

Jakość oprogramowania.

Część 1

Wykładowca
Dr inż. Zofia Kruczkiewicz

Literatura

1. Roger S. Pressman, Praktyczne podejście do oprogramowania, WNT, 2004
2. Stephen H. Kan, Metryki i modele w inżynierii jakości oprogramowania, Mikom, 2006
3. Jacobson, Booch, Rumbaung, The Unified Software Development Process, Addison Wesley, 1999
4. Shalloway A., Trott James R., Projektowanie zorientowane obiektowo. Wzorce projektowe. Gliwice, Helion, 2005
5. Robert C. Martin, Micah Martin, Agile Programowanie zwinne. Zasady, wzorce i praktyki zwinnego wytwarzania oprogramowania w C#, Helion 2008
6. D. Alur, J. Crupi, D. Malks, Core J2EE. Wzorce projektowe
7. <https://docs.oracle.com/javaee/7/JEETT.pdf>

Zagadnienia

- 1. Zasady procesu zwinnego**
- 2. Metryki jakości oprogramowania**

Zagadnienia

1. Zasady procesu zwinnego [5]

1. Zasada pojedynczej odpowiedzialności (Single-Responsibility Principle – SRP)

Żadna klasa nie może być modyfikowana z więcej niż jednego powodu.

1) Klasa obsługująca reguły biznesowe nie powinna zarządzać trwałością

2) Klasa tworząca obiekty nie powinna ich używać

3) Oddzielanie wzajemnie powiązanych odpowiedzialności- np. obiektowa idea katalogu książek:

–Zmiana sposobu przechowywania treści książki: papierowej lub w formie nagrania - tylko modyfikacja kodu klas z rodziny

TitleBook, TitleBookRead

–Zmiana numeru przeznaczonego na wypożyczenie egzemplarza książki – ściśle związany z egzemplarzem książki **Book**

–Obiekt typu **Reservation** powiązany z obiektem typu **Book** określa, czy książka w danym okresie może być zarezerwowana

2. Zasada otwarte-zamknięte (Open/Closed Principle – OCP)

Składniki oprogramowania (klasy, moduły, funkcje itp.) powinny być otwarte na rozbudowę, ale zamknięte dla modyfikacji .

- 1) Stosowanie **dziedziczenia, polimorfizmu, implementacji interfejsów** tylko w takich przypadkach, gdy istnieje możliwość zmian.
- 2) Należy wyeliminować rozpoznawanie klas zarządzanych lub używanych np. instrukcją **switch** przez klasy, które **używają** lub **zarządzają** tymi klasami
- 3) Przykłady: obiektowa idea katalogu książek:
 - Zmiana strategii przechowywania treści książki (z papierowej do formy nagrania) - tylko modyfikacja kodu klas przez polimorfizm i dziedziczenie: tylko klasa **TitleBookRead**
 - Zmiana sposobu określania czasu przeznaczzonego na rezerwację egzemplarza książki – określa obiekt typu **Reservation**, powiązany z obiektem typu **Book**. Klasa ta może być rozbudowana przez dziedziczenie, zachowując polimorfizm metody **isFree**.

3. Zasada podstawiania Liskov

Musi istnieć możliwość zastępowania typów bazowych ich podtypami. Jest to warunek zasady otwarte-zamknięte (OCP).

- 1) Klasa potomna nie może mieć mniejszej funkcjonalności niż jej klasa bazowa. Podstawianie klasy potomnej powinno następować automatycznie, bez potrzeby rozpoznawania typu obiektu np. instrukcją **switch**
- 2) Przykład: obiektowa idea katalogu książek:
 - automatyczne dostosowanie się do formy przechowywania zawartości książki (klasy: **TitleBook, TitleBookRead**)
 - automatyczne dostosowanie się do typu obiektu z rodziny **TitleBook** tytułu książki zwracanego przez metodę **createTitleBook** obiektu typu **Factory**.

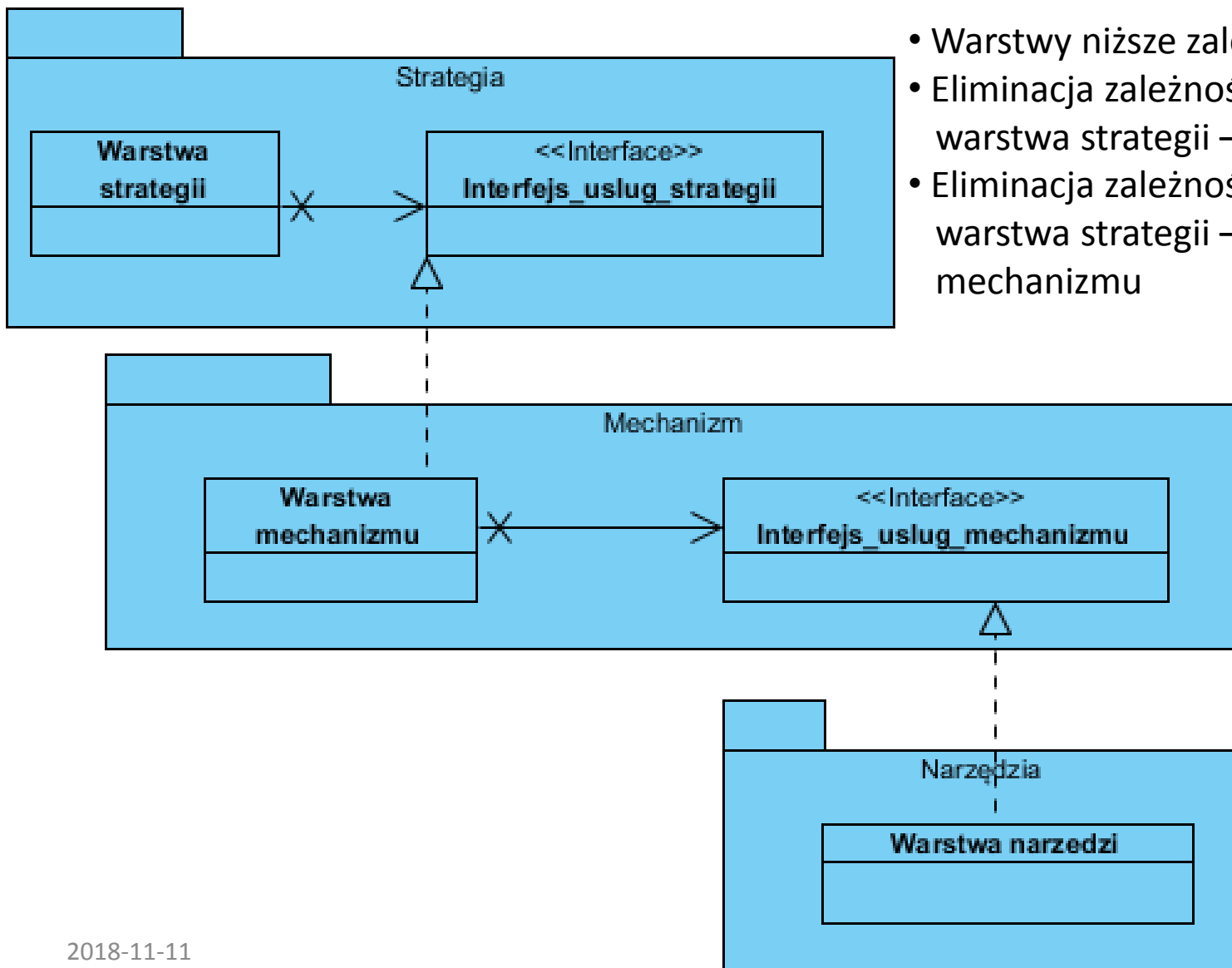
4. Zasada odwracania zależności (Dependency Inversion Principle –DIP)

