

Metryki obiektowe jako wskaźniki jakości kodu i projektu

Bartosz Walter

Instytut Informatyki Politechniki Poznańskiej

1 Wprowadzenie

Metryki stanowią szybki i wygodny sposób oceny jakości oprogramowania. Znajomość wartości poszczególnych metryk i wielkości, które mierzą, pozwala oszacować złożoność systemu, koszt jego pielęgnacji czy elastyczność. W artykule przedstawiono trzy najpopularniejsze zestawy metryk obiektowych: MOOD, Chidambers i Kemerera oraz Martina, wraz z interpretacją. Ponadto zaprezentowano przykładowe narzędzie do pomiaru: wtyczkę Metrics Plugin for Eclipse, dostępną za darmo w portalu sourceforge.net.

2 Metryki obiektowe

Metryki obiektowe stanowią odpowiedź na zmianę paradygmatu programowania, która nie znalazła odzwierciedlenia w metodach pomiaru i szacowania dotyczących programowania strukturalnego. Zaproponowane metryki pozwalają mierzyć charakterystyczne dla obiektowości cechy: dziedziczenie, polimorfizm, hermetyzację, powiązania i spójność.

2.1 Zestaw metryk Chidambersa i Kemerera

Zestaw ten został zaproponowany przez S.R. Chidambersa i C.F. Kemerera w 1991 roku [1]. Zaproponowali oni 6 metryk badających różne aspekty obiektowości, które dotychczas nie były badane w kontekście pomiarów oprogramowania: złożoność klasy, dziedziczenie, spójność i powiązania między klasami. Metryki z tego zestawu badają pojedyncze klasy i ich zbiory, zatem stanowią narzędzie do niskopoziomowej oceny jakości kodu obiektowego

- **Response For a Class** (RFC, z ang. *odpowiedź klasy*) określa liczbę komunikatów, które można wysłać do danej. Jej wartość to suma publicznych metod klasy i jej podklas (lub implementacji tego interfejsu). Uznaje się, że wysoka wartość tej metryki wskazuje na duże wewnętrzne skomplikowanie klasy oraz zbyt dużą odpowiedzialność, jaką bierze na siebie ta klasa, co może prowadzić do wysokich kosztów jej testowania i pielęgnacji. Warto zwrócić uwagę, że metryka ta w pewien sposób pomija położenie klasy w hierarchii

dziedziczenia: wysoce abstrakcyjna nadklasa wprawdzie posiada niewiele metod, ale po zsumowaniu z liczbą metod w podklasach wartość metryki jest porównywalna z liczbą metod w jednej z podklas. Typowe wartości mieszczą się w przedziale 20-100 [5].

