30
Przyjęte: 2016-10-08

Metodyka tworzenia oprogramowania a jakość kodu – analiza porównawcza dwóch przypadków

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

Streszczenie. Jakość kodu zależy od stosowania dobrych praktyk jego wytwarzania. W artykule przedstawiono metryki pomiaru jakości kodu tworzonego zgodnie z paradygmatem obiektowym i narzędzia informatyczne wyznaczające ich wartości. Rozpatrzono dwa przypadki rozwoju oprogramowania przez różne zespoły w różny sposób zarządzane. Przeanalizowano jakość kodu w kolejnych pięciu wersjach programów. Rezultaty badań pozwoliły na sformułowanie wniosku o przewadze metodyki lekkiej, na co wskazują lepsze wartości większości metryk.

Słowa kluczowe: jakość kodu; metryki pomiaru; metodyki wytwarzania oprogramowania

*Autor do korespondencji.

Adres E-mail: b.r.zalewski@gmail.com

Methodology of software development vs. code quality – a comparative analysis of two cases

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

Abstract. Code quality is strongly dependent on using best coding practices during it's development. This paper presents various code quality metrics in object oriented programming and computer tools to it automatic measurement. Two cases of software development by two different teams were considered. Code quality was analyzed in five following program versions. This study shows better value of almost (but not all) code quality metrics developed using agile methodology. It raises the conclusion about agile methodology advantage.

Keywords: code quality; measurement metrics; methodologies of software development

*Corresponding author.

E-mail address: b.r.zalewski@gmail.com

1. Wstęp

Aby wytwarzać dobre jakościowo oprogramowanie należy nieustannie poddawać je testom. Najczęściej programy testuje się pod względem spełniania funkcjonalności. Często zapomina się lub jawnie lekceważy testowanie jakości kodu, który realizuje poszczególne funkcjonalności.

Statyczna analiza kodu umożliwia sprawdzenie jakości kodu użytego w aplikacji. Dzięki analizie statycznej można na bieżąco sprawdzać, czy tworzony kod jest zgodny z regułami projektowania systemów informatycznych, które przekładają się na cechy oprogramowania takie jak niezawodność, złożoność zarządzania czy utrzymanie. Obliczanie wartości metryk mających za zadanie sprawdzić jakość napisanego kodu nie musi wiązać się z koniecznością poświęcania na to dodatkowego czasu, ponieważ istnieją narzędzia lub dodatki do IDE (ang. *Integrated Development Enviroment*), które robią to automatycznie na wybranym projekcie [1]. Jakość wytworzonego oprogramowania wynikać może również z tego, w jaki sposób realizowany był projekt, którego wynikiem było to oprogramowanie. Obecnie znaleźć można kilka podejść do realizacji projektu informatycznego. Metodykami polecanymi, powszechnie uważanymi za skuteczne, są metodyki zwinne (ang. *Agile*), np. SCRUM, Extreme Programming, TDD (ang. *Test-Driven Development*). Są to metodyki lekkie i elastyczne, zorientowane na dostarczenie produktu. Jednocześnie wiele projektów realizowanych jest bez oparcia o sprawdzone metodyki. Spowodowane jest to często niewiedzą lub chęcią zaoszczędzenia czasu oraz pieniędzy, co niekoniecznie przynosi zamierzony cel [2]. Stosowanie metodyk realizacji projektów często wiąże się z pójściem na kompromis. Dla przykładu metodyka SCRUM wymaga planowania zadań na tydzień lub więcej w przód, co w niektórych zespołach może wydawać się nierealne, gdyż właściciel produktu wymaga od nich podejścia ad-hoc. Panuje opinia, że metodyki zwinnego programowania stosuje się tylko w dużych firmach. W rzeczywistości jednak takie praktyki stosowane są również w małych przedsiębiorstwach [3]. Metodyki zwinne wprowadzają również większą dyscyplinę przy realizowaniu projektu informatycznego [2].

2. Metryki pomiaru jakości kodu

W trakcie rozwoju programowania zorientowanego obiektowo zaczęto tworzyć zasady, dzięki którym wytwarzany kod miał posiadać lepszą jakość. Jakość ta miała przekładać się na łatwiejsze zarządzanie, testowanie kodu, zmniejszenie jego złożoności oraz pozwalać na ponowne wykorzystanie już istniejących modułów bez konieczności ich powielania. Różni badacze dziedziny inżynierii oprogramowania analizując dobre praktyki i wzorce wytwarzania oprogramowania zorientowanego obiektowo przedstawiali własne miary mające ocenić, jak bardzo wytworzony kod spełnia daną zasadę. W tym artykule przedstawione zostaną następujące metryki: złożoność cyklomatyczna McCabe'a, miary Halsteada,

metryki z zestawu MOOD, metryki Roberta C. Martina oraz zestaw metryk CK.