A. Moduły wysokopoziomowe nie powinny zależeć od modułów niskopoziomowych. Obie grupy modułów powinny zależeć od abstrakcji

B. Abstrakcje nie powinny zależeć od szczegółowych rozwiązań. To szczegółowe rozwiązania powinny zależeć od abstrakcji.

1. Strategia programu nie powinna zależeć od szczegółowych rozwiązań w zakresie implementacji.
2. **Interfejsy są związane ze swoimi właścicielami**, a nie implementującymi je klasami
3. Warstwa strategii może być wielokrotnie używana w dowolnym kontekście pod warunkiem, że moduły niższego poziomu **implementują interfejs usług strategii**

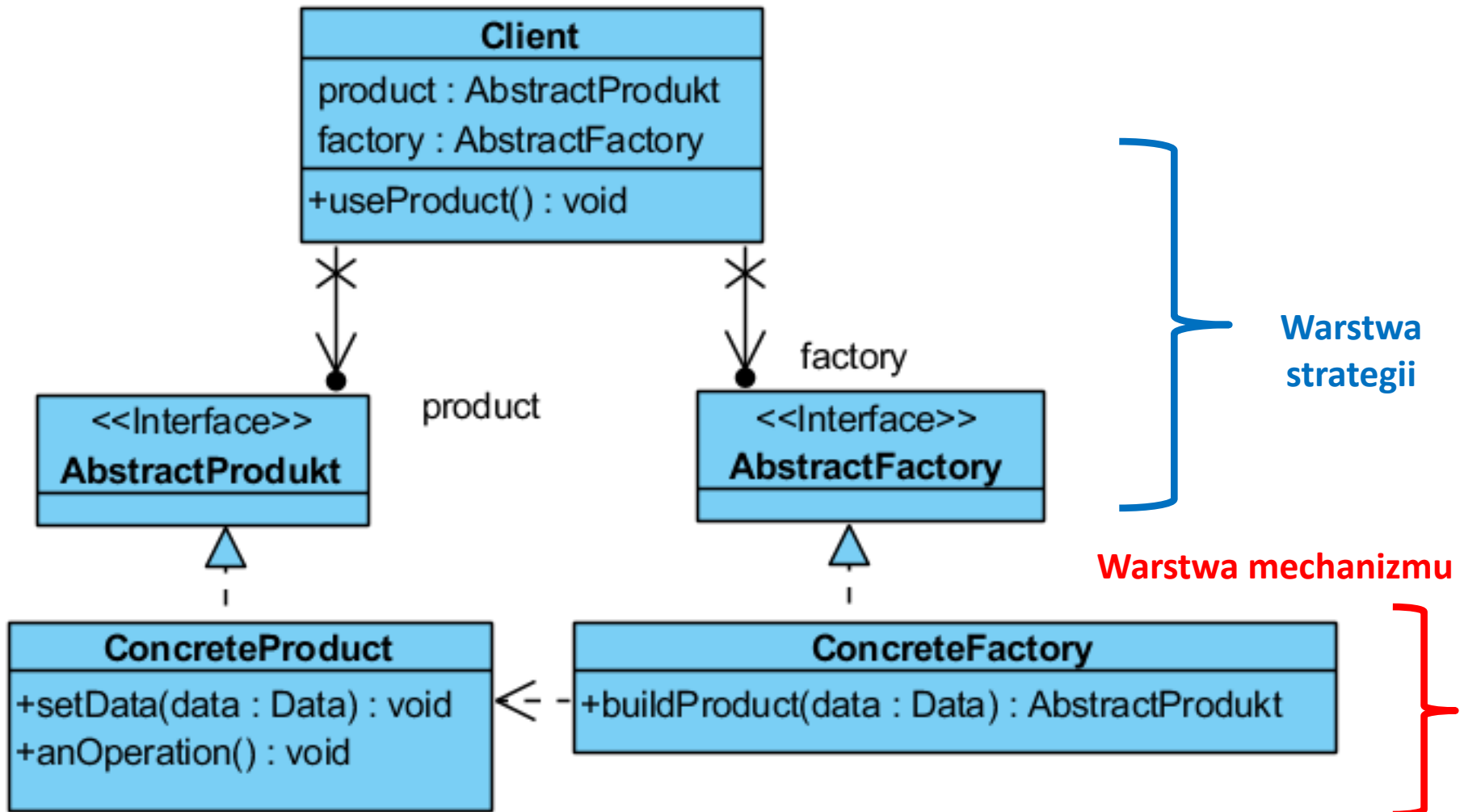
4 A - Podział oprogramowania na poziomy abstrakcji



- Warstwy niższe zależą od wyższych
- Eliminacja zależności przechodniej: warstwa strategii – warstwa narzędzi
- Eliminacja zależności bezpośredniej: warstwa strategii – warstwa mechanizmu