- **Depth of Inheritance Tree (DIT**, z ang. *głębokość drzewa dziedziczenia*) określa maksymalną liczbę poziomów nadklas, z których rozpatrywana klasa dziedziczy. Wysoka wartość DIT wskazuje, że klasa ta jest specjalizowana, o niskim poziomie abstrakcji, oraz prawdopodobnie posiada wiele metod odziedziczonych po przodkach. Z jednej strony wskazuje to na wysoki stopień powtórnego użycia kodu poprzez dziedziczenie, ale z drugiej może ograniczać wykorzystanie innych mechanizmów, np. kompozycji i wiązania klas przez referencje, dlatego zalecane wartości tej metryki znajdują się w przedziale 2-3 [5].
- **Lack of Cohesion Of Methods (LCOM**, z ang. *brak spójności metod*) to metryka określająca spójność metod wewnątrz klasy. Spójność jest jedną z pozytywnych cech związanych z klasą i oznacza, że metody klasy są silnie związane ze sobą nawzajem i polami zdefiniowanymi w tej klasie. Spójne klasy mają precyzyjnie określony obszar odpowiedzialności i zwykle komunikują się ze światem zewnętrznym przez stosunkowo wąskie interfejsy. Spójność klas należy do najważniejszych czynników decydujących o poprawności projektu obiektowego. Klasyczna metryka LCOM zdefiniowana przez C&K (oznaczana często jako LCOM1) mierzy brak odwołań do wspólnych atrybutów klasy i przyjmuje wartość równą różnicy liczby metod odwołujących się do wspólnego atrybutu klasy oraz liczby metod o rozłącznych zbiorach wykorzystywanych atrybutów. Posiada ona jednak pewne wady (m.in. zliczanie jedynie wspólnych pól a nie wywoływanych metod oraz podawanie tej samej wartości dla całkowicie odmiennych klas), które powodują jej niewrażliwość na pewne własności badanego projektu. Dlatego zaproponowano kilka innych definicji, wśród nich m.in. metrykę LCOM3, zaproponowaną przez Hendersona i Sellersa [6], która przyjmuje wartości od 0 do 2. Za anomalię uznaje się wartości przekraczające 1 – klasy o takim wskaźniku powinny zostać podzielone na mniejsze.
- **Coupling Between Objects (CBO**, z ang. *powiązania pomiędzy obiektami*) jest drugą, obok LCOM, ważną metryką oceniającą jakość projektu obiektowego. Niski stopień powiązań klas między sobą (ang. *coupling*) wskazuje na ścisłe określenie granic pomiędzy klasami i sposobu ich komunikowania się. Ponadto zwiększa stopień abstrakcji projektu, ponieważ łatwiej modyfikować fragmenty kodu w nieznacznym stopniu zależne od pozostałych. Metryka CBO zlicza liczbę klas związanych z klasą rozpatrywaną w sposób inny niż poprzez dziedziczenie (a więc jako atrybuty klasy, parametry metod, wartości zwracane, klasy wyjątków etc.). Wartość tej metryki powinna być utrzymana na możliwie niskim poziomie (oczywiście, wartość 0 oznaczałaby, że klasy w ogóle nie odwołują się do siebie, co jest niemożliwe). Typowo wynosi ona do 5 [5]
- **Weighted Method per Class (WMC**, z ang. *ważona liczba metod w klasie*) jest miarą złożoności klasy. W podstawowej wersji jest liczbą wszystkich

metod zdefiniowanych w klasie (czyli waga każdej metody jest równa 1), w wersji rozszerzonej wagi przypisywane metodom są obliczane na podstawie złożoności cyklomatycznej CC McCabe'a). Metryka WMC pozwala na precyzyjniejszą ocenę niż RFC, ponieważ uwzględnia złożoność każdej metody. Typowe wartości to 0-100 [5]

- **Number of Children** (NOC, z ang. *liczba klas pochodnych*) jest liczbą bezpośrednich potomków danej klasy lub liczbą klas implementujących dany interfejs. Wysoka wartość tej metryki wskazuje na nieprawidłową faktoryzację nadklasy lub interfejsu (jest ona wysoce abstrakcyjna i prawdopodobnie obejmuje zbyt duży obszar odpowiedzialności) i może powodować trudności z jej pielęgnacją.

2.2 Zestaw metryk MOOD

Metryki oznaczone akronimem MOOD zostały zaproponowane przez F. B. e Abreu w 1995 roku [2]. W skład zestawu wchodzi 6 metryk charakteryzujących najbardziej widoczne elementy projektu obiektowego: hermetyzację, dziedziczenie, polimorfizm i powiązania między klasami.

Wszystkie metryki MOOD posiadają pewne cechy wspólne. Są one zdefiniowane jako ilorazy rzeczywistego wykorzystania pewnego mechanizmu do maksymalnego możliwego poziomu. Dzięki temu są to wartości niemianowane, wyrażane w procentach, i (z założenia) nie zależą (przynajmniej bezpośrednio) od rozmiaru badanego systemu i języka jego implementacji. Z tego powodu pozwalają na szybką i zgrubną ocenę całego projektu, co jest przydatne zwłaszcza dla poziomu zarządzania.