Złożoność cyklomatyczna McCabe'a (ang. *Cyclomatic Complexity – CC*) jest metryką, która została zaprojektowana z myślą o strukturalnych językach programowania, ale aktualnie znajduje swoje zastosowanie w określeniu miejsc trudnych do testowania oraz utrzymania w obiektowych językach programowania [4]. Zasada wyliczania tej miary polega na określeniu liczby ścieżek wykonania danej część oprogramowania [4].

Miary Halsteada dotyczą funkcji i argumentów danego modułu oprogramowania [5]. Halstead przedstawił pięć następujących miar [5]: długość programu (*ang. Length of a Program*), słownik programu (*ang. Vocabulary*), objętość (*ang. Volume*), trudność (*ang. Difficulty*) oraz pracochłonność (*ang. Effort*). Na bazie miar przedstawionych przez Halsteada opracowano metryki bardziej złożone, m.in. [6] poziom programu (ang. *Program Level*), czas implementacji (ang. *Time*) a także szacunkowa liczba błędów (ang. *Number of Delivered Bugs*).

Zestaw metryk MOOD (*ang. Metrics for Object-Oriented Design*) to zbiór składający się z sześciu metryk, takich jak: współczynnik polimorficzności (*ang. Polymorphism Factor - PF*), współczynnik ukrycia atrybutów (*ang. Attribute Hiding Factor – AHF*), współczynnik ukrycia metod (*ang. Method Hiding Factor – MHF*), współczynnik dziedziczenia atrybutów (*ang. Attribute Inheritance Factor – AIF*), współczynnik dziedziczenia metod (*ang. Method Inheritance Factor – MIF*) oraz współczynnik powiązań (*ang. Coupling Factor – CF*). Istotnym, z perspektywy zarządzania projektem, jest fakt, że zastosowanie metryk MOOD skutkuje możliwością szybkiej i ogólnej oceny jakości oprogramowania [7]. Metryki MOOD pozwalają na pomiar elementów programu takich jak: hermetyzacja, polimorfizm, dziedziczenie i powiązania pomiędzy klasami. Sześć istniejących metryk odpowiada za pomiar poszczególnych składników paradygmatów programowania zorientowanego obiektowo [8].

Metryki Roberta C. Martina to zbiór składający się z pięciu metryk. Są to metryki: powiązania do wewnątrz (*ang. Affarent Coupling – Ca*), powiązania na zewnątrz (*ang. Efferent Copling – Ce*), abstrakcja (ang. Abstractness – A), niestabilność (*ang. Instability – I*) oraz znormalizowana odległość od ciągu głównego (*ang. Normalized Distance from Main Sequence – Dn*). Metryki te dotyczą zależności pomiędzy klasami w projekcie. Dwie najważniejsze metryki opierają się na rozróżnieniu powiązań na powiązania do wewnątrz i powiązania na zewnątrz [9].

Metryki Chidambera i Kemerera nazywane są powszechnie zestawem metryk CK [10]. Jest to obecnie jeden z najpopularniejszych zbiorów metryk obiektowych pozwalający na pomiar złożoności klas [11]. Zawiera on sześć metryk odnoszących się do różnych aspektów obiektowości: dziedziczenia, złożoności klasy, powiązania pomiędzy klasami i spójności. Metryki te pozwalają na niskopoziomowe szacowanie jakości kodu w obiektowych językach programowania. W skład zbioru metryk CK wchodzą [10]: ważona liczba metod w klasie (*ang. Weighted Methods per Class – WMC*), odpowiedź klasy (*ang. Response for a Class – RFC*), głębokość drzewa dziedziczenia (*ang. Depth of Inheritance Tree – DIT*), liczba pośrednich potomków klasy (*ang. Number of Childre of Class – NOC*), powiązania między klasami (*ang. Coupling Between Objects – CBO*) oraz brak spójności metod (*ang. Lack of Cohesion in Methods – LCOM*).

W tabeli 1 przedstawiono optymalne wartości metryk, które zostały użyte do porównania jakości dwóch projektów rozpatrywanych w tej pracy. W tabeli zawarto tylko te metryki, które posiadają określoną wartość optymalną.