4 B- Zasady procesu zwinnego - przykład

Fabryka abstrakcyjna – *Abstract Factory*

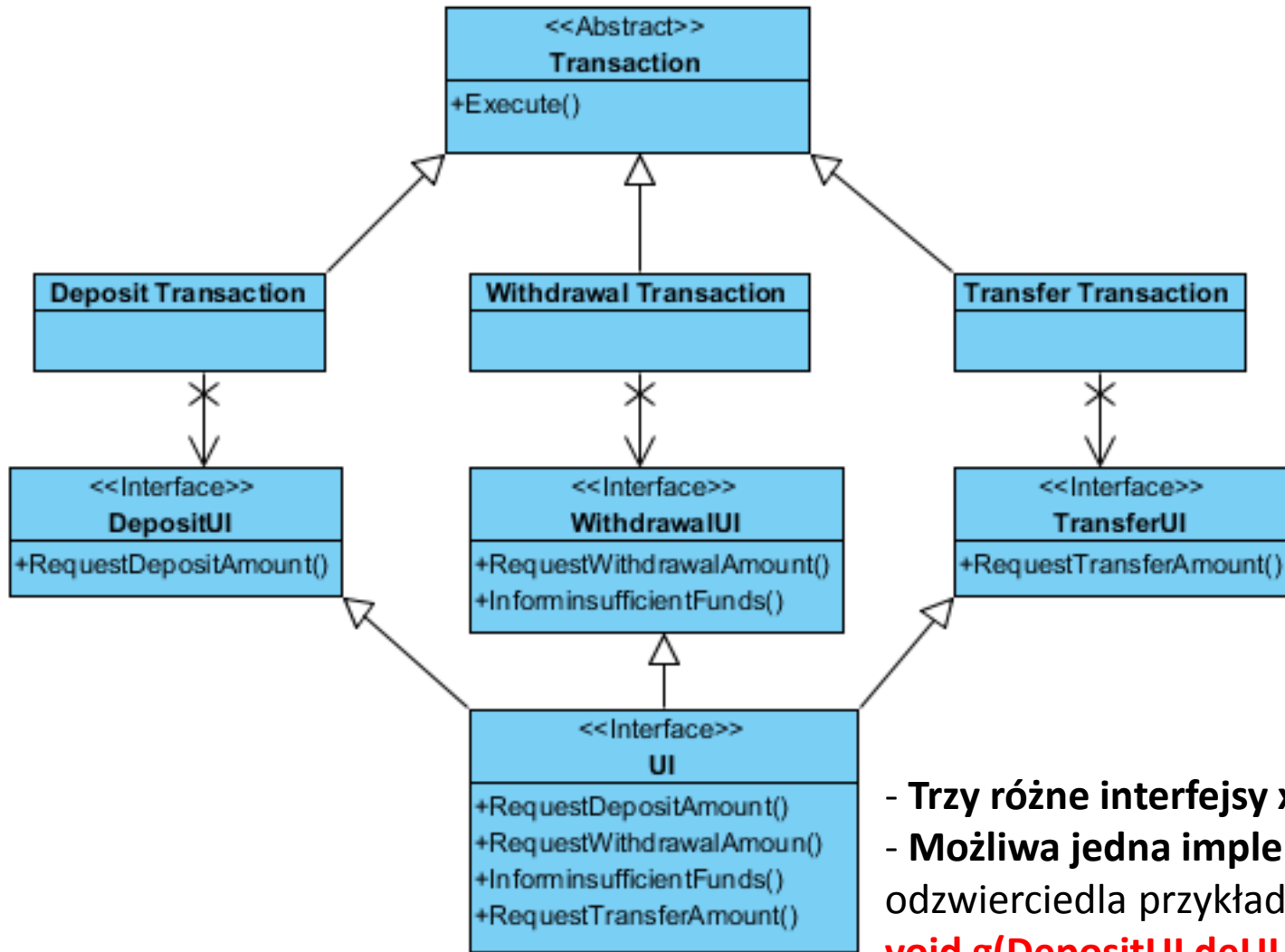


5. Zasada segregacji interfejsów (Interface Segregation Principle – ISP)

Klasa implementująca nie powinna być zmuszana do zależności od metod, których nie używa.

- 1) Separacja przez implementowanie wielu interfejsów
- 2) Dziedziczenie wielobazowe

Przykład ISP



- Trzy różne interfejsy xxxUI
- Możliwa jedna implementacja UI, co odzwierciedla przykład:
void g(DepositUI deUI, TransferUI tranUI)

Możliwe wywołania:

g(ui, ui);

g(ui1, ui2);

gdzie ui, ui1, ui2 są obiektami klasy implementującej interfejs UI. Obiekt **depUI** udostępnia jedynie metodę **RequestDepositAmount**, a **tranUI** metodę **RequestTransferAmount**

Podsumowanie zasad procesu zwinnego

- Kluczowy mechanizm niskiego poziomu
- Podwyższa odporność kodu na zmiany
- Prowadzi do tworzenia kodu wielokrotnego użycia

Struktura wykładu

1. Zasady procesu zwinnego
2. **Metryki jakości oprogramowania**

Literatura

1. D. Harel, Rzecz o istocie informatyki. Algorytmika, WNT 1992
2. I. Sommerville, Inżynieria oprogramowania, s. Klasyka informatyki, WNT 2003
3. Roger S. Pressman, Praktyczne podejście do oprogramowania, s. Inżynieria oprogramowania, WNT, 2004
4. J. Górski i inni, Inżynieria oprogramowania w projekcie informatycznym, MIKOM 1999
5. Stephen H. Kan, Metryki i modele w inżynierii jakości oprogramowania, MIKOM, 2006
6. K. Frączkowski, Zarządzanie projektem informatycznym. Projekty w środowisku wirtualnym. Czynniki sukcesu i niepowodzeń projektów, Oficyna Wydawnicza Politechniki Wrocławskiej
7. J.M.Brady, Informatyka teoretyczna w ujęciu programistycznym, WNT 1983
8. Mourad Badri, Fadel Toure, Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes, Journal of Software Engineering and Applications, 2012, 5, 513-526
9. M. Bruntink, A. van Deursen, An Empirical Study into Class Testability, Journal of Systems and Software , 01/2006
10. A. H. Watson, T. J. McCabe, Structured Testing: A Testing Methodology using the Cyclomatic Complexity Metrics, <http://www.mccabe.com/pdf/mccabe-nist235r.pdf>

Pomiar oprogramowania (wg Brian Henderson-Sellers)

2.1. Złożoność struktury programu jest odpowiedzialna za następujące *atrybuty (charakterystyki) zewnętrzne oprogramowania:*

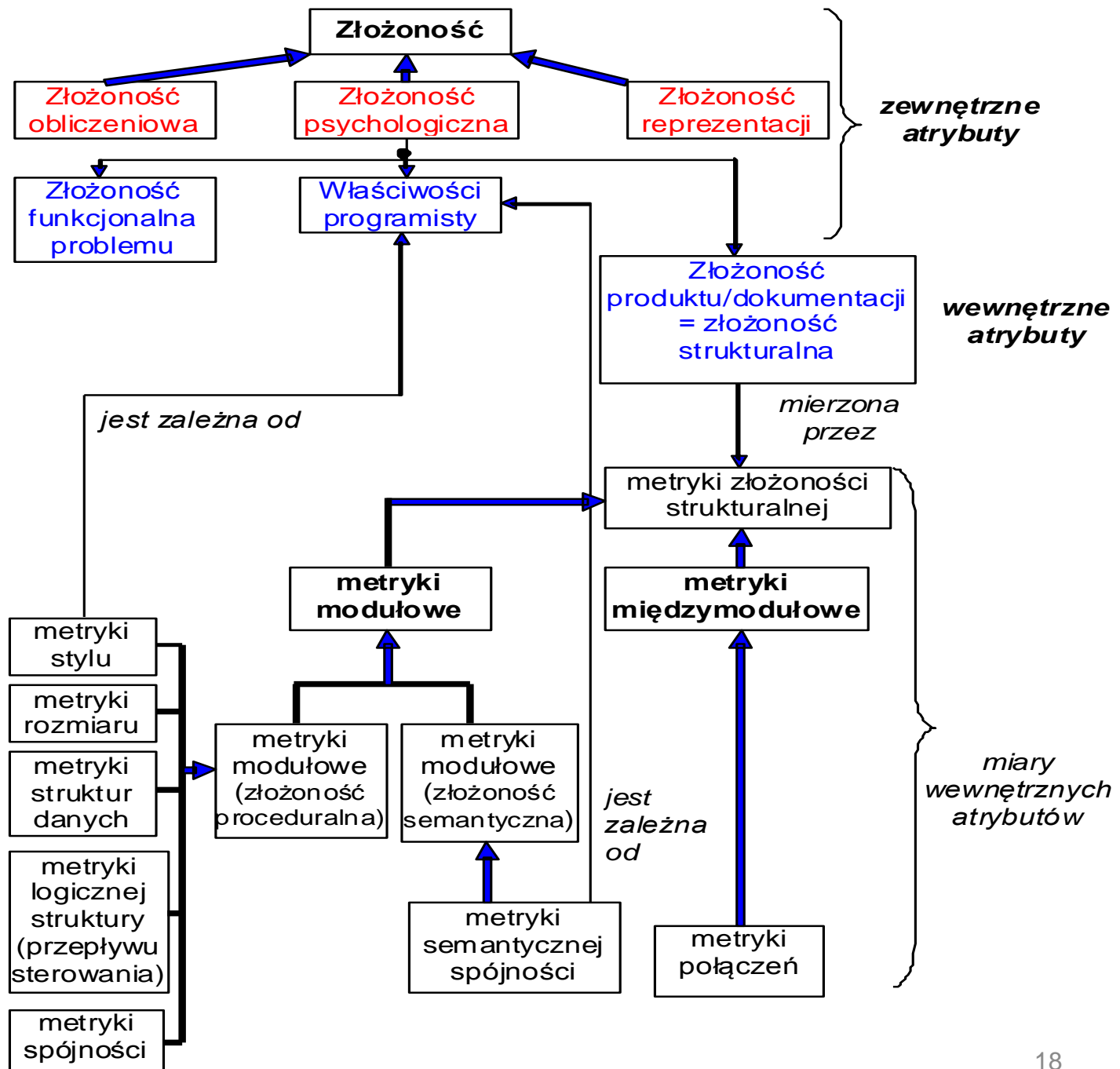
1) jakość oprogramowania:

- testowalności, a więc również niezawodności,
- stopnia osiągniętej abstrakcji
- zrozumiałości programu
- stopnia pielęgnacji
- wieloużywalności

2) funkcjonalność

3) koszt.

**Klasyfikacja
złożoności
oprogramowania -
związki między
zewnętrznymi
atributami
oprogramowania i
miarami
wewnętrznych
atributów (wg
Brian Henderson-
Sellers)**



2.2. Podstawy formalne pomiaru oprogramowania

„**Pomiar** jest to proces, w którym atrybutom elementów świata rzeczywistego przydzielane są liczby lub symbole w taki sposób, aby charakteryzować te atrybuty według określonych zasad.

Jednostki przydzielane atrybutom w ramach pomiaru nazywane są **miarą** danego atrybutu.

Metryka to proponowana (postulowana) miara.”

Formalna definicja elementów środowiska pomiarowego metryk, m. in.:

- system relacji:
 - $A = (A, R_j, o_j)$, gdzie A oznacza niepusty zbiór obiektów rzeczywistych z dziedziny oprogramowania, R_j są empirycznymi relacjami (np. równy, większy), o_j oznacza binarne operacje w zbiorze A ;
 - $B = (B, S_j, o_j)$, gdzie B jest niepustym zbiorem formalnych obiektów (np. liczby, wektory) w dziedzinie pomiarów, S_j są relacjami (np. równy, większy) w zbiorze B , o_j oznacza binarne operacje w zbiorze B ;
- odwzorowanie homomorficzne: $\mu: A \rightarrow B$, stąd mamy $\forall a \in A \exists b \in B (\mu(a) = b)$;
- typy skal jako (A, B, μ) : nominalna, porządkowa, przedziałowa, absolutna, względna.

Typ skali ogranicza obszar odwzorowań, czyli ogranicza pomiar.

Przykłady skal:

- a) skala nominalna: przypisanie obiektom pewnych ustalonych etykiet
np. numery autobusów
- b) skala porządkowa: przypisanie obiektom etykiet wg ustalonego porządku
np. false, true
- c) skala przedziałowa: znaczenie ma odległość między obiektami:
 $g(x)=a*x+b$ ($a>0$)
np. temperatura w stopniach Celsjusza
- d) skala względna: znaczenie ma odległość między obiektami oraz
jedyne punkt odniesienia: $g(x)=a*x$ ($a>0$)
np. temperatura w stopniach Kelvina, pomiar długości, pomiar czasu
- e) skala absolutna: każdy obiekt ma jedną dopuszczalną wartość $g(x) = x$
np. liczba liter w zdaniu, miary statystyczne (wartość przeciętna, mediana, wariancja itp.)

Aksjomaty oceny złożoności oprogramowania:

1. Przy porównywaniu dwóch programów pomija się fragmenty o tej samej złożoności.
2. Istnieje skończona liczba programów o tej samej złożoności.
3. Istnieją różne programy P i Q, takie że mają identyczną złożoność
4. Istnieją równoważne $P \equiv Q$ tzn. spełniające te same funkcje, lecz o różnej złożoności.
5. Złożoność programów P i Q połączonych jest niemniejsza od złożoności każdego z osobna.
6. Dodanie do każdego z pewnych programów P i Q o identycznej złożoności pewnego programu R może spowodować różnice w złożoności uzyskanych programów.
7. Jeżeli program Q powstał przez permutację porządku elementów programu P, to P i Q mają różną złożoność.
8. Jeśli program Q różni się jedynie nazwami od programu P (czyli Q powstał z P po zmianie nazw), to P i Q mają identyczną złożoność.
9. Złożoność programu w wyniku połączenia dowolnych programów P i Q jest większa niż suma złożoności każdego z nich.

2.3. Podstawy metryk złożoności kodu

Poszczególne metryki złożoności strukturalnej mogą być wyznaczane w dowolnym stadium rozwoju oprogramowania, jednak to rzutuje na wybór elementów produktu i rodzaj wyrażenia.

Całkowita złożoność C_p programu jest równa:

$$C_p = C_I + C_M + (C_{Hmax} - C_H)$$

gdzie

- C_I -złożoność międzymodułowa,
- C_M -złożoność modułu,
- C_{Hmax} - całkowita złożoność wynikająca ze spójności modułu zmniejszona o złożoność semantyczną modułu C_H

Złożoność struktury programu jest reprezentowana za pomocą **atrybutów (charakterystyk) wewnętrznych oprogramowania.**

Atrybuty wewnętrzne oprogramowania są wyrażane w postaci obiektywnych miar tych atrybutów czyli tzw. **metryk**.

Metryka - proste wyrażenie, wiążące pewne elementy programu (projektu, kodu źródłowego itp.). Wybór elementów wynika z ich odpowiedzialności za dany atrybut wewnętrzny, a wyrażenie określa wartościowanie atrybutu.

Wyróżnia się następujące **atrybuty wewnętrzne** oprogramowania:

- **atrybuty międzymodułowe** czyli wszelkie związki między modułami (przekazywanie sterowania i parametrów, wspólne korzystanie z pól danych, przy projektowaniu jednego modułu uwzględnia się właściwości innego modułu)
- **atrybuty modułowe** związane z semantycznymi zależnościami między elementami modułu oraz charakterystykami:
 - stylu programowania,
 - rozmiaru oprogramowania,
 - charakteru struktur danych,
 - przepływu sterowania czyli struktury logicznej oprogramowania,
 - oraz spójności oprogramowania jako związku między funkcjami działającymi na danych, a tymi danymi.

2.4. Metryki złożoności międzymodułowej

Oslabienie powiązań między-modułowych prowadzi do zmniejszenia oddziaływań między modułami oraz poprawy struktury oprogramowania.

