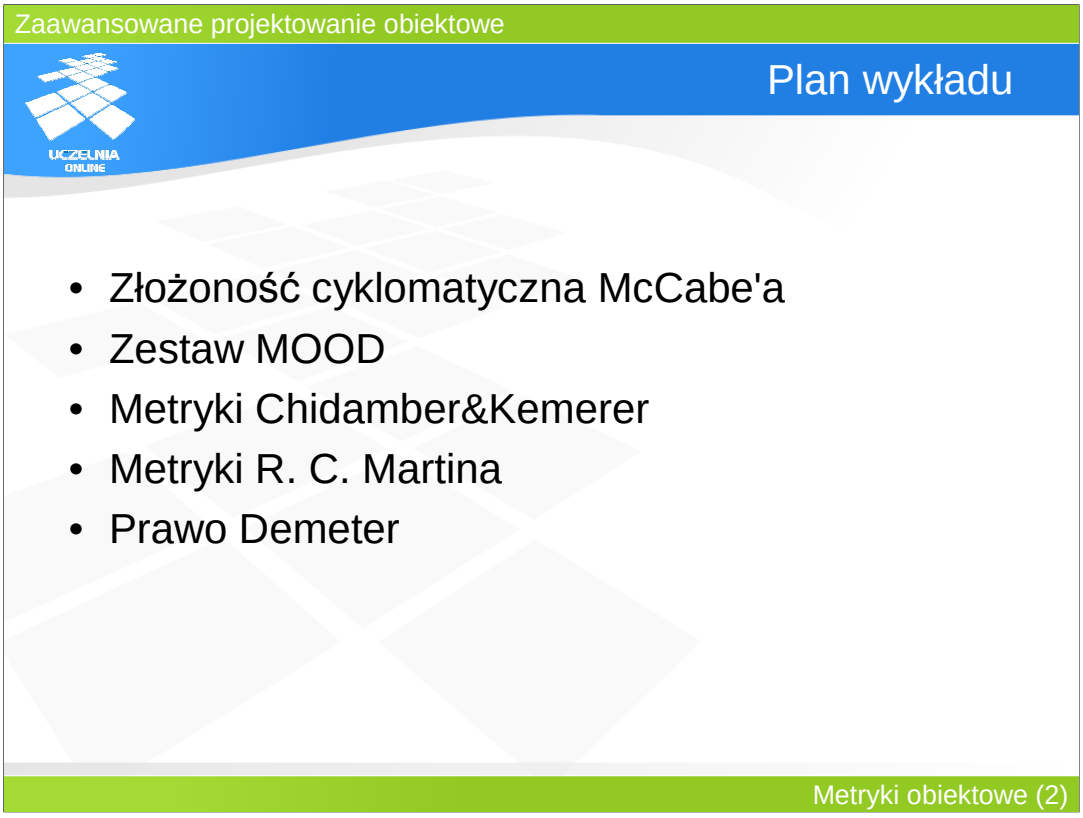

Zaawansowane projektowanie obiektowe
Metryki obiektowe
Prowadzący: Bartosz Walter
UCZELNIA
ONLINE



Wykład jest poświęcony ilościowej ocenie jakości programów. Podczas wcześniejszych wykładów mówiliśmy jedynie o zasadach projektowania obiektowego, wskazując co jest dobrym rozwiązaniem, a co złym. Jednak wówczas omówiono tylko podstawowe dwa kryteria oceny projektu obiektowego: spójności i powiązaniach, i to jedynie w kategoriach jakościowych. Dlatego obecny wykład wprowadza metryki obiektowe, służące do obiektywnej i ilościowej oceny jakości projektu obiektowego.

Podczas wykładu zostanie przypomniana metryka złożoności programu McCabe'a oraz wprowadzone trzy zestawy metryk: MOOD autorstwa e Abreu, zestaw Chidambera i Kemerera oraz metryki zaproponowane przez R. Martina. Pozornie te trzy zestawy metryk dotyczą tego samego zagadnienia, a zatem są redundantne. Jednak bliższa ich poznanie pozwala stwierdzić, że każdy zestaw kieruje się nieco innymi zasadami konstrukcji i interpretacji tych metryk – dlatego warto poznać je wszystkie.

Ostatnim elementem wykładu będzie przedstawienie prawa Demeter, określającego dopuszczalny stopień powiązań między obiektami.

Złożoność cyklomatyczna McCabe'a

CC służy do oceny wewnętrznej złożoności metody

$$CC = \overline{E} - \overline{V} + 2$$

$$CC = d - 1$$

E – liczba krawędzi grafu (możliwych przejść)

V – liczba wierzchołków w grafie (instrukcji)

d – liczba węzłów decyzyjnych w grafie

CC jest liczbą niezależnych ścieżek w grafie przepływu sterowania metody

McCabe (1976)

Złożoność cyklomatyczna (CC), mimo swojej długiej historii – została zdefiniowana w 1976 roku z myślą o programowaniu strukturalnym – jest nadal podstawową miarą złożoności dowolnego fragmentu kodu.

Ogólna definicja mówi, że CC to liczba niezależnych ścieżek w grafie reprezentującym przepływ sterowania w metodzie. Istnieją dwa wzory określające tę wartość: pierwszy opiera się na liczbie wierzchołków i łuków w grafie, natomiast drugi – jedynie na liczbie węzłów decyzyjnych.

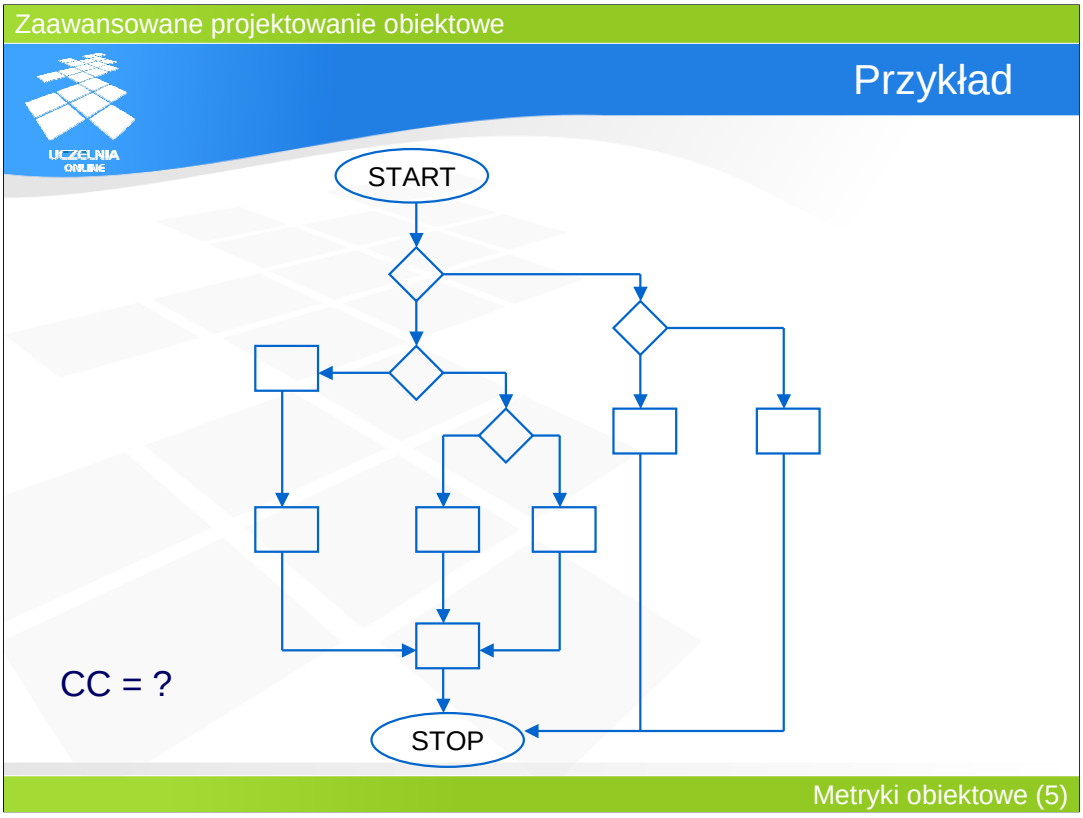
Istnieją następujące interpretacje wartości tej metryki

- niższa wartość *CC* może wskazywać na lepszą czytelność metody lub odsunięcie w niej decyzji poprzez delegację, jednak niekoniecznie dowodzi, metoda jest prosta;
- CC służy do pomiaru złożoności metod, a nie klas (metody mogą zostać odziedziczone), ale w powiązaniu z innymi metrykami może posłużyć do oceny klas;
- wartości przekraczające 20 wskazują na zbytnią złożoność badanej metody.

Interpretacja metryki CC

CC	Interpretacja
1 – 10	prosta metoda
11 – 20	metoda złożona
21 – 50	metoda bardzo złożona
> 50	testowanie niemal niemożliwe

Optymalne wartości tej metryki wynoszą poniżej 50. Należy jednak pamiętać, że są to wartości podane dla programowania strukturalnego, dlatego w przypadku programów obiektowych warto przyjąć nieco obniżony próg.



Na slajdzie przedstawiono graf sterowania pewnej metody. Aby przypomnieć sobie sposób obliczania metryki CC, proszę obliczyć ją korzystając z obu wzorów podanych na poprzednim slajdzie. Owalami oznaczono stany początkowy i końcowy, rombami – punkty decyzyjne, natomiast instrukcje prostokątami.

Zestaw MOOD

- Zdefiniowane przez F. Brito e Abreu w 1995 r
- Zestaw 6 metryk
- Ocena całego systemu, a nie pojedynczych elementów
- Bezwymiarowe, wyrażane w procentach
  - licznik: rzeczywiste wykorzystanie mechanizmu
  - mianownik: maksymalne możliwe wykorzystanie mechanizmu
- Niezależne od rozmiaru oprogramowania
- Niezależne od języka programowania

Pierwszym omawianym zestawem metryk jest MOOD – Metrics for Object-Oriented Design. Został on zaproponowany przez F. Brito e Abreu w 1995 roku. Składa on się z 6 metryk, których celem jest ocena stopnia wykorzystania poszczególnych mechanizmów obiektowych: hermetyzacji, dziedziczenia, powiązań i polimorfizmu oraz związku tych wielkości z zewnętrznymi atrybutami oprogramowania: jakością, testowalnością etc.

Ich charakterystyczną cechą jest bezwymiarowość – są one wyrażone w procentach reprezentujacych stopień wykorzystania badanej cechy w systemie. Licznik procentu odzwierciedla faktyczny stopień użycia mechanizmu, natomiast mianownik – maksymalny, teoretycznie możliwy stopień. Obiektem pomiaru nie są zatem pojedyncze klasy i ich relacje, ale cały system, dlatego metryki te pozwalają oceniać projekt obiektowy na znacznie wyższym poziomie abstrakcji niż poziom klasy.

Co ważne, metryki te dobrze się skalują (ich wartości z założenia nie zależą od rozmiaru badanego oprogramowania), a także nie zależą od języka programowania (możliwe jest porównanie wyników metryk dla języka strukturalnego i języka obiektowego – oczywiście jeżeli projekt systemów stworzonych w tych językach ma cechy obiektowości).

Metryki z zestawu MOOD

Podstawowe mechanizmy programowania obiektowego

polimorfizm

- Polymorphism Factor (PF)

hermetyzacja

- Attribute Hiding Factor (AHF)
- Method Hiding Factori (MHF)

dziedziczenie

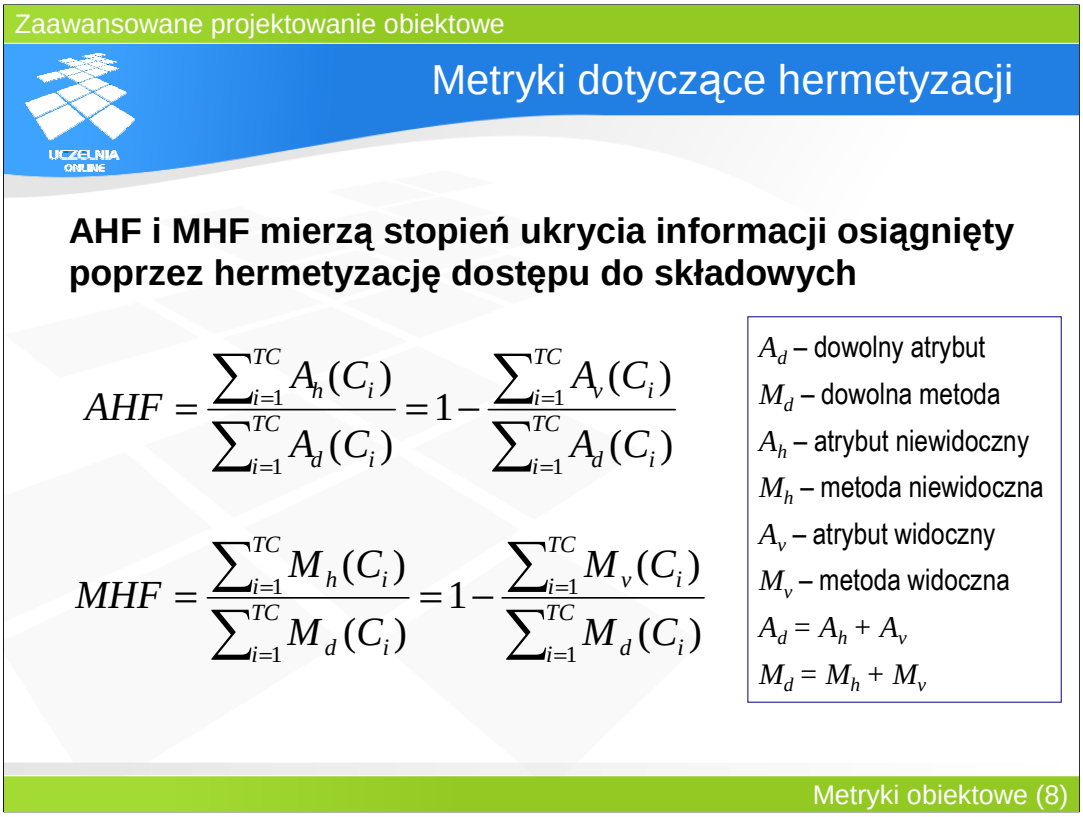
- Attribute Inheritance Factor (AIF)
- Method Inheritance Factor (MIF)

przekazywanie komunikatów

- Coupling Factor (CF)

Metryki wchodzące w skład tego zestawu badają cztery kluczowe mechanizmy obiektowe: polimorfizm, hermetyzację, dziedziczenie oraz wysyłanie i obsługa komunikatów. W przypadku hermetyzacji i dziedziczenia istnieją osobne metryki służące do oceny badanej wielkości w odniesieniu do atrybutów (pól) i metod.