Tabela 1. Wartości optymalne metryk użyte do porównania projektów informatycznych. Źródło: opracowanie własne na podstawie [12].

Nazwa metryki	**Optymalna wartość**
Ca	0-500
Ce	0-20
I	0,7 - 1 pakiety niestabilne 0 - 0,3 pakiety stabilne
Dn	0,2
WMC	20
RFC	20-100
DIT	0-6
NOC	2-5
CBO	0-7
LCOM2	0-1

3. Narzędzia informatyczne do analizy jakości kodu

W konsekwencji rozwoju badań nad jakością kodu powstały narzędzia umożliwiające zautomatyzowany pomiar wartości z wykorzystaniem poszczególnych metryk jakości kodu. Obecnie istnieje wiele programów umożliwiających pomiar kodu stworzonego w różnych językach programowania. Są to programy niezależne lub zintegrowane ze środowiskami rozwoju oprogramowania. Istnieje kilka istotnych programów umożliwiających badanie jakości kodu napisanego w języku C#. Są to narzędzia analityczne w Visual Studio, SonarQube, NDepend, Source Monitor.

Visual Studio od roku 2007 pozwala na przenalizowanie projektu pod kątem wartości pięciu metryk [13]. Narzędzie to nosi nazwę Visual Studio Code Metrics Powertool for Visual Studio 2015. Umożliwia ono pomiar metryk [14]: *Maintainability Index* (metryka indeksu trudności utrzymania kodu), *Cyclomatic Complexity* (metryka złożoności cyklomatycznej), *Depth of Inheritance* (metryka głębokości drzewa dziedziczenia z zestawu metryk CK), *Class Coupling* (metryką powiązań pomiędzy klasami z zestawu metryk CK) oraz *Lines of Code* (liczba linii kodu).

Oprogramowanie SonarQube, zwane uprzednio Sonar [15], jest platformą open source. SonarQube wylicza wartości podstawowych metryk, np. złożoność cyklomatyczna, liczba linii kodu. Wyniki każdej z sekcji mierzonej przez SonarQube mogą być prezentowane w formie graficznych raportów [15].

Source Monitor jest darmowym programem dostarczających podstawowych informacji na temat kodu. Source Monitor mierzy: liczbę linii kodu, łączną liczbę

instrukcji, procent komentarzy w kodzie, procent komentarzy będących dokumentacją, liczbę klas, średnią liczba metod w klasie, średnią liczbę instrukcji w metodzie, maksymalną oraz średnią złożoność klasy, maksymalną oraz średnią wielkość drzewa dziedziczenia. Dodatkowo możliwe jest wyświetlenie grafu kiviat obrazującego, czy wartości metryk mieszczą się w rekomendowanych wartościach [16].

NDepend jest najpopularniejszym narzędziem do badania jakości kodu na platformie .NET [17]. W przeciwieństwie do pozostałych programów jest narzędziem płatnym. Posiada jednak dwutygodniową wersję próbną dającą te same możliwości, co płatna wersja programu. NDepend umożliwia zautomatyzowanie procesu analizy kodu poprzez zintegrowanie go z programami do ciągłej integracji (ang. *Continous Integration*), np. TeamCity lub z mechanizmem MSBuild. Umożliwia to generowanie szczegółowego raportu przy każdej kompilacji programu bez konieczności ręcznego uruchamiania analizy. NDepend pozwala również na integrację z Visual Studio. Posiada również opcję raportowania otrzymanych wyników do przejrzystych stron html [17].

4. Plan i rezultaty badań

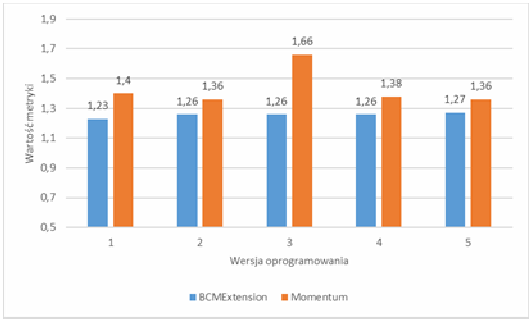
Celem pracy badawczej było sprawdzenie postawionej następującej tezy:
Stosowanie dobrych praktyk wytwarzania oprogramowania zorientowanego obiektowo podnosi wartość metryk sprawdzających jakość kodu.