Elementami łączącymi wyjściowymi z innymi modułami są:

- **Funkcja/metoda wywołująca funkcję z innego modułu**
- wszystkie elementy importowane z innych modułów
- każda informacja z poza modułu potrzebna do zdefiniowania ciała funkcji (np. obsługa błędów), definicji typu strukturalnego, definicji dowolnej zmiennej

Elementami łączącymi wejściowymi dany moduł z innymi modułami są:

- **funkcja/metoda danego modułu wywoływana przez funkcję/metodę z innego modułu**
- Wszystkie elementy modułu przekazywane w importowanych modułach
- informacja zawarta w module potrzebna w innych modułach do dowolnej definicji (np. obsługa błędów), definicji typu strukturalnego, definicji dowolnej zmiennej

RFC - metryki połączeń wyjściowych

$$\text{RFC} = M + R \text{ oraz } \text{RFC}' = M + R'$$

Zakres wartości (1 – 50)

gdzie

M – liczba metod w danej klasie

R – liczba metod wywoływanych przez metody M z innych klas

R' – R + pozostałe metody wywoływane zgodnie z drzewem wywołań

R i R' są wywołanymi metody zwykłymi lub wirtualnymi (tyle razy liczonymi, ile klas przesłania metodę)

Uwagi:

1. Duża wartość metryki oznacza dużo błędów
2. Duża wartość **metryki** oznacza duży wysiłek przy testowaniu
3. Duża wartość metryki oznacza trudność w zrozumieniu klasy

CBO – metryka połączeń wyjściowych z innymi klasami, z którymi jest powiązana dana klasa Zakres wartości (0..14)

Wartość metryki oznacza liczbę klas powiązanych przez wywołanie metod zwykłej lub wirtualnej innych klas (tyle razy liczonej, ile klas przesłania metodę), zastosowanie odwołań do zmiennej (wzajemne powiązanie między klasami jest liczone tylko raz) własnej klasy i przez dziedziczenie, przez argumenty metody, przez typy danych zwracane przez return oraz powiązania za pomocą wyjątków – wartość do 14

Uwagi:

1. Zbyt duża wartość wymaga dużego wysiłku przy testowaniu
2. Ograniczone zastosowanie zbyt powiązanej klasy w innych programach – gorsza wieloużywalność

Fan-out – metryka połączeń wyjściowych

Metryka *Fan-out* wyznacza liczbę połączeń *elementów wyjściowych* jednego modułu z *elementami wejściowymi* innych modułów. Uwzględnia się tylko jedno dowolne połączenie wyjściowe-wejściowe z każdym z modułów.

Fan-in – metryka połączeń wejściowych

Metryka *Fan-in* wyznacza liczbę połączeń *elementów wejściowych* jednego modułu z *elementami wyjściowymi* innych modułów. Uwzględnia się tylko jedno dowolne wejściowo-wyjściowe połączenie z każdym z modułów.

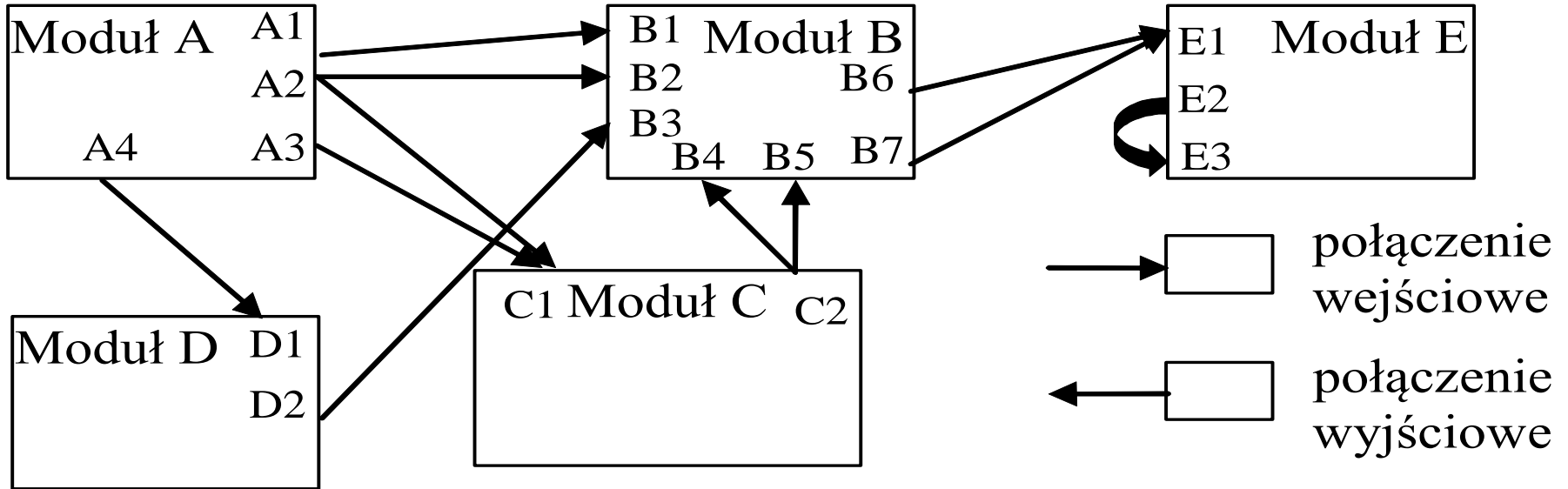
Ca - metryka połączeń wejściowych

Metryka CA wyznacza liczbę klas, które używają danej klasy przez wywołanie jej metod zwykłych lub wirtualnych (tyle razy liczonych, ile klas przesłania metodę), zastosowanie odwołania do zmiennej (wzajemne powiązanie między klasami jest liczone tylko raz) typu danej klasy i dziedziczonych przez nią atrybutów, przez argumenty metod typu danej klasy, wyniki typu danej klasy zwracane przez return oraz wyjątki– definicja powiązań wejściowych jest taka sama jak CBO.

Przykład rozwiązania dla modułu A (rysunek z następnego slajdu)

- Moduł A zawiera elementy łączące wyjściowe: A1, A2, A3, A4. Moduł B dla modułu A zawiera łączące elementy wejściowe B1, B2, moduł C zawiera łączący element wejściowy C1 oraz moduł D zawiera element wejściowy łączący D1 oraz:
- A1 łączy się z B1
- A2 łączy się z B2, C1
- A3 łączy się z C1
- A4 łączy się z D1
- $RS = \{A1, A2, A3, A4\} \cup \{B1, B2\} \cup \{D1\} \cup \{C1\} = \{A1, A2, A3, A4, B1, B2, D1, C1\}$
- $RFC = |RS| = 8$
- $Fan-out = |\{ \langle A1, B1 \rangle, \langle A2, C1 \rangle, \langle A4, D1 \rangle \}| = 3$ //dowolny element wejściowy
- $Fan-in = |\{\}| = 0$
- $R = \{ \langle A1, B1 \rangle, \langle A2, B2 \rangle, \langle A2, C1 \rangle, \langle A3, C1 \rangle, \langle A4, D1 \rangle \}$
- $|R| = 5$

Przykłady metryk międzymodułowych dla modułów A, B, C, D, E cd.



	A	B	C	D	E
Fan-out	3	1	1	1	1
Fan-in	0	3	1	1	2
RFC	8	3	3	2	2
R	5	2	2	1	1

2.5 Metryki złożoności modułowej

Wzmocnienie powiązań wewnątrz-modułowych prowadzi do zmniejszenia oddziaływań między modułami oraz poprawy struktury oprogramowania.