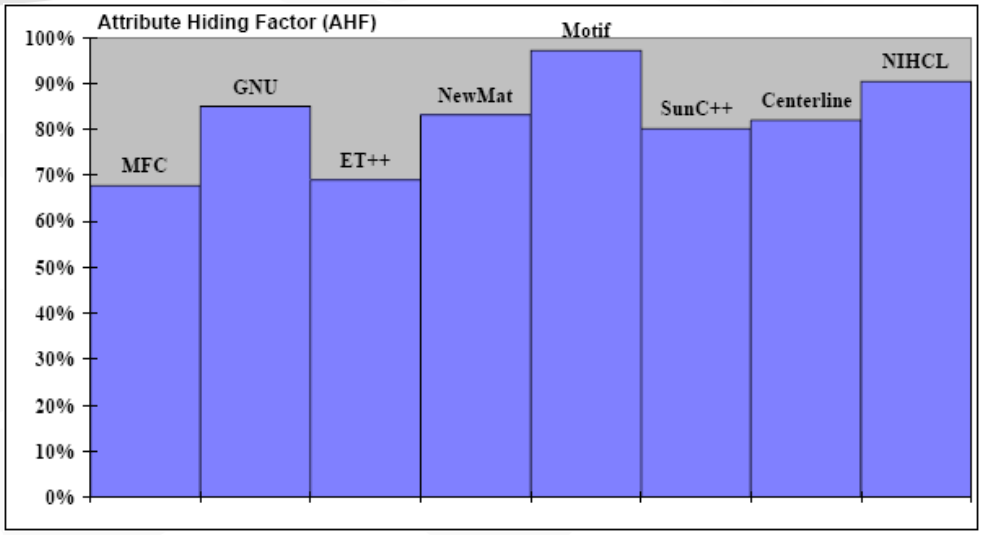
Celem twórcy tych metryk była ogólna ocena jakości całego systemu, a nie pojedynczych jego elementów: klas lub komponentów. Metryki te i ich interpretacja okazują się (zresztą zgodnie z intuicją) zawodne w przypadku systemów w dużej mierze opartych o formularze, generację kodu i moduły współdzielone przez wszystkie elementy systemu, w których przydział odpowiedzialności do klas zachodzi w oparciu o inne kryteria.



Dwie metryki hermetyzacji: AHF dla atrybutów i MHF dla metod, oceniają stopień ukrycia składowych. Wszystkie atrybuty/metody znajdujące się w całym systemie są dzielone na dwie klasy: widocznych i niewidocznych dla klientów.

Obie metryki mają podobną strukturę: w liczniku znajduje się liczba atrybutów/metod widocznych (czyli takich, do których można poprawnie się odwołać) dla klientów, natomiast w mianowniku – liczba wszystkich atrybutów/metod we wszystkich klasach rozważanego systemu. Ich wartość jest zatem procentowym udziałem atrybutów/metod widocznych w całym systemie

Attribute Hiding Factor (AHF)

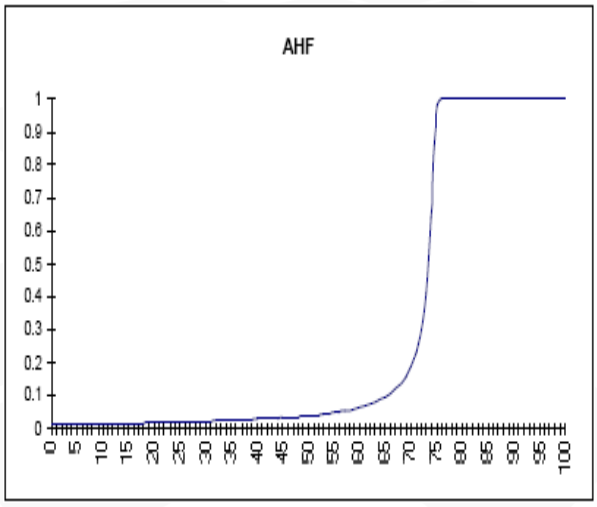


F.B. e Abreu (1995)

Wykres ten przedstawia wartości metryki AHF zmierzone dla kilku znanych systemów obiektowych o dostępnym kodzie źródłowym. Warto zauważyć, że systemy te zostały stworzone przy pomocy różnych języków programowania, w tym także nieobiektowych (np. Motif). Wynika z tego, że wartości uzyskane za pomocą zestaw MOOD faktycznie nie zależą od zastosowanego języka programowania.

W tym przypadku współczynnik ukrycia atrybutów wyniósł pomiędzy 67 (MFC) i 96 procent (Motif). Zatem można się spodziewać, że w dobrze zaprojektowanych systemach metryka ta nie powinna przyjmować wartości niskich.

Optymalne wartości metryki AHF



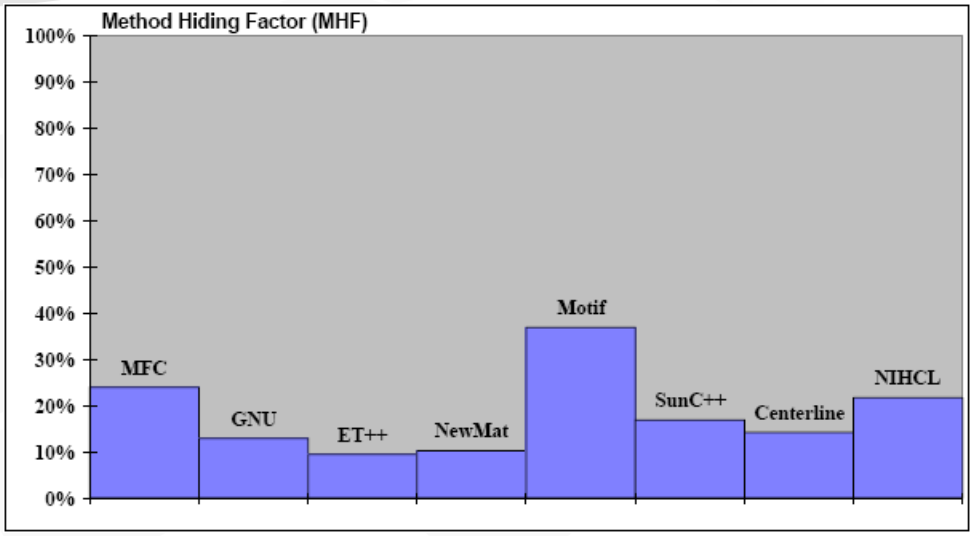
Atrybuty powinny być ukryte w jak największym stopniu.

Optymalnym rozwiązaniem jest brak bezpośredniego dostępu do atrybutów.

F.B. e Abreu (1995)

Optymalny rozkład tej metryki przyjmuje przedstawiony na wykresie kształt: jednym z kryteriów dobrego projektu jest ukrycie jak największej liczby pól, i ew. udostępnienie ich poprzez metody. Optymalnym rozwiązaniem byłoby osiągnięcie pełnego ukrycia wszystkich atrybutów, co jednak w praktyce, ze względu na rzeczywistą jakość projektu lub wygodę programowania, okazuje się niemożliwe lub bardzo trudne do osiągnięcia.

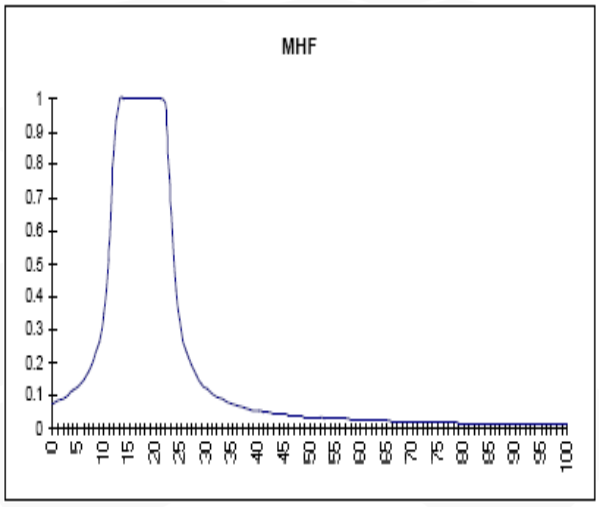
Method Hiding Factor (MHF)



F.B. e Abreu (1995)

Na slajdzie przedstawiono zmierzone w rzeczywistych projektach wartości drugiej z metryk dotyczącej hermetyzacji – MHF. Wahają się one od 10% (ET++) do 37% (Motif). Płynie stąd wniosek, że w przypadku tej metryki 100% nie jest optymalną wartością.

Optymalne wartości metryki MHF



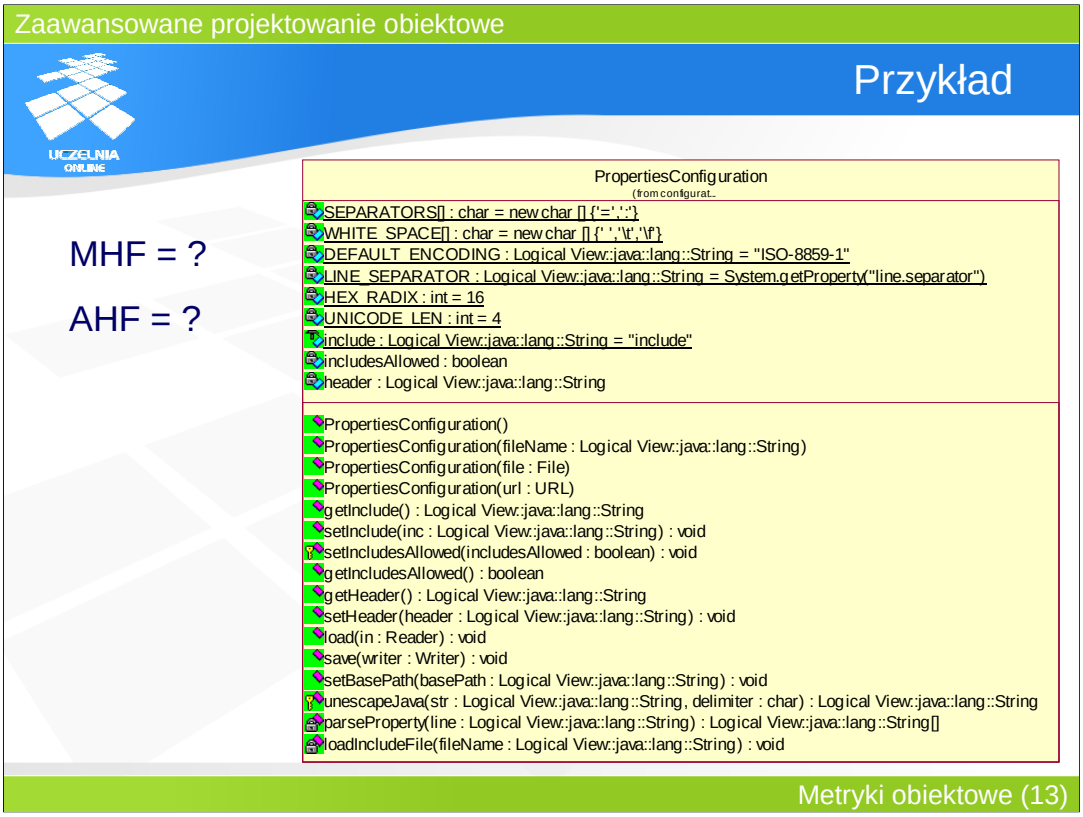
Niski poziom ukrycia metod wskazuje na niewystarczająco abstrakcyjną implementację.

Wysoki poziom – niską funkcjonalność systemu i jego klas składowych.

F.B. e Abreu (1995)

Na wykresie przedstawiono optymalne wartości metryki MHF. Zgodnie z intuicją opisaną na poprzednim slajdzie, celem nie jest osiągnięcie ani zbyt wysokiego, ani zbyt niskiego ukrycia składowych, a pożądane wartości mieszą się w przedziale 10%-25%.

Obserwację tę można interpretować następująco: niski poziom ukrycia oznacza, że większość metod jest dostępna dla każdego klienta. Wskazuje to na niewystarczająco skuteczne wykorzystanie hermetyzacji jako mechanizmu abstrakcji. Z drugiej strony skuteczne ukrycie metod przed klientem uniemożliwia mu dostęp do nich i ich wywołanie – zatem z punktu widzenia użyteczności klasa o małej liczbie dostępnych metod posiada małą funkcjonalność.



Dla przykładu spróbujmy obliczyć wartości metryk MHF i AHF dla klasy PropertiesConfiguration z biblioteki Configuration będącej produktem projektu Jakarta Commons. Metody publiczne są zaznaczone jedynie różowym prostokątem, natomiast pola publiczne – niebieskim prostokątem (bez innych dekoracji).

Metryki dotyczące hermetyzacji

AHF i MHF mierzą stopień wykorzystania dziedziczenia

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)} = 1 - \frac{\sum_{i=1}^{TC} A_d(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} = 1 - \frac{\sum_{i=1}^{TC} M_d(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

A_a – dowolny atrybut
M_a – dowolna metoda
A_d – atrybut zdefiniowany
M_d – metoda zdefiniowana
A_i – atrybut odziedziczony
M_i – metoda odziedziczona
$A_a = A_i + A_d$
$M_a = M_i + M_d$

Kolejne dwie metryki dotyczą innego aspektu obiektowości – dziedziczenia. Metryka ta zatem wskazuje, w jakim stopniu dziedziczenie atrybutów i metod jest wykorzystane w badanym systemie.

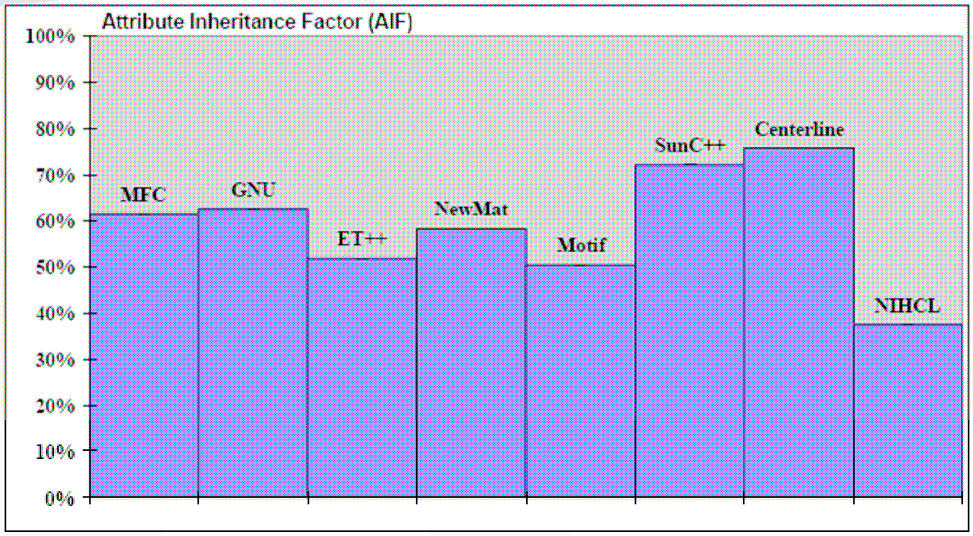
Dziedziczenie pozwala wyrazić trzy różne relacje pomiędzy klasami:

- ich podobieństwo
- relację generalizacji i specjalizacji
- powtórne użycie kodu nadklasy przez podklasy.