Eksperyment polegał na porównaniu dwóch programów komputerowych pod kątem wartości metryk mierzących jakość napisanego kodu. Oba programy były wytwarzane w tych samych technologiach i przy użyciu jednakowych narzędzi programistycznych. Programy różniły się stosowanymi metodykami wytwarzania oprogramowania, a mianowicie projekt BCMExtension rozwijany jest przez zespół programistów, których praca sterowana jest metodologią SCRUM, a projekt Momentum rozwijany jest bez stosowania żadnej metodyki. Do obliczeń wartości metryk jakości napisanego kodu został wybrany program NDepend. Do obróbki statystycznej otrzymanych wartości poszczególnych metryk jakości oprogramowania każdego z projektów użyty został program Microsoft Excel 2016. Wybranych zostało pięć wersji każdego z programów. Przy wyborze wersji programów polegano na sprintach wykonywanych przy wytwarzaniu oprogramowania BCMExtension. Wybrano wersję programu ze sprintów numer 1, 10, 20, 30 i 40. Odstępy czasowe pomiędzy poszczególnymi wersjami wyniosły od 2,5 do 3 miesięcy co pozwoliło na zauważenie różnic pomiędzy poszczególnymi wersjami programów. Następnie wybrano wersje programu Momentum adekwatną do dat ukończenia poszczególnych sprintów programu BCMExtension. Do określenia jakości kodu badanych programów posłużyła metryka: złożoność cyklomatyczna McCabe'a – do określenia złożoności metod zawartych w programach. By zobrazować CC każdego z programów została wyliczona średnia tej metryki oraz dodatkowo wartość maksymalna. By określić powiązania pomiędzy poszczególnymi klasami w projekcie wyliczone zostały wartości metryk z zestawu CK. Metryki z zestawu CK odnoszą się do konkretnych klas programu, więc w eksperymencie została obliczona wartość średnia oraz wartość maksymalna każdej z metryk tego zestawu. Dodatkowo obliczona została liczba klas, które przekraczają przyjęte optymalne wartości każdej z metryk. Do określenia poprawności konstrukcji pakietów porównywanych programów zostały wyliczone wartości poszczególnych metryk z zestawu Roberta C. Martina.

Szczegółowe rezultaty badań zostały przedstawione w pracy [18].

5. Analiza porównawcza jakości kodu w obu aplikacjach

Wyniki przeprowadzonego eksperymentu posłużyły do przeprowadzenia analizy porównawczej badanych projektów. Analiza opiera się na porównaniu poszczególnych wartości metryk projektu BCMExtension w odniesieniu do wartości metryk jakości projektu Momentum.

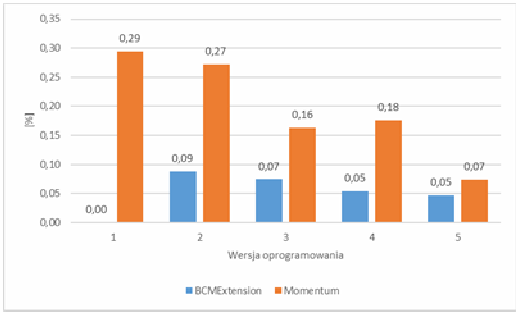


Rys. 1. Porównanie średnich wartości metryk złożoności McCabe'a dla poszczególnych wersji programu BCMExtension i Momentum.

Rys. 1 wskazuje, że średnia wartość metryki McCaba'a dla wszystkich wersji programu BCMExtension jest niższa, niż dla programu Momentum. Ostatnie wersje obu programów różnią się średnią wartością metryki CC o 0,09, na korzyść programu BCMExtension. Metryka CC wskazuje na trudność utrzymania i testowania badanego kodu. Program BCMExtension osiągną nieznacznie niższe wartości tej metryki w porównaniu z programem Momentum. Oznacza to, że program BCMExtension powinien być łatwiejszy w utrzymaniu i testowaniu od programu Momentum.

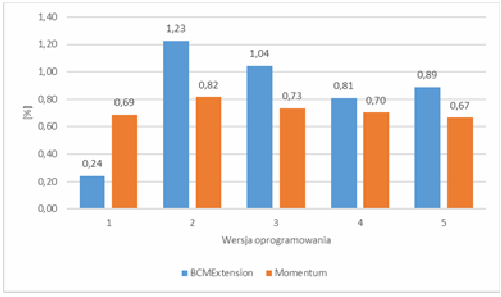
Dla zobrazowania jakości klas zawartych w programach BCMExtension i Momentum uśredniono wartość każdej z metryk zestawu CK. Na rys. 2 przedstawiono procentowy udział klas przekraczających optymalną wartość metryki LCOM2 w programach BCMExtension i Momentum. Liczba klas przekraczających optymalną wartość tej metryki została określona przez zsumowanie klas, dla których wartość metryki LCOM2 przekracza wartość 1. Udział klas przekraczających optymalną wartość metryki LCOM2 jest niewielki zarówno w programie BCMExtension oraz Momentum i nie przekracza 0,3%. Projekt BCMExtension osiąga jednak lepsze wartości metryki LCOM2 w porównaniu do projektu Momentum. Ostateczna różnica procentowa jest spowodowana większą liczbą klas występujących w wersji piątej programu BCMExtension. Niski udział procentowy