- **Attribute Hiding Factor** (AHF, z ang. *współczynnik ukrycia atrybutów*) określa stopień hermetyzacji atrybutów klasy. Ograniczenie dostępu do klasy wyłącznie do jej metod jest ogólnie znanym postulatem, dlatego współczynnik AHF powinien przyjmować wysokie wartości, w optymalnym przypadku nawet 100%. W praktyce wartość ta dla kilku znanych produktów wyniosły od 70 do 95%.
- **Method Hiding Factor** (MHF, z ang. *współczynnik ukrycia metod*) określa liczbę stopień hermetyzacji metod klasy. Sytuacja jest nieco odmienna od metryki AHF, ponieważ 100% wartość oznaczałaby, że klasa jest całkowicie niedostępna dla innych klas, a więc nie mogą one się z nią komunikować. Z drugiej strony, oczywiście, każda klasa zawiera grupę metod, które powinny być ukryte i niedostępne. Z tych powodów wartość tej metryki waha się od 10 do 40%.
- **Attribute Inheritance Factor** (AIF, z ang. *współczynnik dziedziczenia atrybutów*) określa wykorzystanie mechanizmu dziedziczenia i stanowi stosunek odziedziczonych atrybutów do wszystkich atrybutów obecnych w klasie. Niska wartość tej metryki wskazuje na niski stopień powtórnego wykorzystania kodu poprzez dziedziczenie. Wartości zbliżone do 100% oznaczają często zbyt skomplikowane hierarchie dziedziczenia i nadużycie tego mechanizmu. W typowych projektach wartość oscyluje pomiędzy 40 a 80%.

- **Method Inheritance Factor (MIF)**, z ang. *współczynnik dziedziczenia metod*) służy do oceny stopnia dziedziczenia metod. Podobnie jak wskaźnik AIF, MIF powinien przyjmować wartości pośrednie, pomiędzy 60 a 80%.
- **Polymorphism Factor (PF)**, z ang. *współczynnik polimorficzności*) wskazuje, jaka część metod jest pokrywana w podklasach. Wartość ta powinna być stosunkowo niska, ale wyższa od zera. Wartości bliskie zera wskazują na strukturalny projekt, który nie wykorzystuje możliwości języka obiektowego, natomiast wysokie – na zbyt skomplikowane powiązania pomiędzy klasami i nieprawidłową faktoryzację.
- **Coupling Factor (CF)**, z ang. *współczynnik powiązań*) ocenia stopień zależności poszczególnych klas od siebie. Pełni on podobną rolę jak CBO z zestawu C&K, pozwalając określić stopień wypełnienia grafu zależności. Niska wartość wskazuje, że odpowiedzialność jest nieprawidłowo podzielona pomiędzy klasy i kilka z nich realizuje większość funkcjonalności. Wysoka wartość CF też sugeruje złe zaprojektowanie systemu, z wieloma zależnościami pomiędzy klasami, co znacznie ogranicza możliwości rozbudowy i podwyższa koszt pielęgnacji kodu. Wartości powinny należeć do przedziału 5 do 20%.

2.3 Zestaw metryk R. Martina

Metryki autorstwa R. Martina [3], w odróżnieniu od poprzednich dwóch zestawów, przede wszystkim rozpatrują pojęcie zależności (powiązania) jednej klasy od drugiej. Martin wyróżnił zależności do wewnątrz i na zewnątrz, co pozwoliło wskazać klasy stanowiące rdzeń aplikacji i klasy wykonawcze, zależne od tego rdzenia. Klasy takie różnią się odpowiedzialnością i niezależnością, dlatego w oparciu o te pojęcia zdefiniowano następujące metryki:

- **Affarent coupling (Ca)**, z ang. *powiązania do wewnątrz*) to liczba typów, które zależą od rozpatrywanego modułu. Klasa o wysokiej wartości metryki Ca jest obciążona dużą odpowiedzialnością wobec innych modułów, co oznacza, że zmiany w niej powodują konieczność modyfikacji zależnych od niej typów.
- **Efferent coupling (Ce)**, z ang. *powiązania na zewnątrz*) określa zależność rozpatrywanego modułu od modułów zewnętrznych. Wartością tej metryki jest liczba typów, od których zależy dany typ. Zmiany w klasach o wysokiej wartości tej metryki w niewielkim stopniu wpływają na zewnętrzne moduły, natomiast zmiany w nich z dużym prawdopodobieństwem będą miały wpływ na rozpatrywany moduł.
- **Abstractness (A)**, z ang. *abstrakcja*) to miara abstrakcyjności pakietu i jest mierzona stosunkiem liczby typów abstrakcyjnych wchodzących w skład tego pakietu do wszystkich typów. Klasy o wysokim wskaźniku A trudno podlegają zmianom, ponieważ są odpowiedzialne wobec typów zależnych od nich.
- **Instability (I)**, z ang. *niestabilność*) jest miarą niestabilności pakietu, czyli wpływu zewnętrznych typów na rozpatrywaną klasę. Jest ona obliczana jako