Metryki zostały skonstruowane w sposób analogiczny do metryk dotyczących hermetyzacji. Są ułamkiem, w którego liczniku znajduje się faktyczne liczba atrybutów/metod odziedziczonych we wszystkich klasach całego systemu do całkowitej liczby atrybutów/metod w systemie. Oznacza to, że metryki są niemianowane i przyjmują wartości od 0 do 100%.

Należy zwrócić uwagę, że dziedziczenie jest jednym z narzędzi oferowanych przez obiektowe języki programowania, ale nie jest bezpośrednio kryterium oceny jakości projektu. Dlatego wartości tych dwóch metryk nie niosą bezpośrednio informacji o jakości projektu.

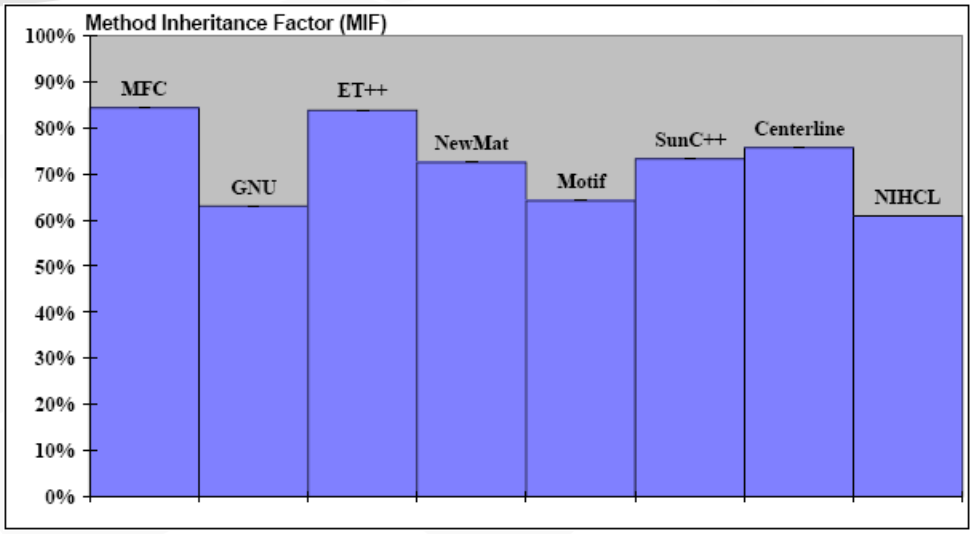
Attribute Inheritance Factor (AIF)



F.B. e Abreu (1995)

Wykres przedstawia wartości metryki AIF zmierzone w kilku systemach obiektowych. Wartości tej metryki policzone dla wybranych projektów obiektowych wskazują ponownie, że nie przyjmuje ona wartości bardzo niskich ani bardzo wysokich. Zmierzony stopień dziedziczenia atrybutów dla tych projektów wyniósł od 40% (NIHCL) do 70% (Centerline). Intuicja wskazuje, że dziedziczenie atrybutów jest stosowane w mniejszym stopniu niż dziedziczenie metod, a więc można oczekiwać, że metryka MIF będzie przyjmowała wyższe wartości.

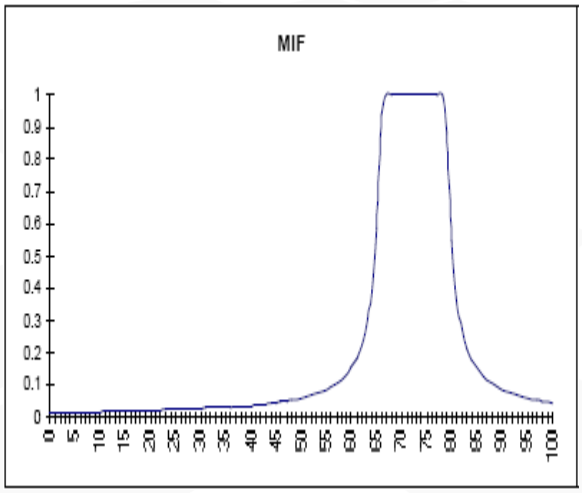
Method Inheritance Factor (MIF)



F.B. e Abreu (1995)

Wykres ten przedstawia zmierzone wartości współczynnika dziedziczenia dla metod. Faktycznie, wartości tej metryki są wyższe niż w przypadku dziedziczenia atrybutów. Ma to intuicyjne wyjaśnienie: dziedziczenie metod dotyczy podobieństwa w zachowaniu klas, natomiast dziedziczenie atrybutów jest związane z podobieństwem w strukturze klas. Dlatego metody są zwykle dziedziczone przez wiele poziomów w hierarchii dziedziczenia, od klas abstrakcyjnych do konkretnych, natomiast atrybuty zwykle występują jedynie na poziomach dziedziczenia odpowiadających bardziej konkretnym klasom.

Optymalne wartości metryki MIF



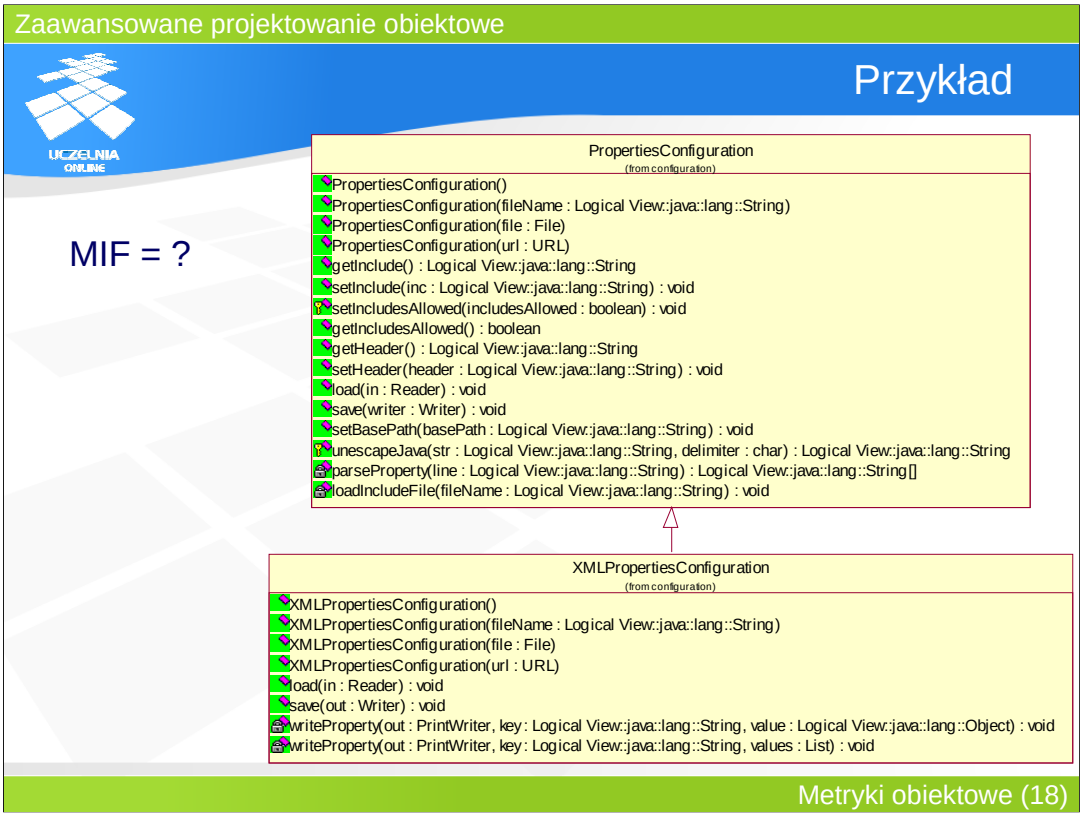
Niski współczynnik dziedziczenia metod wskazuje na niskie powtórne użycie kodu.

Wysoki współczynnik oznacza skomplikowaną hierarchię dziedziczenia

F.B. e Abreu (1995)

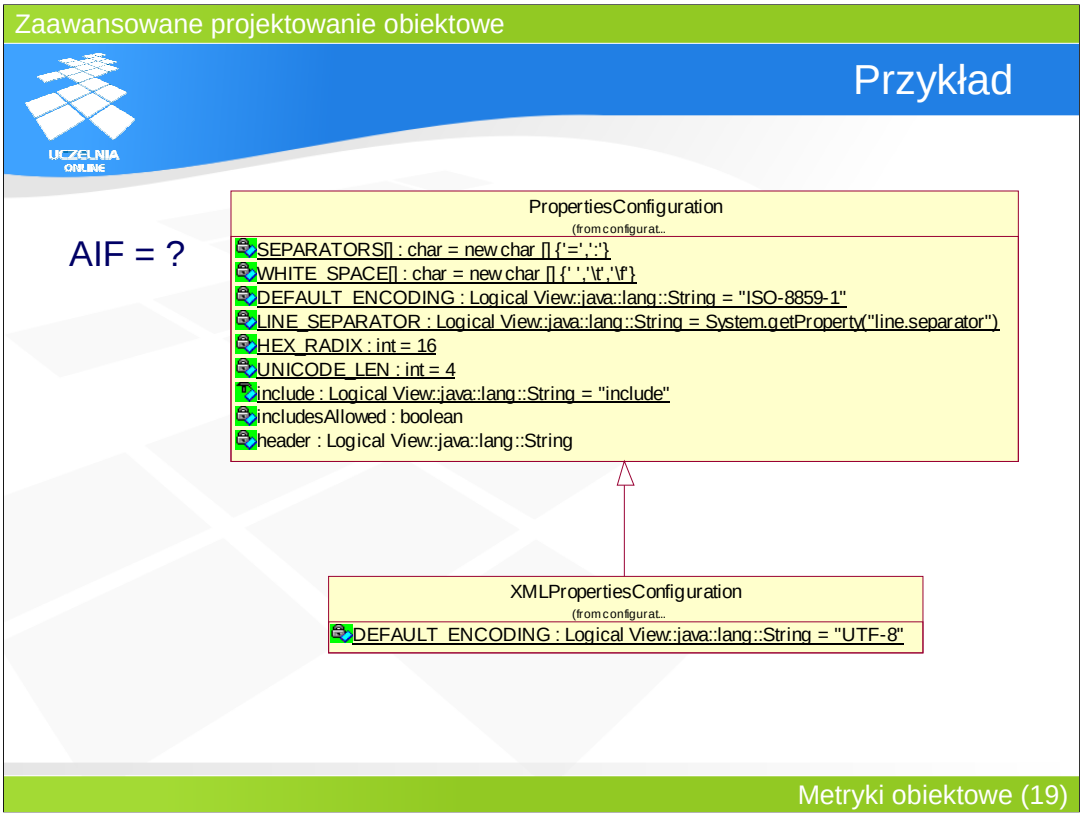
Tę zależność można zauważyć na przedstawionym wykresie: oczekiwaną wartością metryki jest 60%-80%, a więc więcej niż w przypadku dziedziczenia atrybutów.

Zbyt małe wartości MIF są interpretowane jako zbyt małe powtórne użycie kodu, co może objawiać się np. w dużej liczbie powielonych fragmentów programu. Z kolei wartości przekraczające 80% wskazują na skomplikowaną hierarchię dziedziczenia: duża część relacji pomiędzy klasami to właśnie dziedziczenie, co – jak już powiedziano na pierwszym wykładzie – zmniejsza elastyczność systemu i jego otwartość na zmiany. Zatem optymalne wartości znajdują pomiędzy tymi skrajnymi, przy czym z reguły wartość MIF jest wyższa niż AIF.

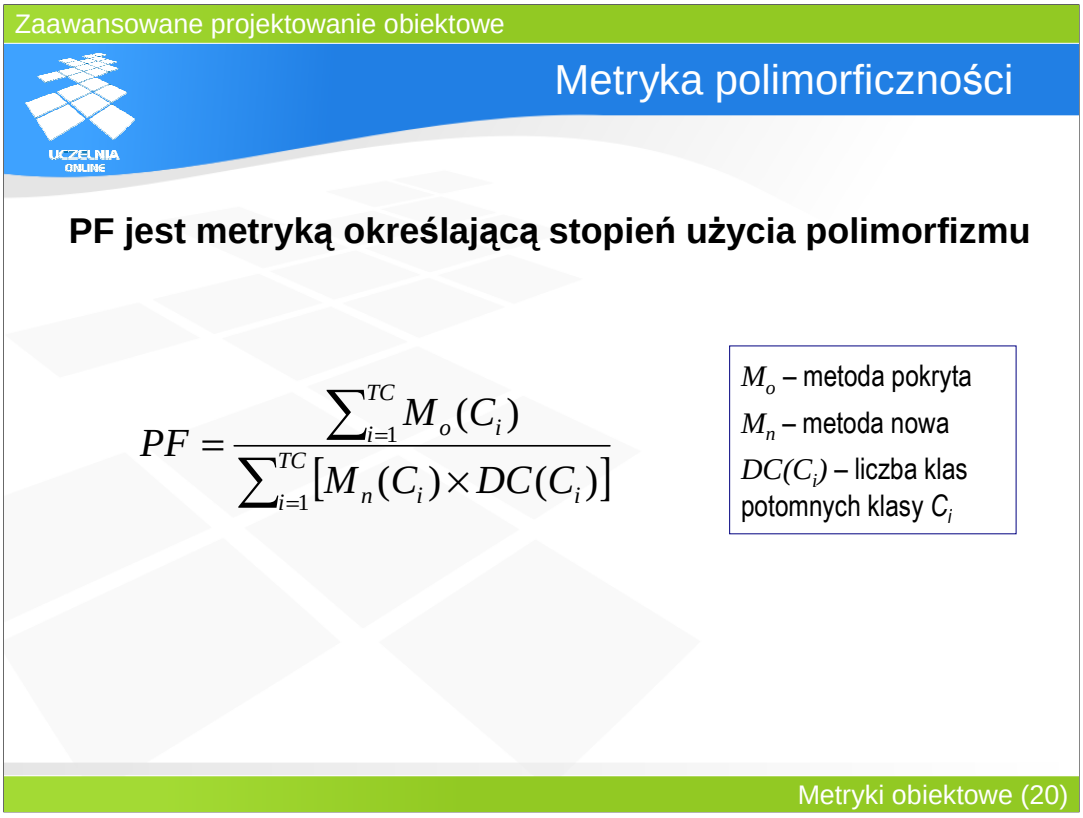


Ponownie, przykład jest zaczerpnięty z biblioteki Configuration będącej jedynym z produktów projektu Apache Jakarta Commons. Dwie klasy: PropertiesConfiguration i XMLPropertiesConfiguration są związane ze sobą relacją dziedziczenia. Na diagramie zaznaczono wszystkie metody zdefiniowane w danej klasie, zatem pominięto metody odziedziczone.