klas przekraczających optymalną wartość metryki LCOM2 świadczy o wysokiej jakości obu programów.



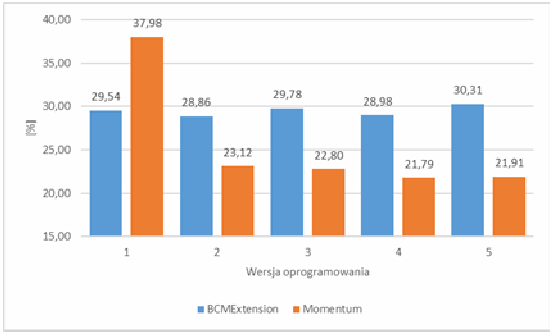
Rys. 2. Porównanie procentowego udziału typów przekraczających optymalną wartość metryki LCOM2 dla poszczególnych wersji programów BCMExtension i Momentum.

Inaczej niż w przypadku poprzedniej metryki, procentowe wartości klas przekraczających odpowiednią wartość metryki NOC są minimalnie gorsze dla projektu BCMExtension. Różnica ta nie przekracza 1%. Rys. 3 potwierdza, że oba projekty reprezentują wysoką jakość mierzoną metryką NOC.

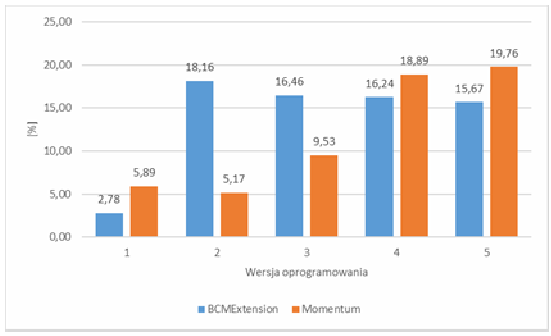


Rys. 3. Porównanie procentowego udziału typów przekraczających optymalną wartość metryki NOC dla poszczególnych wersji programów BCMExtension i Momentum.

Oba projekty reprezentują wysoki poziom klas przekraczających poprawną wartość metryki CBO (rys. 4), co powinno zaalarmować zespoły wytwarzające programy BCMExtension i Momentum i zdopingować do poprawy kodu klas przekraczających optymalne wartości tej metryki.



Rys. 4. Porównanie procentowego udziału typów przekraczających optymalną wartość metryki CBO dla poszczególnych wersji programów BCMExtension i Momentum.

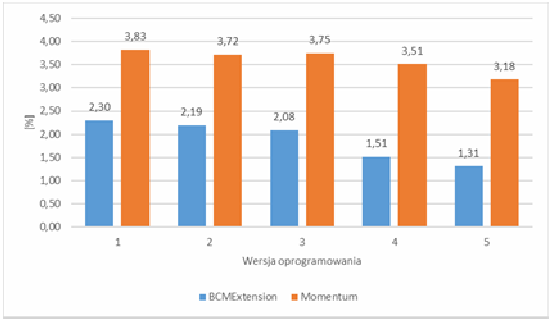


Rys. 5. Porównanie procentowego udziału typów przekraczających optymalną wartość metryki RFC dla poszczególnych wersji programów BCMExtension i Momentum.