Metryki rozmiaru

SLOC

- Jest to liczba wierszy kodu źródłowego programu liczona niezależnie od liczby instrukcji lub fragmentów instrukcji znajdujących się w każdym wierszu. Nie wlicza się wierszy z komentarzami lub pustych wierszy.
- SLOC jest powszechnie używaną metryką do szacowania nakładów pracy nad programem oraz jest mocno skorelowana z testowalnością, konserwowalnością i zrozumiałością.
- Zakres wartości 5 -1000 linii

S/C

- Metryka ta jest liczbą wszystkich elementów programu należących do bloków logicznych:
 - inicjowanie zmiennych sterujących
 - porównanie
 - zwiększanie zmiennej sterującej
 - liczba instrukcji w każdym bloku
- ```
int i=0
i <10
i++
for (;;) {...}
```

# Żetony

- Jest to zbiór metryk, które określają liczbę:
- $\eta_1$  - liczbę typów operatorów (słownik typów operatorów), czyli liczbę: operatorów predefiniowanych (logicznych, arytmetycznych, przypisania, relacyjnych itp.), słowa kluczowe instrukcji (**while, if, else, do**), nazwy funkcji
- $\eta_2$  - liczbę typów argumentów (słownik typów argumentów), czyli liczbę: wszystkich symboli reprezentujących dane przy deklaracji i definicji
- $\eta_3$  - liczbę wszystkich wystąpień operatorów
- $\eta_4$  - liczbę wszystkich wystąpień argumentów

## NPM - liczba metod publicznych

- Metryka wyznacza liczbę metod publicznych, która pozwala wyznaczyć miarę rozmiaru API pakietu, w którym znajduje się klasa.

## WMC - Liczba metod w klasie

### Zakres wartości (1 - 50)

- **Suma złożoności metod** w klasie (struktura logiczna i rozmiar)

$$WMC = \sum_{i=1}^n c_i$$

- gdzie  $c_i$  jest statyczną złożonością każdej z  $i$  - metod (złożoność cyklomatyczna materiał podany dalej). Jeżeli  $c_i$  jest równe 1, wtedy  $WMC$  jest równe liczbie metod  $n$ .  $WMC$  maleje przy wykorzystaniu polimorfizmu i dziedziczenia

### Uwagi:

- Zbyt duża wartość metryki powoduje w klasie więcej błędów
- Zbyt duża wartość oznacza mniejszą wieloużywalność klasy
- Zbyt duża wartość powoduje mniejsze zrozumienie odpowiedzialności klasy



## DIT - Głębokość dziedziczenia

### Zakres wartości (0 - 5)

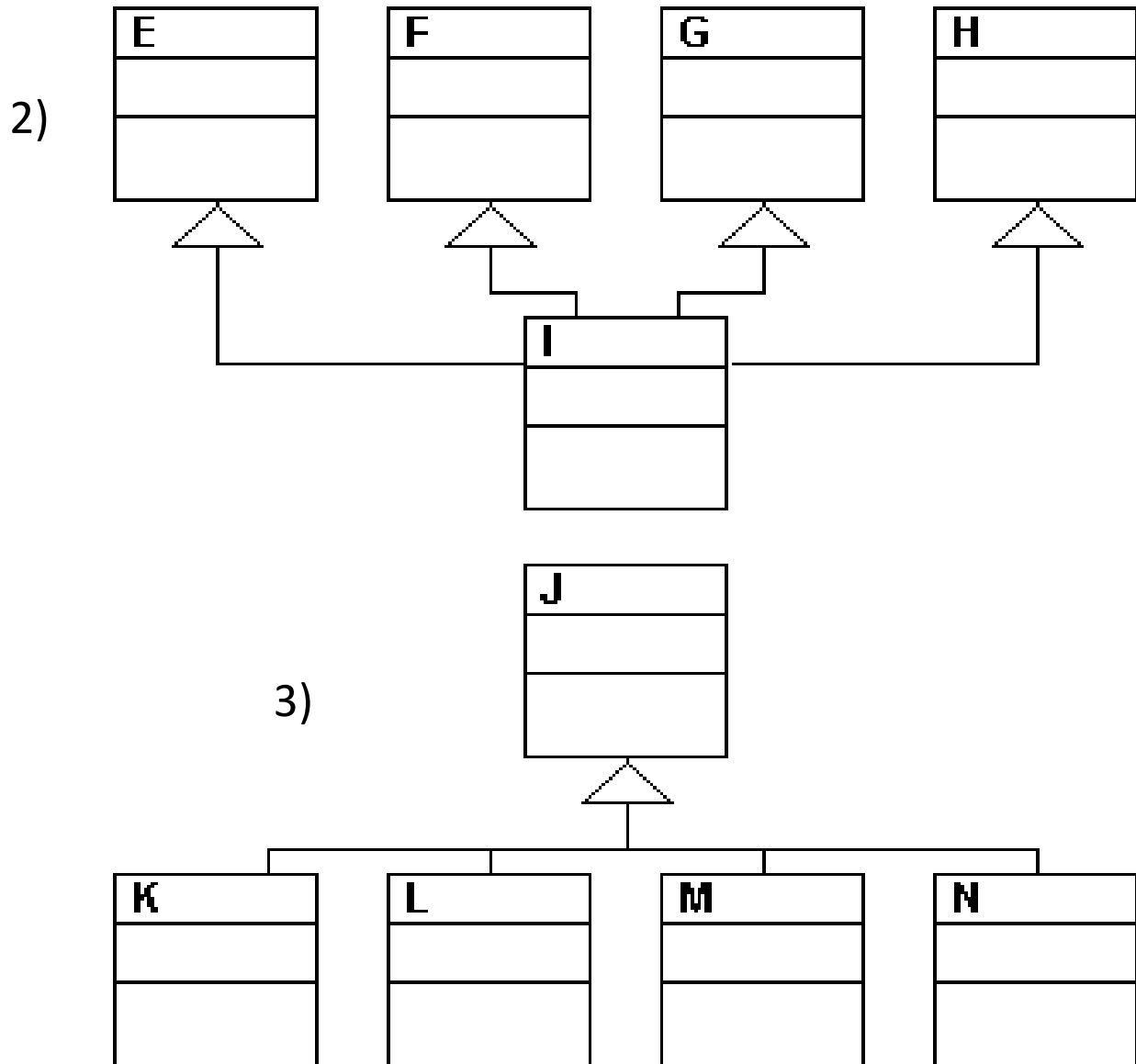
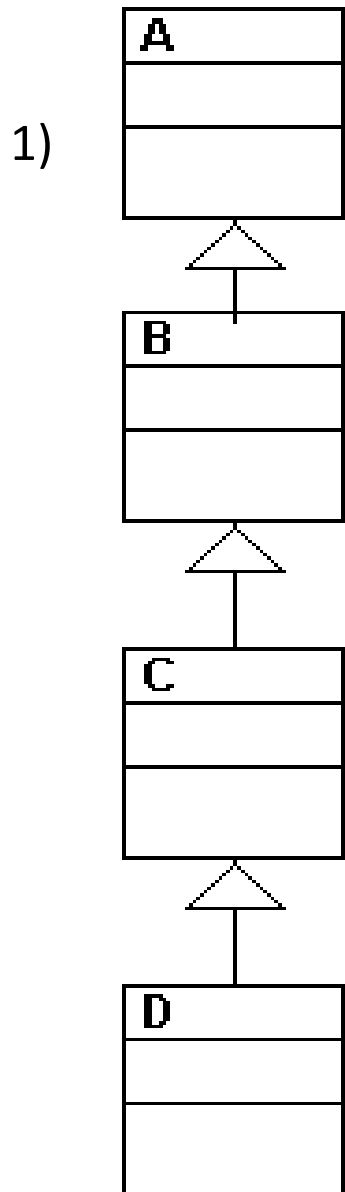
- czyli liczba poziomów w drzewie dziedziczenia odniesiona do liczby klas, określająca zakres dziedziczenia (rozmiar)

$$DIT = \frac{\sum \text{glebokosc dziedziczenia}}{\text{calkowita liczba klas}}$$