Zadanie polega na obliczeniu metryki MIF dla tych dwóch klas.



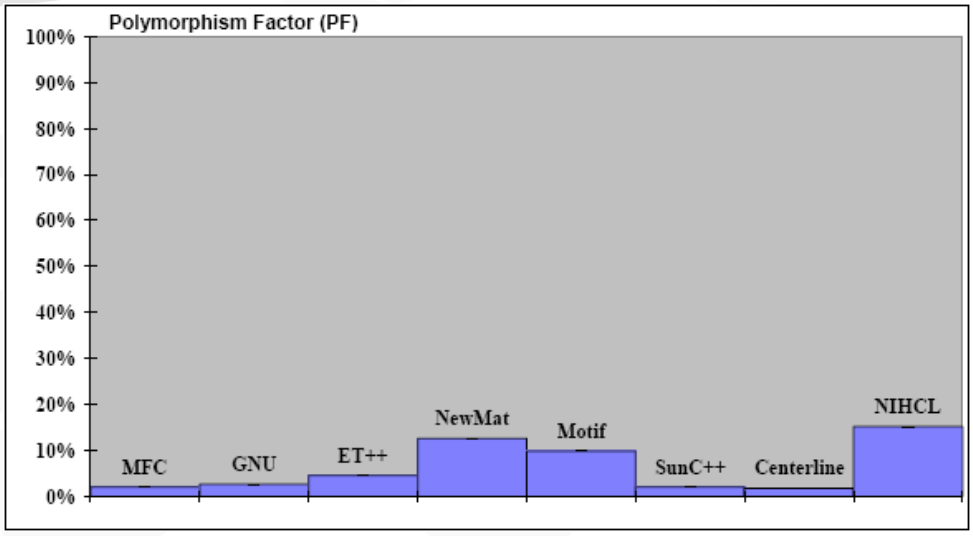
Drugi przykład, pochodzący z tego samego źródła, dotyczy dziedziczenia atrybutów w tych samych klasach. Zadanie polega na obliczeniu wartości metryki AIF dla tego układu klas.



Kolejną metryką należącą do zestawu MOOD jest PF – współczynnik wykorzystania polimorfizmu. Określa on, w jakim stopniu zastosowane jest polimorficzne pokrywanie metod. Ponownie, metryka ta jest stosunkiem liczby metod pokrytych w całym systemie do wszystkich metod, które mogłyby być pokryte.

Polimorfizm pozwala związać wspólnym wywołaniem metody wiele instancji klas, dzięki czemu system zbudowany w ten sposób jest elastyczniejszy, a klasy w większym stopniu ukrywają szczegóły swojej implementacji. Zamiana klas polimorficznych nie wprowadza żadnych zmian do struktury systemu.

Polymorphism Factor (PF)

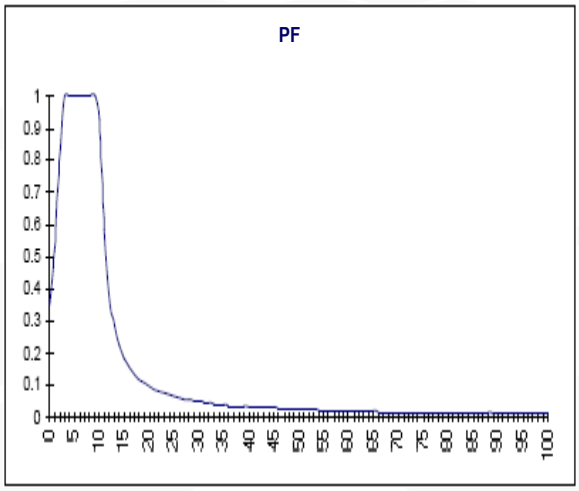


F.B. e Abreu (1995)

Wykres ten przedstawia wartości metryki PF zmierzone w znanych już systemach. Co ciekawe, wartości te są znacznie niższe niż w przypadku metryki MIF, która przecież dotyczy dziedziczenia, a zatem jest blisko związana z mechanizmem polimorfizmu. Ponadto dziedziczenie nie jest jedynym sposobem uzyskania zachowania polimorficznego. Zatem polimorfizm jest stosowany w znacznie mniejszym stopniu niż inne mechanizmy obiektowe.

Wartości tej metryki mieszczą się w granicach od ok. 3% (Centerline) do ok.13% (NIHCL).

Optymalne wartości metryki PF



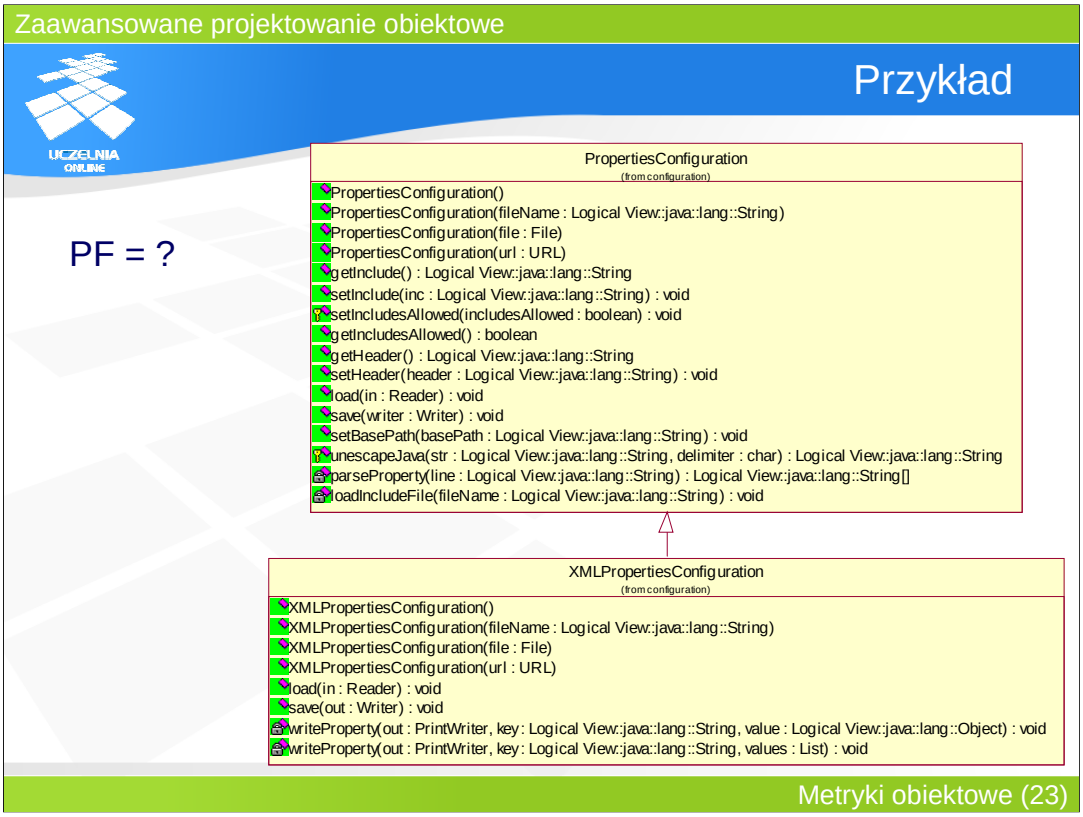
Niski współczynnik użycia polimorfizmu wskazuje na strukturalny (a nie obiektowy) projekt systemu.

Wysoki współczynnik oznacza skomplikowaną hierarchię dziedziczenia.

F.B. e Abreu (1995)

Obserwacje z poprzedniego slajdu są potwierdzone przez wykres optymalnych wartości metryki PF. Mieszczą się one w granicach od kilku do dwudziestu kilku procent.

Wartość niższa niż kilka procent może wskazywać na strukturalny (a zatem nie korzystający z polimorfizmu) projekt systemu, w którym każdorazowo odbiorca wywołania metody jest znany w momencie kompilacji i łączenia. Z drugiej strony, zbyt wysoka wartość współczynnika PF sugeruje zbyt skomplikowaną hierarchię dziedziczenia, w której większość metod jest pokrywana w klasach potomnych. Łączy się to z niską elastycznością systemu oraz słabym powtórnym wykorzystaniu kodu przez podklasy.



Wymieniony wcześniej układ dwóch klas tym razem jest obiektem zainteresowania ze względu na pokrywanie metod. Zadanie polega na obliczeniu wartości tej metryki oraz interpretacji uzyskanego wyniku.

Metryka powiązań pomiędzy obiektami

CF jest metryką określającą stopień powiązań pomiędzy obiektami

$$CF = \frac{\sum_{i=1}^{TC}\left(\sum_{j=1}^{TC} is\_client(C_i, C_j)\right)}{TC^2 - TC - 2 \times \sum_{j=1}^{TC} DC(C_i)}$$

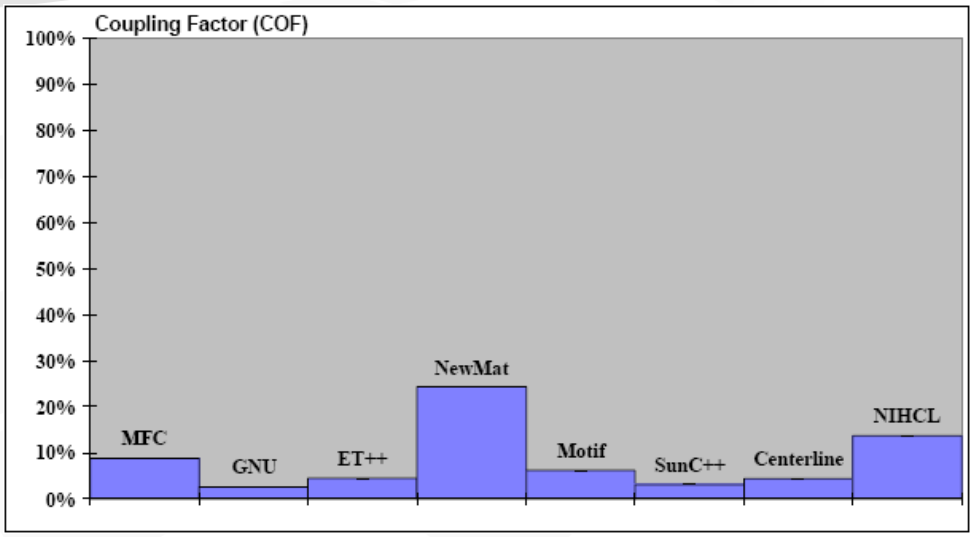
$DC(C_i)$ – liczba klas potomnych klasy C_i

$$is\_client(C_c, C_s) = \begin{cases} 1 \text{ iff } C_c \Rightarrow (C_s \wedge C_c \neq C_s) \wedge \neg(C_c \rightarrow C_s) \\ 0 \text{ w przeciwnym przypadku} \end{cases}$$

Ostatnią, szóstą metryką należącą do zestawu MOOD, jest CF – współczynnik określający stopień powiązań pomiędzy obiektami innych niż poprzez dziedziczenie. Ponownie jest on liczbą niemianowaną i jest zapisywany w postaci ułamka. W jego liczniku znajduje się suma metod po wszystkich klasach, pomiędzy którymi występują powiązania inne niż wynikające z dziedziczenia (przy czym powiązania są jednokierunkowe; powiązania dwukierunkowe są liczone jako dwa powiązania jednokierunkowe). Nie ma jednak rozróżnienia między naturą tego powiązania: asocjacje, kompozycje czy agregacje są traktowane tak samo. Mianownik z kolei określa maksymalną liczbę powiązań nie związanych z dziedziczeniem, jakie mogłyby zaistnieć w systemie. Wartość metryki CF jest zatem miarą stopnia wypełnienia grafu powiązań pomiędzy obiektami.

Z pierwszego wykładu wiadomo, że niski stopień powiązań jest jednym z kryteriów wysokiej jakości projektu, zatem można się spodziewać, że metryka ta przyjmuje niskie wartości (jednak większe od 0)

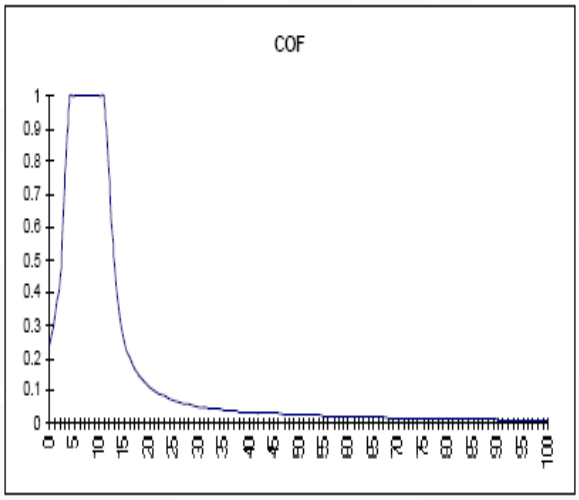
Coupling Factor (CF, COF)



F.B. e Abreu (1995)

Ten sam wniosek płynie z analizy wartości współczynnika CF we wspomnianych wcześniej systemach. Wartości znajdują się w przedziale od ok 3% (GNU) do ponad 20% (NewMat). Warto zauważyć, że systemy te są uznawane za przykłady dobrego projektowania, zatem prawdopodobnie ich wartości metryki CF znajdują się na prawidłowym poziomie.