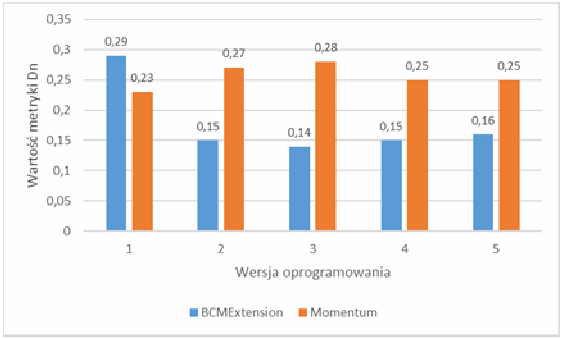
Kolejnym analizowanym rezultatem przeprowadzonego badania jest procentowy udział klas przekraczająca poprawną wartość metryki RFC przedstawiony na rys. 5. Pierwsze trzy wersje programu Momentum posiadają poprawny odsetek klas przekraczających optymalną wartość metryki RFC, który nie przekracza 10%. Dużo gorzej wypada wersja czwarta i piąta. W porównaniu do wersji trzeciej procent klas źle zaimplementowanych wzrasta dwukrotnie w wersji czwartej, osiągając blisko 20% wynik w wersji piątej. Program BCMExtension najniższy procent (wynoszący 2,5%) klas przekraczających rekomendowaną wartość metryki RFC osiąga w pierwszej wersji programu. W kolejnej wersji następuje gwałtowny wzrost do 18% niepoprawnie napisanych klas. Kolejne wersje programu BCMExtension notują delikatny spadek odsetka klas przekraczających poprawną wartość metryki RFC, by ostatecznie osiągnąć 15% w wersji piątej.

Program BCMExtension osiągnął dużo gorsze wyniki, niż program Momentum, w wersji drugiej i trzeciej. Od wersji czwartej BCMExtension prezentuje jednak niższy odsetek niepoprawnych klas, by ostatecznie w wersji piątej programu posiadać 5% mniej klas przekraczających optymalną wartość metryki RFC. Metryka RFC wskazuje na złożoność klasy, a wartości tej metryki większe niż 100 wskazują na zbyt dużą złożoność klasy powodującą problemy z testowaniem i utrzymaniem kodu. Oba projekty posiadają wysoki procent klas źle zaimplementowanych względem metryki RFC. Zespoły wytwarzające oba projekty powinny popracować nad poprawą kodu niepoprawnych klas.

Porównanie przedstawione na rys. 6 obrazuje, że program BCMExtension posiada niższy procent klas przekraczających poprawną wartość metryki WMC, we wszystkich pięciu wersjach programu, w porównaniu z programem Momentum. Piąta wersja programu BCMExtension posiada ponad dwukrotnie niższy procent niepoprawnie zaimplementowanych klas w porównaniu do piątej wersji programu Momentum. Oba projekty cechuje niski odsetek źle napisanych klas nieprzekraczający 4%.



Rys. 6. Porównanie procentowego udziału typów przekraczających optymalną wartość metryki WMC dla poszczególnych wersji programów BCMExtension i Momentum.



Rys. 7. Porównanie metryki Dn warstwy dostępu do danych, dla poszczególnych wersji programów BCMExtension i Momentum.

Do wyliczenia metryki I świadczącej o niestabilności danej klasy posłużyły wartości dwóch metryk: Ca i Ce. Wartości tej metryki dla pakietów warstwy kontrolera programów BCMExtension i Momentum są równe 1, co jednoznacznie określa te pakiety jako niestabilne. Inaczej rzecz się ma z pakietami warstwy biznesowej obu programów. Ich wartości oscylują pomiędzy 0,5 a 0,64, co uniemożliwia jednoznaczne określenie ich stanu. Są to pakiety, których wartości metryki I wskazują na to, że są to pakiety niestabilne bardziej niż stabilne, wartości I dla tych pakietów nie osiągają wartości 0,7 będącej minimalną optymalną wartością dla pakietów niestabilnych. Bliżej optymalnej wartości dla metryki I są pakiety warstwy biznesowej wersji od 2 do 5 programu BCMExtension w porównaniu z pakietami programu Momentum. Nie osiągnięcie optymalnej wartości metryki I może skutkować dużym oddaleniem od ciągu głównego, czyli wartością metryki Dn.

Kolejną analizowaną metryką z zestawu metryk Roberta C. Martina jest metryka A wskazująca na stosunek klas abstrakcyjnych do wszystkich klas pakietu. Wartości tej metryki wskazują na praktyczną nieobecność klas abstrakcyjnych w pakietach warstwy kontrolera pakietów programu BCMExtension i Momentum. Wartości metryki A dla pakietów warstwy biznesowej obu programów wahają się pomiędzy 0,15 a 0,21, w ostatecznych, piątych wersjach programów osiągają wartości 0,21 dla programu BCMExtension i 0,15 dla programu Momentum.