stosunek wartości metryki Ce do sumy wszystkich zależności (Ce + Ca). Wartości I bliskie jedynce sugerują, że klasa jest podatna na zmiany w zewnętrznych modułach (powiązania na zewnątrz stanowią niemal 100% wszystkich powiązań), natomiast wartości bliskie zeru charakteryzują klasę trudno poddającą się zmianom z uwagi na ich wpływ na pozostałe typy. Metryka ta powinna przyjmować właśnie wartości ekstremalne: albo bliskie zeru (czyli rdzeniowe klasy typu kontroler), albo bliskie jedynce (klasy korzystające z funkcjonalności dostarczanej przez inne moduły).

- **Normalized distance from main sequence** (Dn, z ang. *znormalizowana odległość od ciągu głównego*) określa odległość od optymalnej linii łączącej punkty (1, 0) i (0, 1) w układzie współrzędnych metryk A i I. Klasy o niskiej wartości A powinny jednocześnie być niestabilne – i na odwrót, ponieważ to świadczy o dobrej kompensacji tych dwóch cech. Odchylenie od tej linii wskazuje na niepoprawne przypisanie odpowiedzialności do danej klasy.

2.3 Złożoność cyklomatyczna McCabe'a

Złożoność cyklomatyczna, zaproponowana przez McCabe'a w 1976 roku [7] jest metryką strukturalną, która odzwierciedla złożoność grafu przepływu sterowania w metodzie. Jej wartość jest równa liczbie niezależnych ścieżek wykonawczych w procedurze. Ponieważ obiektem pomiaru jest właśnie procedura, dlatego metryka ta nie może być bezpośrednio zastosowana w środowisku obiektowym (nie uwzględnia ona dziedziczenia i pokrywania metod), jednak jest z powodzeniem stosowana jako element bardziej obiektowych miar (np. WMC).

3 Wtyczka Metrics for Eclipse

Przykładowym narzędziem do pomiaru kilkunastu metryk obiektowych jest wtyczka Metrics for Eclipse, stworzona dla popularnego środowiska dla języka Java Eclipse. Pozwala ona na automatyczny pomiar większości wymienionych wcześniej metryk obiektowych (rys. 1), a także na podanie ich dopuszczalnych wartości, których przekroczenie powoduje wygenerowanie ostrzeżenia. Dzięki integracji z Eclipse metryki są obliczane przyrostowo równocześnie z kompilacją kodu, dzięki czemu obliczenia są realizowane bardzo wydajnie.

Metric	Total	Mean	Std. Dev.
+ Number of Static Methods (avg/max per type)	1	0,071	0,258
+ Total Lines of Code	365		
+ Affherent Coupling (avg/max per packageFragment)		1,5	1,5
+ Normalized Distance (avg/max per packageFragment)		0,409	0,409
+ Number of Classes (avg/max per packageFragment)	14	7	2
+ Specialization Index (avg/max per type)		0,028	0,079
+ Instability (avg/max per packageFragment)		0,5	0,5
+ Number of Attributes (avg/max per type)	17	1,214	1,698

Rys. 1. Przykładowy fragment raportu

Literatura

1. S.R. Chidamber, C.F. Kemerer, *A metrics suite for object-oriented design*. IEEE Transactions on Software Engineering, Vol. 20, No 6, pp. 476-493
2. F. B. Abreu, *The MOOD Metrics Set*. ECOOP 1995 Workshop on Metrics.
3. R. Martin, *OO Design Quality Metrics*. <http://www.objectmentor.com/publications/oodmetric.pdf>
4. *Eclipse Metrics Plugin*. <http://metrics.sourceforge.org>
5. NASA SATC, *Applying and interpreting OO Metrics*. http://satc.gsfc.nasa.gov/support/STC_APR98/apply_oo/apply_oo.html
6. B. Henderson-Sellers, *Object-Oriented Metrics, measures of Complexity*. Prentice Hall, 1996.
7. T. McCabe, *A Software Complexity Measure*. IEEE Transactions on Software Engineering, 1976.