Optymalne wartości metryki CF



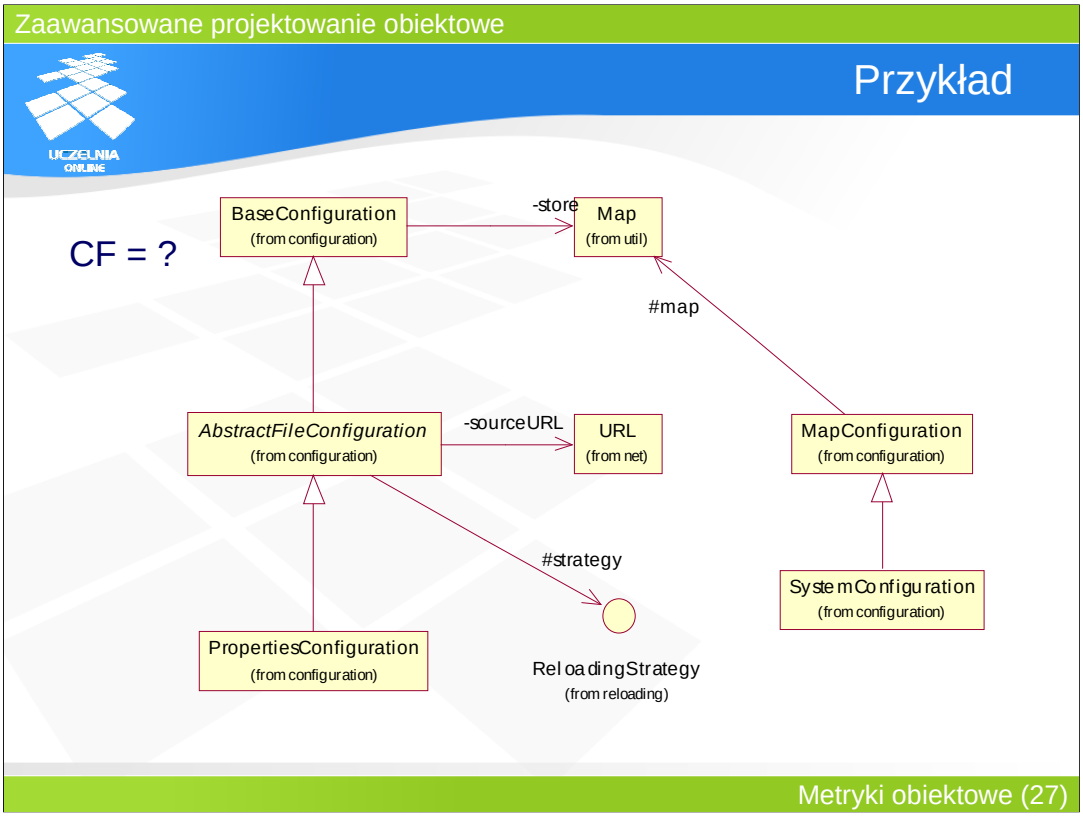
Niski współczynnik powiązań wskazuje na niską funkcjonalność, skupioną w kilku klasach.

Wysoki współczynnik oznacza nieobiektową strukturę projektu.

F.B. e Abreu (1995)

Na tym wykresie, przedstawiającym optymalne wartości metryki CF, widać, że mieszczą się one w przedziale od ok. 5% do ok 15%, zatem na relatywnie niskim poziomie.

Jednak zbyt niska wartość jest niepożądana: wskazuje na niewłaściwy rozkład odpowiedzialności pomiędzy klasami, skoro jedynie kilka z nich wypełnia podstawowe funkcje w systemie. Wysoka wartość współczynnika charakteryzuje systemy o niskiej elastyczności, trudne w testowaniu, modyfikacji i pielęgnacji, w których zmiana jednego elementu wymaga modyfikacji w wielu innych częściach systemu.



Na przedstawionym uproszczonym diagramie klas przedstawiono wybrane klasy i interfejsy pochodzące z omawianego wcześniej projektu Configuration. Należy na podstawie tego diagramu obliczyć wartość metryki CF i ocenić na tej podstawie jakość przedstawionego systemu.

Wyniki empirycznej oceny metryk MOOD

- Metryka AHF jest ograniczona od dołu, natomiast MHF, MIF, AIF, PF są ograniczone od góry [F.B. e Abreu, 1995]
- Wraz ze wzrostem wartości MHF i MIF spada gęstość błędów i koszt ich usuwania [F.B. e Abreu, 1996]
- Nadmierne stosowanie dziedziczenia powoduje wysokie koszty pielęgnacji [F.B. e Abreu, 1996]
- Optymalna wartość AHF to 100% [F.B. e Abreu, 1996]
- Metryki MOOD dostarczają zgrubnej oceny jakości systemu [Harrison, 1998]

Metryki stają się użyteczne, jeżeli są predyktorami zewnętrznych własności oprogramowania. W przypadku zestawu MOOD istnieje wiele raportów dotyczących walidacji metryk i ich korelacji z jakością oprogramowania, liczbą błędów i pielęgnowalnością.

Autorem wielu obserwacji jest twórca metryk, F. Brito e Abreu, który weryfikował je na wybranych bibliotekach obiektowych, spośród których niektóre z nich zostały już przedstawione na poprzednich slajdach. Wyniki jego badań pokazują m.in. następujące fakty:

-metryka AHF jest ograniczona od dołu (tzn. posiada wartość, poniżej której jakość systemu spada), natomiast metryki MHF, AIF, MIF i PF są ograniczone od góry;

-wysokie wartości metryk MHF i MIF (czyli wysoki poziom hermetyzacji oraz powszechne dziedziczenie metod) powodują spadek gęstości błędów, oraz ograniczenie koszty związanego z ich usuwaniem;

-nadużycie dziedziczenia metod prowadzi jednak do podwyższenia kosztu pielęgnacji;

-optymalną wartością metryki AHF jest 100%.

Natomiast ewaluacja metryk MOOD dla 9 komercyjnych systemów dokonana przez Harrisona, Counsella i Nithiego pokazała, że dostarczają one ogólnej informacji o jakości badanego systemu

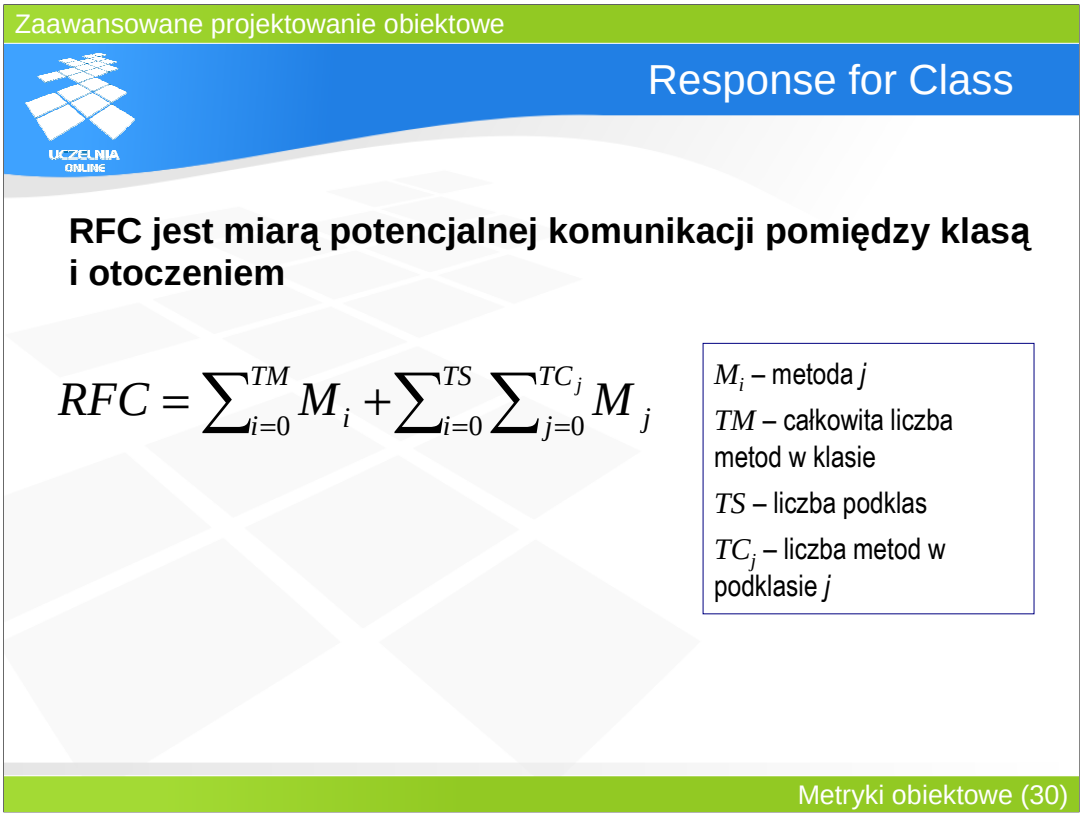
Zestaw metryk CK

- Zdefiniowane przez S.R. Chidambera i C.F. Kemerera w 1991 r.
- Opisują klasy i relacje między nimi
- 6 metryk
  - Response for Class (RFC)
  - Weighted Method per Class (WMC)
  - Depth of Inheritance Tree (DIT)
  - Number of Childer (NOC)
  - Lack of Cohesion of Methods (LCOM)
  - Coupling Between Objects (CBO)

Drugim, choć historycznie pierwszym zestawem metryk obiektowych, był zdefinowany przez Chidambera i Kemerera w 1991 zestaw sześciu metryk obiektowych badających nie – jak w przypadku zestawu MOOD – stopień użycia poszczególnych mechanizmów obiektowych, ale efekt ich zastosowania. Celem stawianym przy ich projektowaniu był pomiar złożoności projektu systemu i jej wpływ na zewnętrzne atrybuty jakościowe produktu – pielęgnowalność, powtórne użycie etc. Nie są jednak dobrymi predyktorami czasu ani pracochłonności, ponadto wymagają znajomości kodu źródłowego programu. Ich wartości mogą być zatem ustalone dopiero w fazie projektowania, a nie planowania, dlatego ich użyteczność z punktu widzenia kierownika projektu jest znacznie mniejsza niż w przypadku MOOD.

Inny jest także obiekt pomiaru: nie jest nim cały system, a pojedyncze klasy (większość metryk z zestawu CK dotyczy właśnie pojedynczych klas) i relacje pomiędzy nimi, co oznacza, że dostarczana przez metryki informacja jest skierowana w większym stopni do projektantów i programistów, a w mniejszym do kierowników projektów.

Wartości każdej z tych metryk obrazują bezwzględną wielkość badanego elementu kodu, co także stanowi różnicę w porównaniu do zestawu MOOD.



Response for Class (RFC) służy do określania potencjalnej komunikacji pomiędzy tą klasą a otaczającym ją środowiskiem. Zgodnie z jej definicją jest to liczba metod, które mogą zostać wywołane w odpowiedzi na komunikat wysłany do obiektu tej klasy, a więc metod zdefiniowanych w analizowanej klasie oraz wszystkich jej podklasach.

Klasy z wysoką wartością metryki RFC są bardziej złożone i oferują większą funkcjonalność. Bardzo wysoka wartość może powodować trudności w testowaniu i weryfikacji poprawności danej klasy.

Weighted Methods for Class

WMC mierzy wewnętrzną złożoność klasy

$$WMC = \sum_{i=1}^{TC} M_i$$

$$WMC = \sum_{i=1}^{TC} [M_i \times CC(M_i)]$$

M_i – dowolna metoda

CC – złożoność cyklomatyczna metody

Metryka Weighted Methods per Class (WMC) służy do określania złożoności klasy jako zbioru metod. Ogólna jej definicja mówi, że jest to suma ważona metod wchodzących w skład klasy. Jednak ponieważ często przyjmowane jest założenie o jednakowej wadze wszystkich metod, wówczas ma ona uproszczoną wersję: WMC jest liczbą wszystkich metod zdefiniowanych w danej klasie.

Innym, często stosowanym współczynnikiem wagi metody, jest jej złożoność cyklomatyczna (CC). Wówczas metryka WMC jest sumą złożoności McCabe'a wyliczonych dla wszystkich metod w klasie.

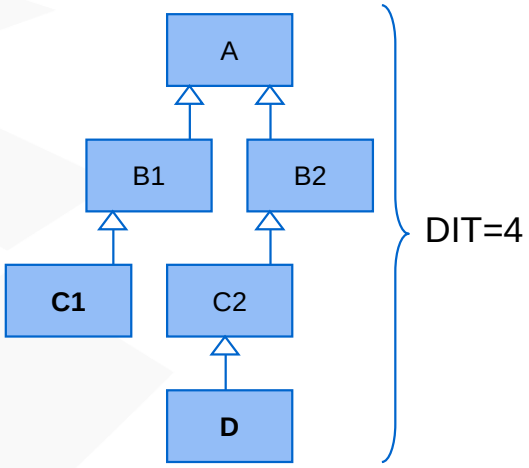
Rolą metryki WMC jest predykcja pracochłonności wymaganej do stworzenia i pielęgnacji klasy. Co ważne, metrykę tę można obliczyć (w przypadku wersji z wagami równymi 1) jedynie na podstawie modelu systemu. Zatem pozwala ona przewidywać ważne atrybuty oprogramowania zanim ono powstanie.

Z przeprowadzonych badań wynika, że wysoka wartość metryki WMC charakteryzuje klasy konkretne, o bardzo wąskiej specjalizacji. Klasy takie rzadko są dalej specjalizowane w postaci kolejnych podklas.

Depth of Inheritance Tree

DIT mierzy złożoność hierarchii dziedziczenia

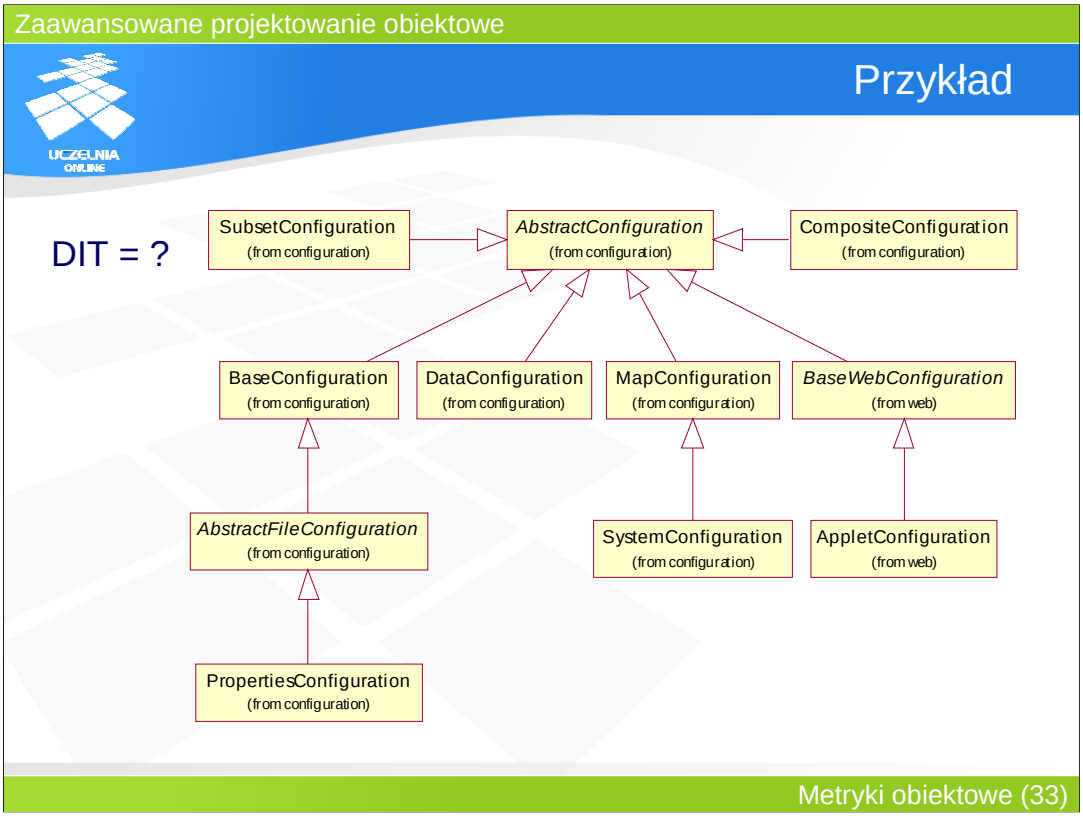
DIT jest maksymalną liczbą poziomów w drzewie dziedziczenia i jest mierzona liczbą "pokoleń" przodków



Metryka DIT służy do pomiaru głębokości hierarchii dziedziczenia. Zgodnie z definicją, DIT opisuje maksymalną wysokość ścieżki dziedziczenia danej klasy, licząc od korzenia drzewa do samej klasy.