Analiza danych z rys. 7 doprowadza do wniosku, że każda wersja programu Momentum posiada pakiet warstwy biznesowej przekraczający maksymalną poprawną wartość metryki Dn. Na tym samym wykresie widać, że pakiet warstwy biznesowej, pierwszej wersji programu BCMExtension również przekracza maksymalna poprawną wartość metryki Dn. Pakiety warstwy biznesowej kolejnych wersji programu BCMExtension mieszczą się poniżej granicy maksymalnej dopuszczalnej wartości tej metryki. Warstwa biznesowa programu Momentum powinna zostać poddana analizie oraz przeprojektowaniu, by obniżyć wartość metryki Dn.

6. Wnioski

Celem badań było przeanalizowanie jakości kodu dwóch programów tworzonych przez różne zespoły programistów, których pracę determinowała inna metodyka wytwarzania oprogramowania. Przedstawiona teoria z zakresu metryk jakości oprogramowania, a także dobrych praktyk wytwarzania oprogramowania pozwalają stwierdzić, że istnieją metryki kodu umożliwiające jednoznaczne określenie jego jakości. Celem publikacji było również przeprowadzenie analizy jakościowej dwóch programów oraz późniejsza analiza porównawcza wyników.

Rezultaty badań (tab. 2 i 3) pokazują, że choć wartość niektórych metryk jest podobna to istnieją metryki, gdzie wartości dla programów BCMExtension i Momentum różnią się w sposób znaczący na korzyść programu BCMExtension. Widać to w przypadku metryk RFC, WMC z zestawu metryk CK oraz metryki Dn. Pokazuje to, że programy tworzone z zastosowaniem różnych metod i technik mają różną jakość. Pomimo różnic dotyczących metod zarządzania projektami BCMExtension oraz Momentum nie można jednoznacznie stwierdzić, który z projektów jest jakościowo lepszy. Istnieją metryki takie jak metryki RFC i WMC, pochodzące z zestawu metryk CK, oraz metryka Dn, z zestawu metryk Roberta C. Martina dla pakietu biznesowego, w których program BCMExtension osiąga znacznie lepsze bądź trochę lepsze wyniki od programu Momentum. Z drugiej strony, wartość metryki CBO z zestawu metryk CK jest znacznie gorsza dla projektu BCMExtension. Oba projekty osiągają porównywalne wyniki, jeżeli chodzi o metrykę złożoności cyklomatycznej McCabe'a, metryki LCOM2 i NOC z zestawu metryk CK oraz metrykę Dn dla pakietów warstwy kontrolera z zestawu metryk Roberta C. Martina.

Choć program BCMExtension nie zawsze osiąga lepsze wartości metryk sprawdzających jakość kodu od programu Momentum to z przeprowadzonej analizy można wysnuć wniosek, że kod programu BCMExtension jest łatwiejszy w utrzymaniu i testowaniu niż kod programu Momentum. Wyjaśnieniem nie zawsze lepszych wyników jakości kodu BCMExtension może być nie zawsze stosowanie się do zasad dobrego wytwarzania kodu przez programistów, a co za tym idzie niedokładna weryfikacja jakości kodu przeprowadzana po wytworzeniu nowej funkcjonalności. Zły wpływ na jakość kodu BCMExtension mogą mieć również pozostałości kodu wytworzonego przed stosowaniem metodyki SCRUM.

Tabela 2. Wartości metryk użytych w badaniu dla 5. wersji programów BCMExtension i Momentum.

Metryka	BCMExtension	Momentum
LoC	55666	40005
Śr. CC	1,27	1,36
Śr. LCOM2	0,038	0,028
Śr. NOC	1,27	0,73
Śr. DIT	1,54	1,57
Śr. CBO	14,16	13,22
Śr. RFC	117,55	48,04
Śr. WMC	1,43	3,19
Ca (warstwa biz./kontro.)	212 / 0	172 / 0
Ce (warstwa biz./kontro.)	356 / 1738	263 / 1006
A (warstwa biz./kontro.)	0,21 / 0	0,15 / 0,01
I (warstwa biz./kontro.)	0,63 / 1	0,6 / 1
Dn (warstwa biz./kontro.)	0,16 / 0	0,25 / 1

Tabela 3. Procentowy udział elementów przekraczających optymalną wartość metryk zestawu Roberta C. Martina dla 5. wersji programów BCMExtension i Momentum.