#### *Uwagi:*

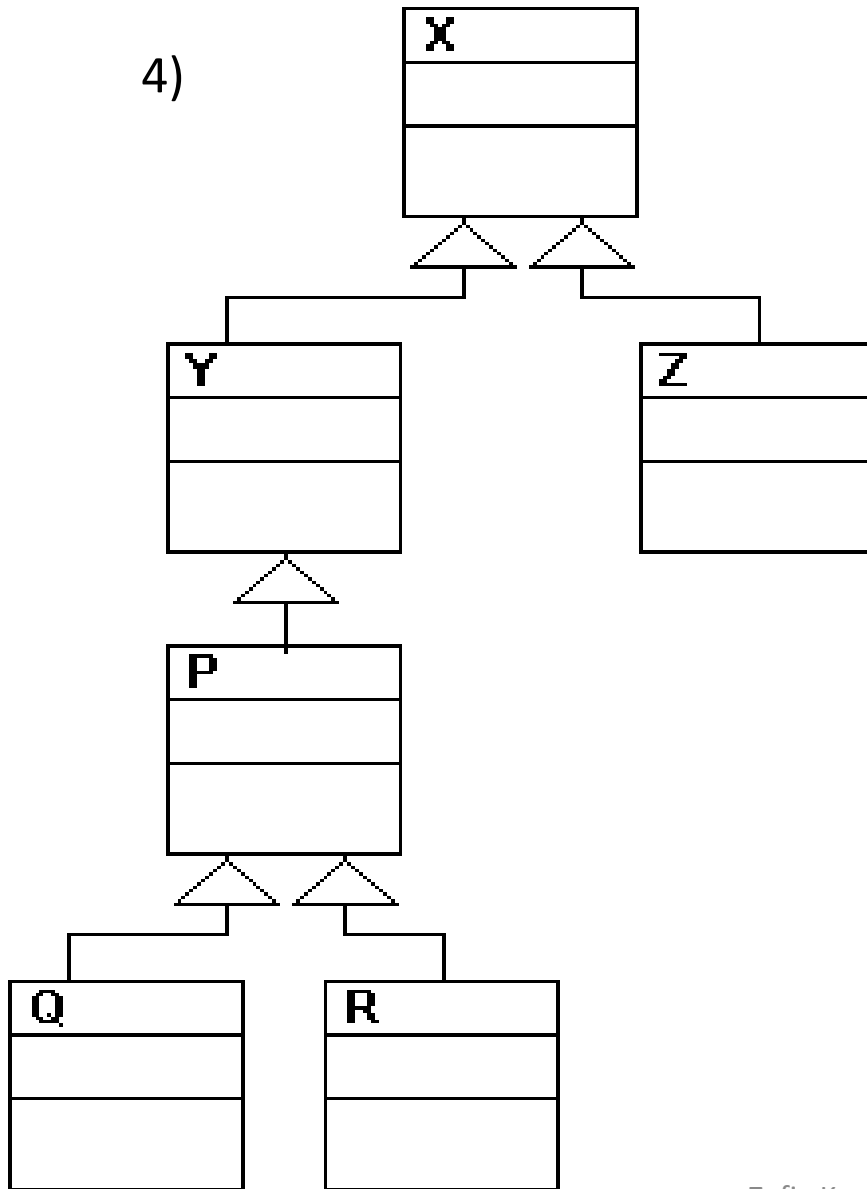
1. Przy głębokim drzewie dziedziczenia rośnie wieloużywalność
2. Przy głębokim drzewie dziedziczenia rośnie też liczba błędów, szczególnie w klasach należących do środkowych poziomów dziedziczenia

# (1) Przykłady modeli do pomiaru metryk dziedziczenia

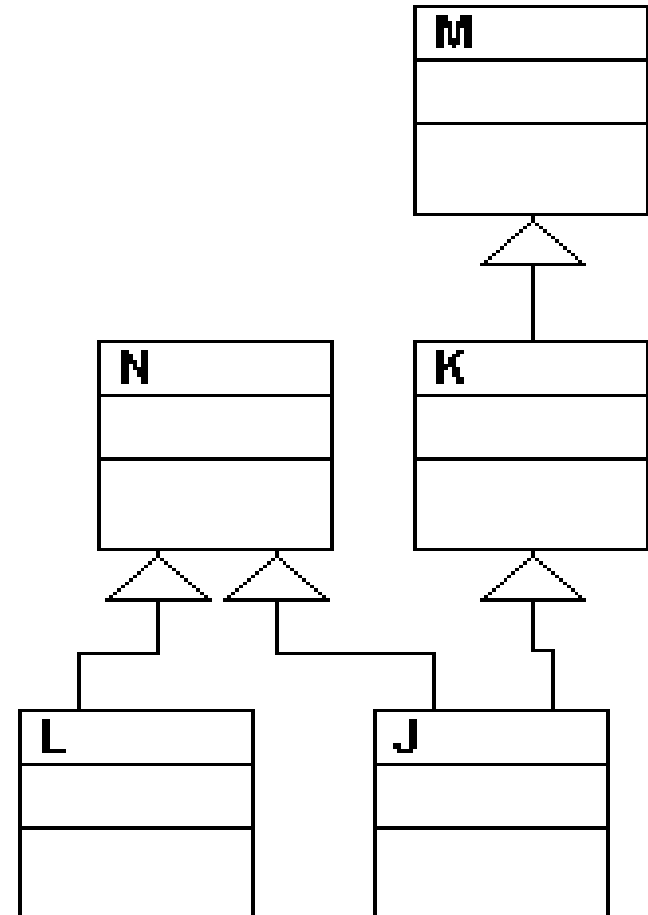


## (2) Przykłady modeli do pomiaru metryk dziedziczenia

4)



5)



| Metryka  | S (specialization) | U (reuse) | DIT                         |
|----------|--------------------|-----------|-----------------------------|
| Przykład |                    |           |                             |
| 1*       | 1/3 → 0            | 3/4 → 1   | (0+1+2+3)/4=1.5             |
| 2*       | 1/4 → 0            | 4/5 → 1   | (0+0+0+0+(1+1+1+1)/4)/5=0.2 |
| 3        | 4/1 → μ            | 1/5 → 0   | (0+1+1+1+1)/5=0.8           |
| 4        | 3/3                | 3/6       | (0+1+1+2+3+3)/6=1.5         |
| 5        | 2/3                | 3/5       | (0+0+1+(1+2)/2+1)/5=0.7     |

$$DIT = \frac{\sum \text{glebokosc dziedziczenia}}{\text{calkowita liczba klas}}$$

$$S = \frac{\text{liczba PodTypów}}{\text{liczba NadTypów}}$$

$$U = \frac{\text{liczba NadTypów}}{\text{calkowita liczba klas}}$$

Przykłady 1 i 2 reprezentują ubogi schemat dziedziczenia.

Wartości  $U$  bliska 1 oraz  $S$  bliską 0 określają liniowy model dziedziczenia.