Metrykę tę interpretuje się następująco:

- im większą wartość metryki DIT klasa posiada, tym większą liczbę metod ta klasa dziedziczy, zatem tym trudniej przewidzieć jej zachowanie; w efekcie koszt pielęgnacji takiej klasy rośnie;
- drzewa dziedziczenia o dużej głębokości charakteryzują systemy o dużej złożoności, w których uczestniczy wiele spokrewnionych ze sobą klas i metod, a więc zwiększające koszty pielęgnacji; z drugiej strony dziedziczenie zwiększa stopień powtórnego użycia kodu, co ogranicza liczbę duplikatów.

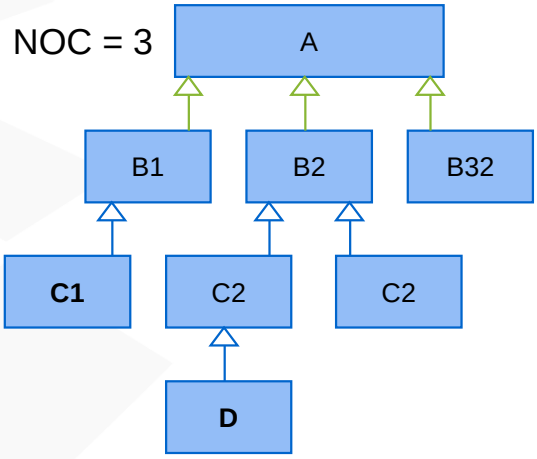


Na slajdzie przedstawiono hierarchię dziedziczenia klas związanych z projektem Configuration. Na podstawie diagramu należy określić wartość metryki DIT dla klas AbstractConfiguration, SystemConfiguration oraz PropertiesConfiguration

Number of Children

NOC mierzy potencjalny wpływ modyfikacji klasy

NOC jest liczbą bezpośrednich podklas lub implementacji klasy

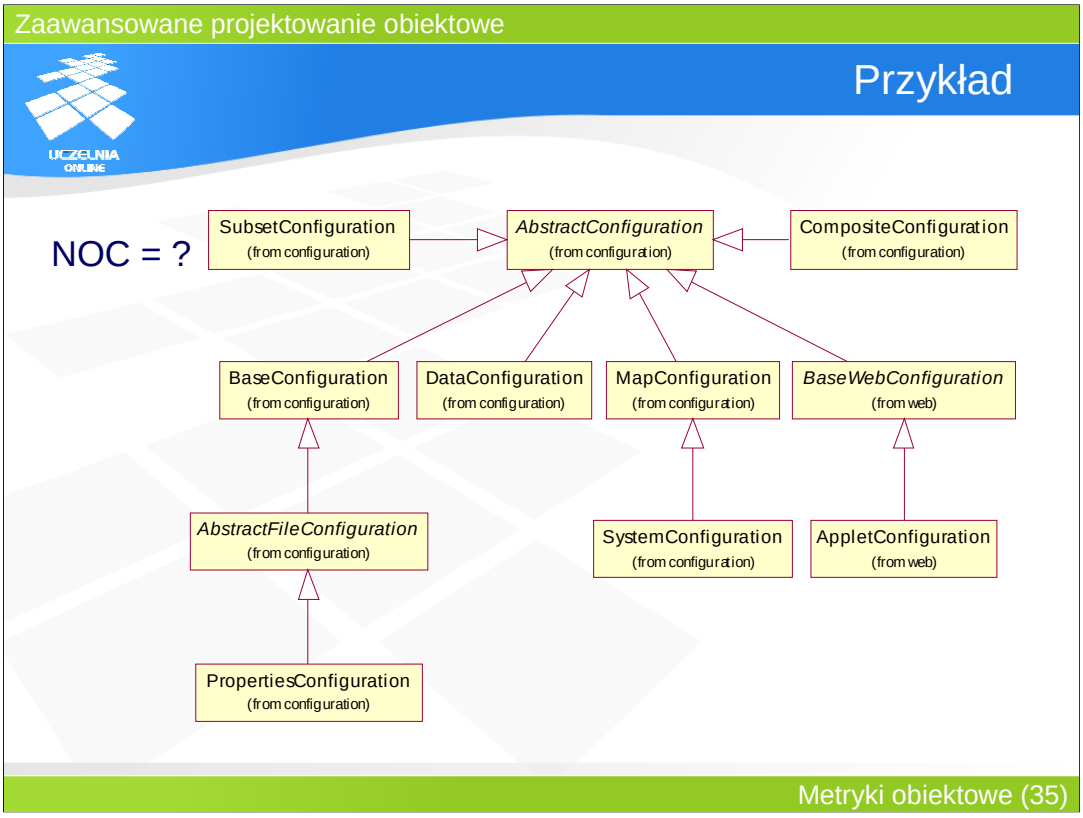


Metryka NOC określa liczbę bezpośrednich potomków danej klasy, a więc podaje, ile razy kod z nadklasy został powtórnie wykorzystany w podklasach.

Metryka posiada następujące interpretacje:

- im większa wartość NOC, tym większe prawdopodobieństwo niewłaściwej abstrakcji nadklasy, skoro wymaga ona wielokrotnego pokrycia; może to być efektem niewłaściwego wykorzystania dziedziczenia;
- im większa wartość NOC, tym więcej wysiłku wymaga testowanie klasy;
- z drugiej strony, wysoka wartość NOC wskazuje na wyższy stopień ponownego użycia poprzez podklasy.

W sumie wartość metryki NOC zwykle przyjmuje wartości 2-5, w którym to przedziale powtórne użycie kodu nadklasy równoważy wady dużej liczby podklas. Wyższe wartości powoduje istotne problemy z pielęgnacją klasy.



Oto znany już przykład klas. Tym razem zadanie polega na obliczeniu metryki NOC dla klasy AbstractConfiguration, oraz zinterpretowaniu wyniku.

Lack of Cohesion of Methods (1)

LCOM1 mierzy brak podobieństwa między metodami i atrybutami (niespójność klas)

$$LCOM1 = (P > Q)\,?\,(P - Q) : 0$$

1. dla każdej pary metod wyznacz zbiory atrybutów, do których się odwołują
2. jeżeli zbiory są rozłączne, zwiększ P o 1;
3. jeżeli choć jeden atrybut jest wspólny, zwiększ Q o 1

Kolejną metryką jest LCOM, opisującą – w odróżnieniu od pozostałych metryk CK – brak pewnej cechy, a nie jej obecność. Cechą to jest spójność klasy, mierzona jako stopień podobieństwa celów realizowanych przez metody klasy. Klasa jest spójna, jeżeli wszystkie jej metody współpracują, tzn. odwołują się do siebie nawzajem oraz wspólnie odwołują się do atrybutów tej klasy.

W wersji pierwszej tej metryki przyjmuje ona wartość będącą różnicą liczby par metod odwołujących się do różnych atrybutów klasy oraz liczby par metod odwołujących się do przynajmniej jednego wspólnego atrybutu. Wersja ta jednak była szeroko krytykowana. Wśród najważniejszych zarzutów wymieniano m.in. następujące obserwacje:

- *LCOM1* przyjmuje z definicji wartość 0 wskutek różnych okoliczności, zatem wartość ta może być mylnie interpretowana;
- definicja jest oparta na interakcji między metodami a danymi, co może nie być właściwym kryterium spójności w świecie obiektowym;
- *LCOM1* opiera się na dostępie do zmiennych, podczas gdy w wielu językach programowania (np. w C#) mechanizmem dostępu do pól klasy są właściwości (ang. *properties*).

Lack of Cohesion of Methods (3)

LCOM3 mierzy brak podobieństwa między metodami i atrybutami (niespójność klas)

$$LCOM3 = \frac{TM - \frac{\sum_{i=1}^{TA} MA_i}{TA}}{TM - 1}$$

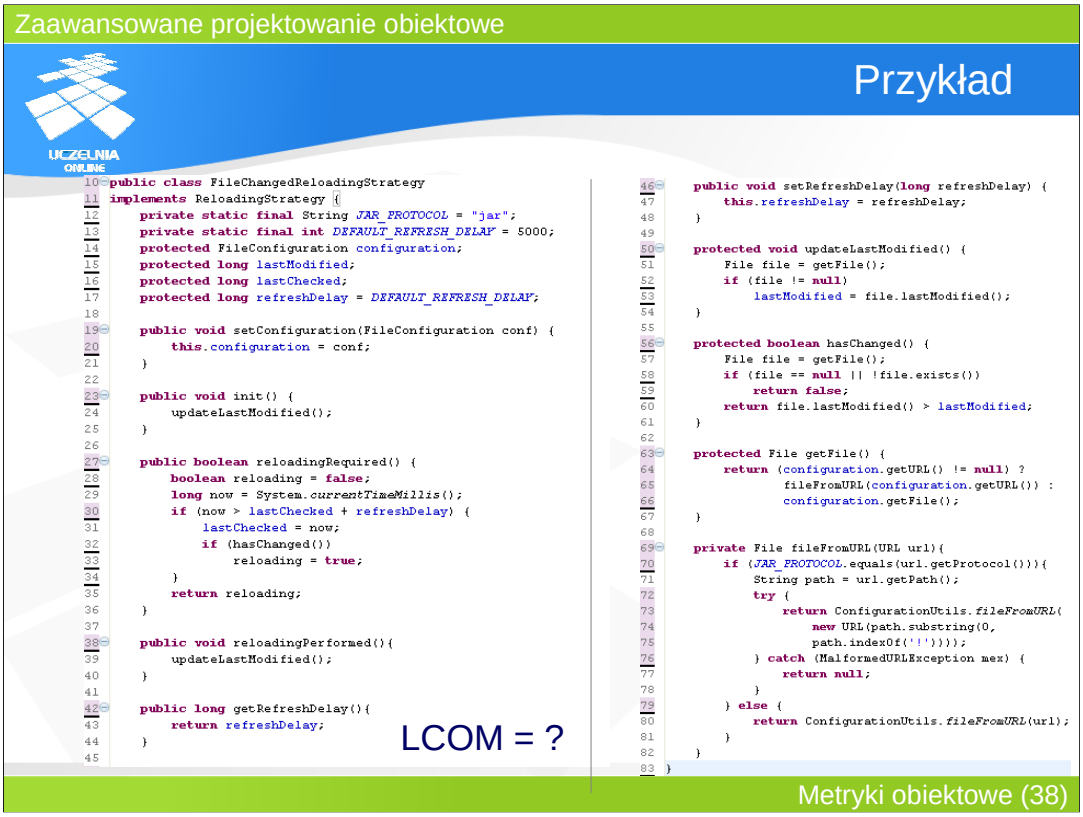
TM – liczba metod w klasie

TA – liczba atrybutów w klasie

MA_i – liczba metod odwołujących się do atrybutu A_i

Dlatego powstały kolejne definicje braku spójności klasy. Jedna z wersji, nazwana LCOM3, autorstwa Hendersona-Sellersa, definiuje metrykę jako względną liczbę metod, które nie odwołują się do poszczególnych atrybutów klasy. W przypadku gdy wartości TA = 0 lub TM = 1, metryka jest niezdefiniowana (w praktyce jej wartość jest podawana jako 0). Definicja ta jest pozbawiona przede wszystkim tych wad, które są związane z interpretacją wartości metryki:

- wartości LCOM3 należą do przedziału od 0 do 2;
- wartości większe od 1 wskazują na brak spójności i prawdopodobną konieczność podziału klasy w przyszłości.

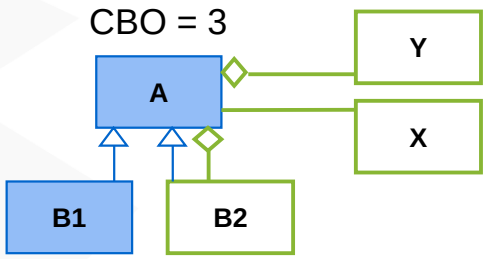


Zadanie związane z tą metryką polega na obliczeniu jej wartości dla przedstawionej klasy.

Coupling Between Objects

CBO mierzy stopień zależności klasy od innych klas nie będących jej przodkami

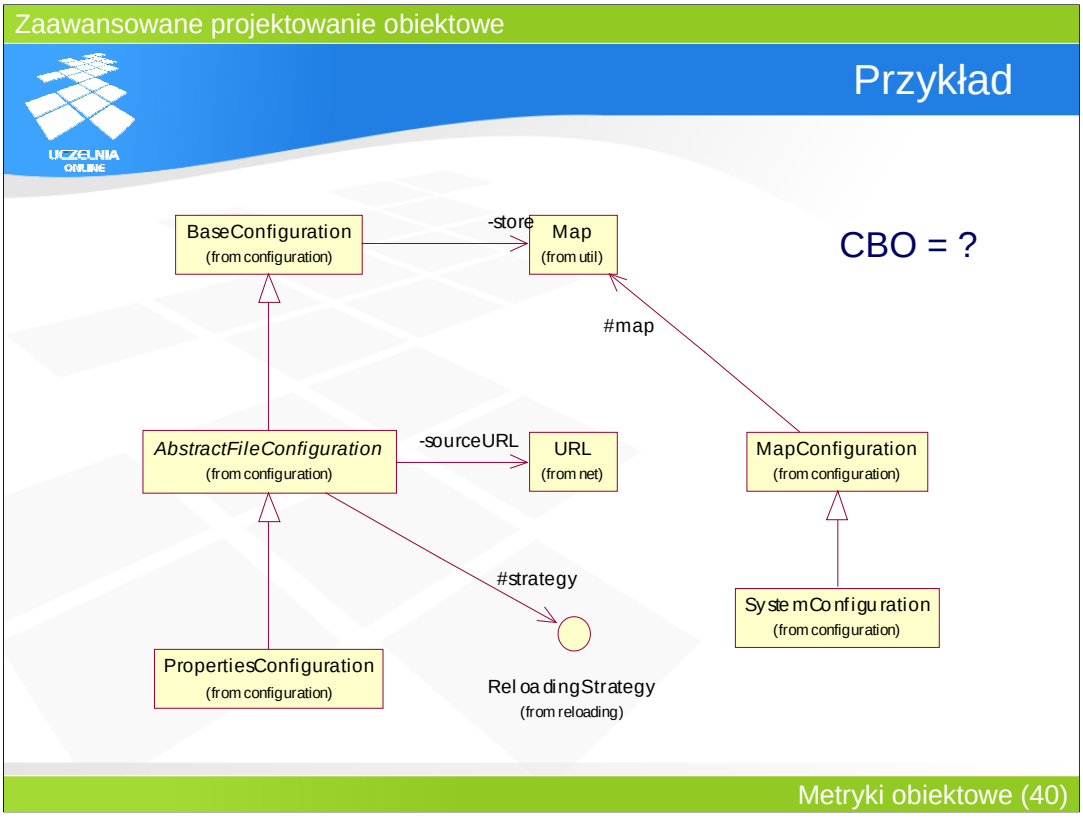
CBO jest liczbą klas związanych z rozważaną klasą relacją inną niż dziedziczenie



Drugim kryterium jakości projektu obiektowego wymienionym podczas pierwszego wykładu jest stopień powiązań pomiędzy obiektami. Metryką należącą do zestawu CK, która mierzy tę wartość, jest CBO. Zgodnie z jej definicją, mierzy ona stopień zależności podanej klasy od innych klas, które nie są związane z nią poprzez dziedziczenie.