Metryka	BCMExtension [%]	Momentum [%]
LCOM2	0,05	0,07
NOC	0,89	0,67
DIT	0	0
CBO	30,31	21,91
RFC	15,67	19,76
WMC	1,31	3,18

W celu potwierdzenia wpływu metodyki SCRUM na jakość kodu potrzebne są dodatkowe badania obu projektów, z uwzględnieniem przedstawionych wyników. Dobrym pomysłem byłoby również zintegrowanie narzędzi do automatycznej weryfikacji jakości kodu wytworzonej funkcjonalności z projektem BCMExtension, Przyczynić się to może do poprawy jakości kodu wytwarzanego przez zespół programistów oraz zwiększyć świadomość stosowania dobrych praktyk. Mogłoby to również zaowocować pisaniem czystego kodu w szybszym czasie. Badania na ten temat prowadzą obecnie Taibi, Janes, Lenarduzzi [19]. Wyniki metryk dla programu BCMExtension, jeżeli nawet okazywały się gorsze niż wyniki metryk dla programu Momentum, to wykazywały tendencję do poprawy. Może to oznaczać, że po przeprowadzeniu głębszej refaktoryzacji kodu, wszystkie metryki jakości kodu programu BCMExtension będą lepsze niż metryki jakości programu.

Na podstawie tej pracy można wysnuć ogólny wniosek, że:

Stosowanie dobrych praktyk wytwarzania oprogramowania zorientowanego obiektowo podnosi wartość metryk sprawdzających jakość kodu.

Literatura

[1] Čeponis, J., Venčkauskas, A., Čeponienė, L., Zonys, A.: Extending Rule Set for Static Code Analysis in. .NET. Platform. Information Technology And Control, 45, 2016, 99-108.

[2] Iivari J.: The relationship between organizational culture and the deployment of agile methods. Information and Software Technology, 53, 2011, 509-520.

[3] Holmström H., Alahyari H., Bosh J.: Climbing the "Stairway to Heaven" -- A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on, 2012, 392-399.

[4] Vinju J.J., Godfrey M.W.: What Does Control Flow Really Look Like? Eyeballing the Cyclomatic Complexity Metric. Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on, 2012, 154 - 163.

[5] Posnett D., Hindle A., Devanbu P.: A simpler model of software readability. MSR '11 Proceedings of the 8th Working Conference on Mining Software Repositories, 2011, 73-82.

[6] Yousef A.H.: Extracting software static defect models using data mining. Ain Shams Engineering Journal, 6, 2015, 133-144.

[7] Elish M.O., Al.-Yafei A.H., Al.-Mulhem M,: Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of Eclipse. Advances in Engineering Software, 42, 2011, 852-859.

[8] Bluemke I.E., Zając P., Metryki MOOD w systemie Rational Rose w: red. Huzar Z., Mazur Z., Problemy i metody inżynierii oprogramowania, Wydawnictwa Naukowo – Techniczne, Warszawa, 2003.

[9] Martin R.C., Zwinne wytwarzanie oprogramowania. Najlepsze zasady, wzorce i praktyki, Helion, 2015.

[10] Radjenović D., Heričko M., Torkar R., Živkovič A.: Software fault prediction metrics: A systematic literature review. Information and Software Technology, 55, 2013, 1397-1418.

[11] http://www.inmost.org.pl/articles/Metryki_obiektowe_jako_wskaAniki_jakoAci_kodu_i_projektu [04.06.2016].

[12] Szyjewski Z., Muszyńska K., Zarządzanie projektami i modelowanie procesów, Polskie Towarzystwo Informatyczne, Warszawa, 2013.

[13] https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/ [08.06.2016].

[14] https://msdn.microsoft.com/en-us/library/bb385914.aspx [08.06.2016].

[15] Arapidis C.: Sonar Code Quality Testing Essentials: Achieve Higher Levels Of Software Quality With Sonar. Birmingham, Packt Publishing, 2012.

[16] Derezińska A., Rudnik M.: Quality Evaluation of Object-Oriented and Standard Mutation Operators Applied to C# Programs. Lecture Notes in Computer Science, 7304, 2012, 42-57.

[17] Kayarvizhy N., Kanmani S.: An Automated Tool for Computing Object Oriented Metrics Using XML. Advances in Computing and Communications, 191, 2011, 69-79.

[18] Zalewski B.: Metryki oceny jakości oprogramowania i ich stosowalność. Praca magisterska pod kierunkiem Miłosza M., Politechnika Lubelska, Lublin, 2016, 71.

[19] Taibi D., Janes A., Lenarduzzi V.: Towards a Lean Approach to Reduce Code Smells Injection: An Empirical Study. Agile Processes, in Software Engineering, and Extreme Programming, 251, 2016, 300-304.