Wartość metryki CBO to liczba typów użytych w atrybutach, parametrach, klauzulach throws metod oraz typach zwracanych przez metody – zatem jest to długość słownika typów obiektowych, do których istnieją odwołania z danej klasy. Wyjątkiem są typy prymitywne (int, double, boolean etc.) oraz podstawowe typy systemowe (np. w przypadku Javy są to klasy z pakietu *java.lang.**), które nie są zaliczane jako odwołania.

Metryka ta powinna przyjmować małe wartości, ponieważ wskazują one, że obiekty są od siebie zależne jedynie w niewielkim stopniu, co z kolei oznacza, że mogą być łatwo modyfikowane lub nawet wymieniane na inne.



Tym razem zadanie polega na obliczeniu wartości metryki CBO dla klasy AbstractFileConfiguration, przedstawionej na diagramie kilku klas wybranych z projektu Configuration.

Metryki R.C. Martina

- Zdefiniowane przez R.C. Martina w 1994 r
- Dotyczą kwestii zależności klas i komponentów pomiędzy sobą
- 5 metryk
  - Efferent Coupling (Ce)
  - Afferent Coupling (Ca)
  - Instability (I)
  - Abstractness (A)
  - Normalized Distance from Main Sequence (D)

Trzecim zestawem metryk omawianych podczas wykładu jest grupa 5 metryk zaproponowanych przez R. Martina w roku 1994. Inaczej niż w przypadku poprzednich dwóch zestawów, nie stanowią one kompletnego i pełnego zestawu do oceny jakości poszczególnych atrybutów projektu obiektowego. Martina interesowały najbardziej problemy zależności pomiędzy komponentami i pakietami, i jego metryki dotyczą właśnie zagadnienia. Ich poziom szczegółowości zatem znajduje się nawet niżej niż w przypadku metryk CK.

Metryki autorstwa Martina to:

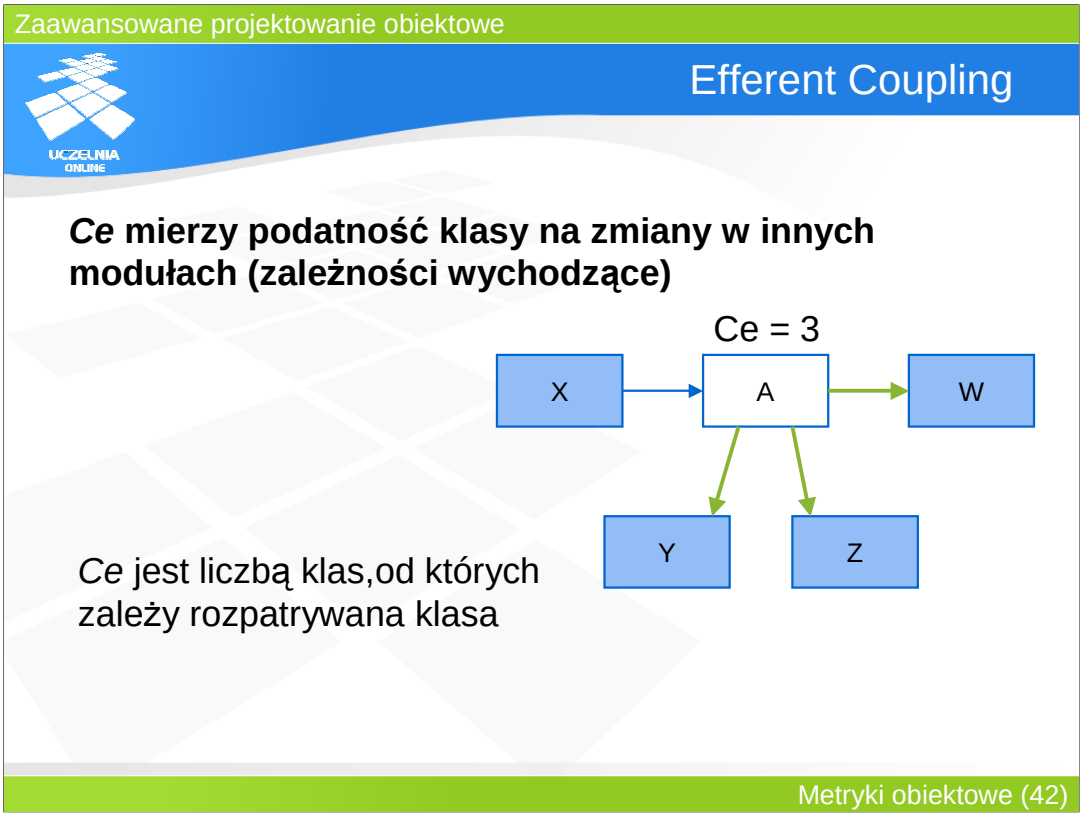
-Efferent Coupling (Ce)

-AfferentCoupling (Ca)

-Instability (I)

-Abstractness (A)

-Normalized Distance from Main Sequence (D)

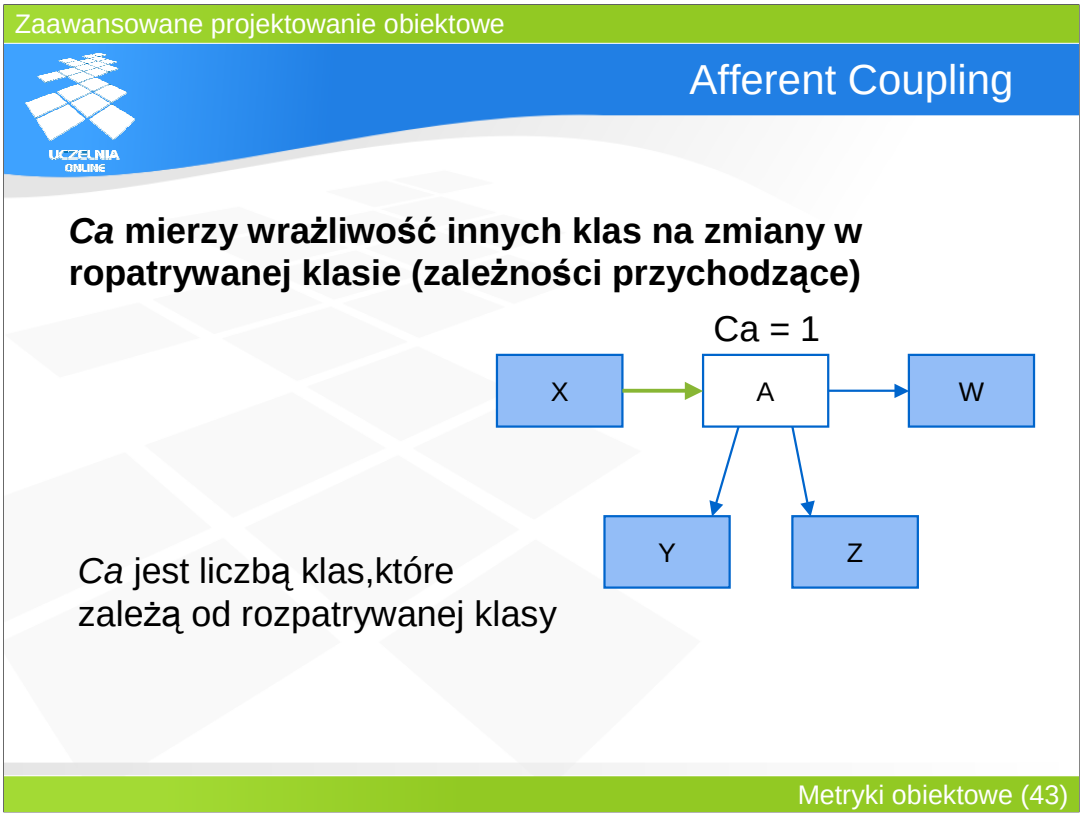


Metryka Ce służy do pomiaru jednego z rodzajów zależności pomiędzy klasami: zależności wychodzących. Mierzy ona podatność rozważanej klasy na zmiany zachodzące w innych modułach, zatem odzwierciedla liczbę źródeł zmian, z którymi klasa musi się liczyć. Na rysunku przedstawiono pięć klas i łączące je relacje wraz z kierunkami zależności. Kolorem zielonym oznaczono zależności wychodzące.

Wysokie wartości metryki wskazują na niestabilność klasy, tzn. że jej stabilność nie zależy od niej samej, ale od zmian w innych miejscach kodu. Jednak komponent z wysoką wartością metryki Ce nie jest odpowiedzialny przed innymi komponentami, co oznacza, że zmiany przeprowadzone w nim nie są propagowane.

Preferowane wartości wynoszą od 0 (klasa jest całkowicie niezależna od innych) do 20. Wartości wyższe powodują problemy z pielęgnacją i rozwojem kodu.

Typowym przykładem komponentów o wysokiej wartości Ce są elementy GUI, ponieważ niemal każda zmiana w logice systemu musi być odzwierciedlona w interfejsie użytkownika.

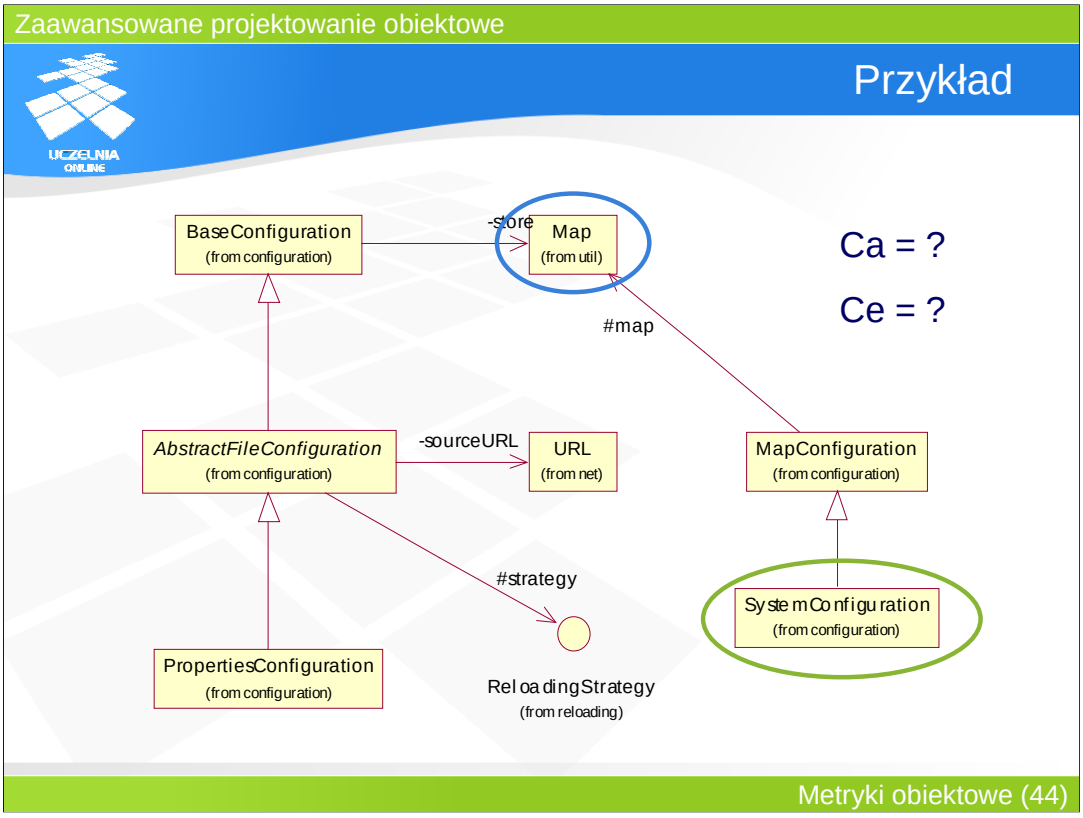


Metryka Ca jest przeciwieństwem metryki Ce. Mierzy ona wrażliwość innych klas na zmiany w analizowanej klasie – a zatem dotyczy zależności przychodzących. Na rysunku tylko jedna zależność ma charakter przychodzący.

Wysoka wartość Ca zwykle (choć niejawnie) sugeruje stabilność komponentu. Wynika to z faktu, że klasa, od której zależy wiele innych klas nie może być modyfikowana często w znaczący sposób, ponieważ zwiększa to prawdopodobieństwo rozprzestrzenienia się zmiany. Z drugiej strony komponenty o wysokiej wartości tej metryki ponoszą znacznie większą odpowiedzialność wobec związanych z nimi innych klas.

Preferowane wartości metryki należą do przedziału od 0 do 500. Górna wartość jest znacznie wyższa niż w przypadku metryki Ce z uwagi na utrudnioną kontrolę nad klasami, które zależą od analizowanej klasy.

Przykładem klas o dużej odpowiedzialności wobec innych klas (czyli o wysokiej wartości metryki Ca) są klasy biznesowe w aplikacji oraz różnego rodzaju kontrolery w aplikacjach zgodnych ze wzorcem MVC.

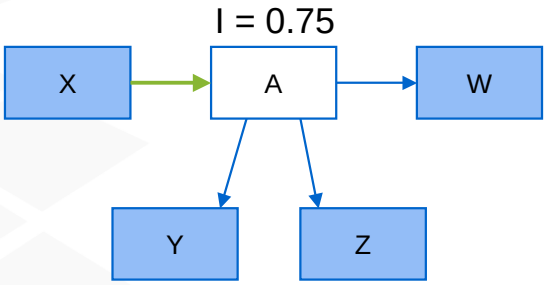


Na rysunku, ponownie zaczerpniętym z projektu Configuration, zaznaczono dwie klasy: Map oraz SystemConfiguration. Pierwsza charakteryzuje się pewną liczbą zależności przychodzących, natomiast druga – wychodzących. Zadanie polega na wyznaczeniu wartości metryk Ca i Ce dla tego układu klas.

Instability

***I* mierzy niestabilność klasy (względną podatność na zmiany)**

$$I = \frac{Ce}{Ce + Ca}$$



I jest relacją powiązań wychodzących (*Ce*) do wszystkich powiązań klasy

Rozróżnienie pomiędzy zależnościami przychodzącymi (*afferent*) i wychodzącymi (*efferent*) pozwoliło Martinowi na dalsze rozważania dotyczące zależności i zobowiązań klas. Wprowadził on pojęcie niestabilności klasy, czyli względnej podatności na zmiany. Metryka ta jest zdefiniowana jako stosunek zależności wychodzących do wszystkich zależności klasy, i przyjmuje wartości od 0 do 1.

Martin na podstawie tej metryki wskazał dwa rodzaje komponentów:

- posiadające wiele wychodzących i niewiele wchodzących zależności (a więc wartość I w ich przypadku jest bliska 1), które są niestabilne z uwagi na możliwość zmian w tych pakietach;
- posiadające więcej zależności przychodzących od wychodzących (a więc I jest bliska 0), które są trudniejsze do zmodyfikowania, ponieważ ponoszą dużą odpowiedzialność wobec klas od nich zależnych.

W zależności od rodzaju komponentu, optymalne wartości metryki I powinny wynosić od 0 do 0.3 albo od 0.7 do 1.0. Oznacza to, że dobry projekt obejmuje komponenty bardzo stabilne albo bardzo niestabilne, natomiast należy unikać komponentów o pośredniej stabilności.

Abstractness

***A* mierzy stopień abstrakcji klasy**

$$A = \frac{T_{abstrakcyjne}}{T_{abstrakcyjne} + T_{konkretne}}$$

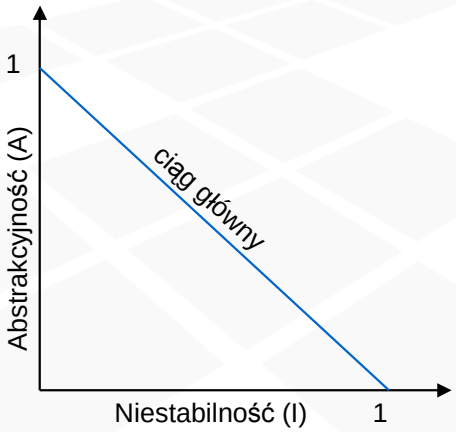
A jest względną liczbą klas abstrakcyjnych wewnątrz pakietu

W pewnym sensie bliźniaczym pojęciem w stosunku do niestabilności jest abstrakcyjność. Metryka A mierzy stopień abstrakcji komponentu, mierzonej jako udział klas abstrakcyjnych do wszystkich klas w komponencie.

Klasy abstrakcyjne są odpowiedzialne wobec innych klas, ponieważ to one stanowią zwykle korzeń dziedziczenia, i zmiany w nich z pewnością wymagałyby modyfikacji w klasach zależnych.

Ciąg główny

***A* mierzy stopień abstrakcji klasy**



- w optymalnym przypadku *I + A = 1*
- *A = 0* i *I = 1* – klasy konkretne, które nie mogą mieć podklas
- *A = 1* i *I = 0* – klasy czysto abstrakcyjne
- inne przypadki: balans pomiędzy *A* i *I*

R.C. Martin (1994)

Powiązanie tych dwóch metryk, I i A, pozwoliło Martinowi sformułować tezę o istnieniu tzw. ciągu głównego. Doszedł on do wniosku, że w optymalnym wypadku niestabilność klasy jest kompensowana jej abstrakcyjnością, a więc zachodzi równanie I + A = 1. Jeżeli na układzie współrzędnych osiami będą wartości I i A, wówczas połączenie punktów o współrzędnych (0, 1) i (1, 0) pozwoli narysować odcinek – ciąg główny, który zawiera punkty spełniające podaną zależność.

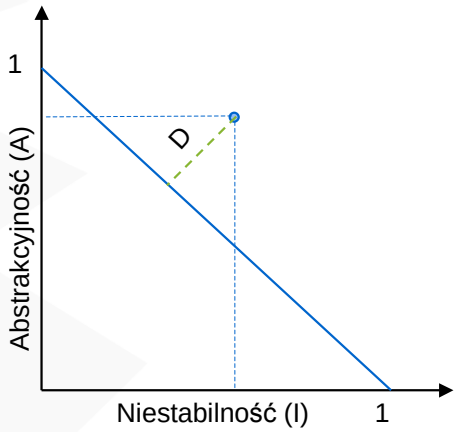
Dobrze zaprojektowane klasy powinny grupować się na tym wykresie wokół punktów skrajnych, tzn. spełniających równanie oraz bardzo niestabilnych lub bardzo odpowiedzialnych.

Znormalizowana odległość od ciągu głównego

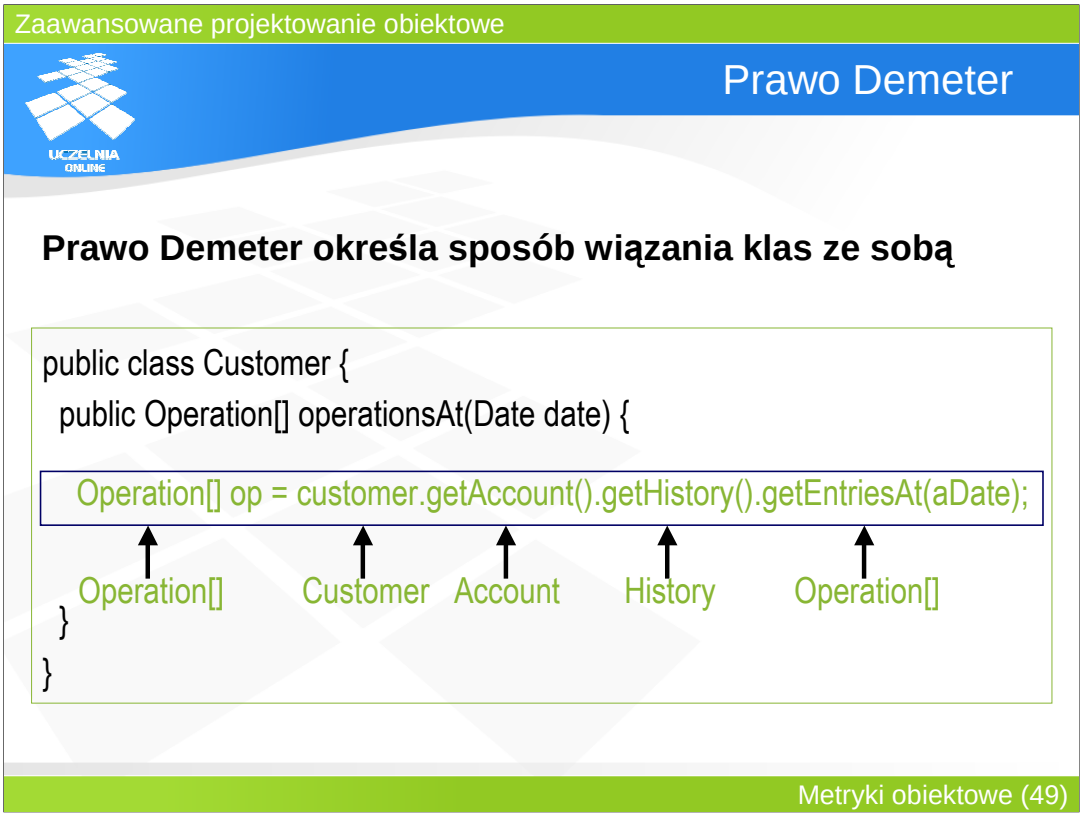
***D* mierzy równowagę pomiędzy stabilnością i abstrakcyjnością**

$$D = (A + I - 1)$$

D jest punktu reprezentującego klasę (I; A) od ciągu głównego



Ponieważ jednak nie wszystkie klasy spełniają podany wcześniej warunek, dlatego Martin zaproponował, aby obliczać odległość klasy od ciągu głównego jako miarę nieprawidłowej faktoryzacji. Duża odległość wskazuje na błędny przydział odpowiedzialności do klasy i konieczność jej restrukturyzacji.



Ostatni punkt wykładu nie dotyczy wprawdzie metryk, jednak także opisuje jedno z kryteriów jakości projektu obiektowego. Kryterium to zostało sformułowane w wyniku projektu Demeter i jest znane właśnie jako prawo Demeter. Dotyczy ono dopuszczalnego sposobu wiązania klas ze sobą, a więc jest powiązane z opisanymi wcześniej metrykami CBO i CF.

Problem dotyczy długich łańcuchów wywołań metod, w których poszczególne ogniwa służą do znalezienia kolejnego obiektu. Na pierwszy rzut oka taki łańcuch jest prawidłowym rozwiązaniem, ponieważ każde jego ogniwo zna tylko swojego następnika i oferuje wszystkim usługę jego zlokalizowania. Jednak istnieje jedna klasa, która jest wówczas powiązana ze wszystkimi innymi: jest nią klasa, której metoda wywołuje łańcuch metod. W przedstawionym przykładzie metoda *operationsAt()* jest związana z klasami Operation, Customer, Account i History, choć one między sobą nie są w żaden sposób powiązane funkcjonalnie.

Prawo Demeter opisuje sposób, w jaki należy ograniczać długie łańcuchy wywołań. Można je streścić zaleceniami "nigdy nie rozmawiaj z obcymi" i "ufaj jedynie swoim bezpośrednim przyjaciołom". W praktyce oznacza to, że obiekt powinien unikać wywoływania metod klasy, której obiekt został mu dostarczony przez inny obiekt

Istnieją dwie postaci prawa Demeter: silna i słaba. Obie określają, że klasa ma prawo wywołać metodę w następujących obiektach

- własnym
- własnych parametrów
- własnych atrybutów i zmiennych tymczasowych
- samodzielnie utworzonych

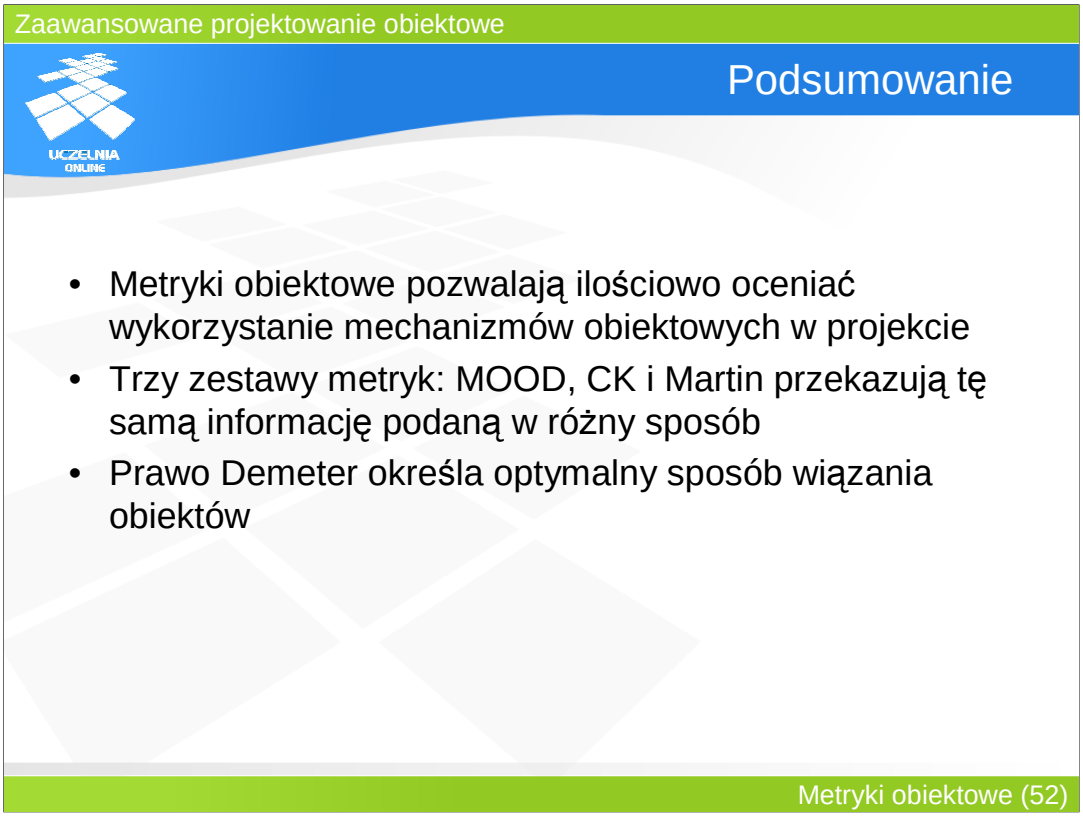
przy czym w przypadku wersji słabej prawo to jest rozszerzone także na podklasy klasy analizowanej.

Wnioski z prawa Demeter

Stosowanie prawa Demeter (LoD):

- ułatwia pielęgnację
- zmniejsza współczynnik błędów
- ogranicza powiązania między obiektami
- wzmacnia hermetyzację i abstrakcję obiektów
- powoduje przekazanie odpowiedzialności za dostęp do składowych z obiektu wywołującego do właściciela składowych

Wnioski płynące ze stosowania prawa Demeter, potwierdzone empirycznie podczas badań na rzeczywistych programach wskazują, że ułatwia ono pielęgnację, zmniejsza gęstość błędów, a także ogranicza liczbę i rodzaj powiązań pomiędzy obiektami, a co za tym idzie – wzmacnia hermetyzację i abstrakcję klasy. Zastosowanie prawa Demeter na poziomie kodu programu powoduje przeniesienie odpowiedzialności za dostęp do metod i atrybutów klasy z obiektu odwołującego się do nich do ich właściciela.



Podczas wykładu omówiono wybrane zestawy metryk obiektowych: MOOD, CK i R. Martina. Metryki obiektowe powstały, aby lepiej przewidywać i odzwierciedlać wykorzystanie mechanizmów obiektowych w systemach, ale także aby obserwować, jak różne decyzje projektowe wpływają na zewnętrzne atrybuty oprogramowania.

Wymienione trzy zestawy metryk, mimo pozornie zbieżnego przedmiotu pomiaru, dostarczają nieco innej informacji, dotyczącej innej warstwy abstrakcji, a więc są adresowane do innych odbiorców.

Prawo Demeter nie jest związane bezpośrednio z żadną z wymienionych metryk: zawiera ono jedynie wskazówki, w jaki sposób należy wiązań ze sobą obiekty, aby nie ograniczyć pielęgnowalności systemu.