**Wartości  $U \ll 1$  oraz  $S \gg 1$  oznaczają pożądaną wartość.**

# NOC – liczba klas dziedziczących

## Zakres wartości (0..10)

### Uwagi

1. Zbyt dużo podklas oznacza dużo testowania
2. Zbyt dużo podklas może powodować błędne użycie tych podklas

# Metryki logicznej struktury programu, czyli przepływu sterowania

## Liczby cyklomatyczne McCabe

### Zakres wartości (1 -10)

$$VLI(G) = e - n + p + 1$$

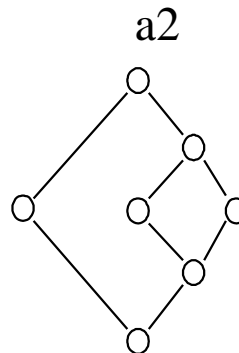
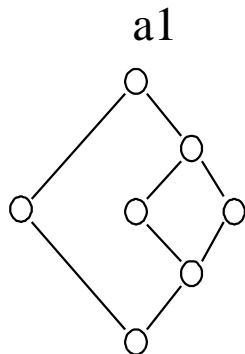
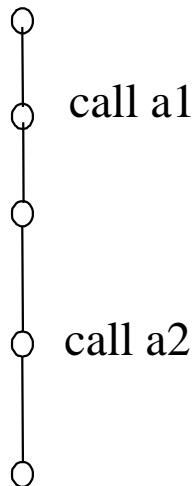
Liczba ta jest wyznaczana na podstawie grafu przedstawiającego drogi sterowania w programie, gdzie  $n$  jest liczbą wierzchołków grafu reprezentujących poszczególne instrukcje, w tym wywołania funkcji,  $e$  jest liczbą krawędzi grafu reprezentujących połączenia poszczególnych realizacji instrukcji,  $p$  jest liczbą podgrafów rozłącznych, a każda funkcja stanowi niezależny podgraf, którego wywołanie jako wierzchołek jest umieszczony w innym podgrafie.

$$V(G) = e - n + 2 * p$$

Metryka  $V(G)$  uwypukla istnienie funkcji za pomocą składnika  $2 * p$ ,  $VLI(G)$  natomiast wywołanie funkcji traktuje na równi z innymi instrukcjami.

# (1) Przykład prezentujący obliczenia metryk MC Cabe

a)



**Cała aplikacja**

a)  $e=20, n=19, p=3$

$$V(G) = e - n + 2 * p = 20 - 19 + 2 * 3 = 7$$

$$V_{LI}(G) = e - n + p + 1 = 20 - 19 + 3 + 1 = 5$$

b)  $e=23, n=20, p=1$

$$V(G) = V_{LI}(G)$$

$$= e - n + 2 = e - n + 2 * p$$

$$= 23 - 20 + 2 = 5$$

**Metody a1 i a2 (przypadki a i b):**

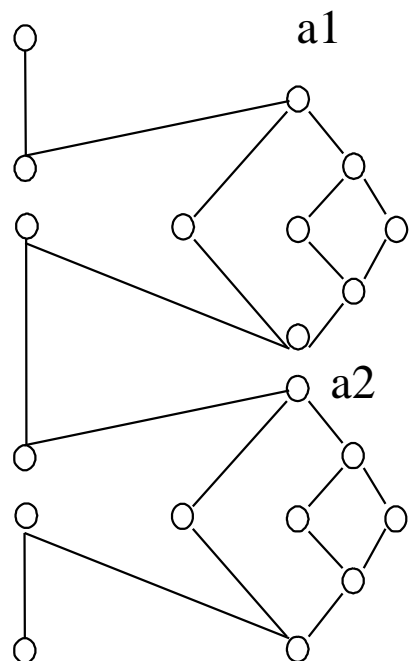
$e=8, n=7, p=1$

$$V(G) = V_{LI}(G) =$$

$$= e - n + 2 = e - n + 2 * p$$

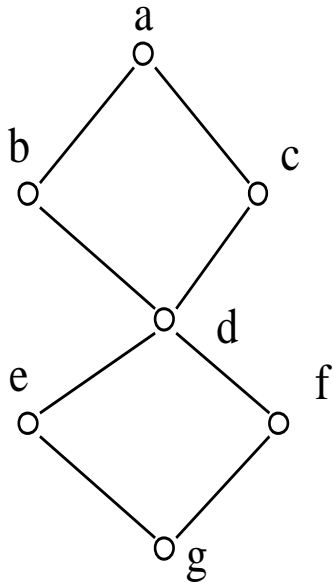
$$= 8 - 7 + 2 = 3$$

b)



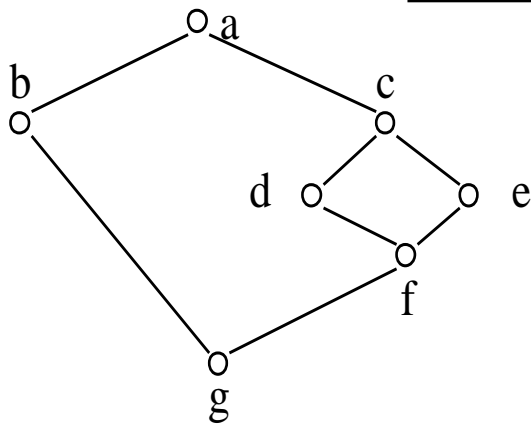
|                     | Metoda a1 | Metoda a2 | Całość          |
|---------------------|-----------|-----------|-----------------|
| a                   |           |           |                 |
| V(G)                | 3         | 3         | 7               |
| V <sub>LI</sub> (G) | 3         | 3         | 5               |
| b                   |           |           |                 |
| V(G)                | 3         | 3         | 5               |
| V <sub>LI</sub> (G) | 3         | 3         | 5 <sub>39</sub> |

## (2) Przykład prezentujący obliczenia metryk MC Cabe



Zgodnie z aksjomatem 7, pętla zagnieżdżona powinna mieć złożoność różną od programu z dwiema sekwencyjnie wykonywanymi pętlami.

Jednak zarówno SLOC,  $V(G)$ ,  $VLI(G)$  są identyczne w obu rozwiązaniach, natomiast różne są wartości metryki  $S/C$ . Wg metryki  $S/C$  bardziej złożony jest program z zagnieżdżoną pętlą.



dwie pętle sekwencyjne

a: **while** ( $x \geq 0$ )

c: {  $x=x-y$ ; } (gdy  $a==true$ )

b: (gdy  $a==false$ )

d: **while** ( $y \geq 10$ ) (koniec a)

f: {  $x=x+1$ ; } (gdy  $d==true$ )

$y=y-1$ ;

}

e: (gdy  $d==false$ )

g: (koniec d, koniec programu)

**$V(G)=e-n+2*p=3$**

**$VLI(G)=e-n+p+1=8-7+2=3$**

**SLOC=7**

**$S/C=7$**

b) podwójna pętla zagnieżdżona

a: **while** ( $x \geq 0$ )

{  $x=x-y$ ; } (gdy  $a==true$ )

c: **while** ( $y \geq 10$ ) (gdy  $a==true$ )

e: {  $x=x+1$ ; } (gdy  $c==true$  i  $a==true$ )

$y=y-1$ ;

d: (gdy  $c==false$  i  $a==true$ )

f: (koniec c i  $a==true$ )

}

b: (gdy  $a==false$ )

g: (koniec a, koniec programu)

**$V(G)=e-n+2*p=3$**

**$VLI(G)=e-n+p+1=8-7+2=3$**

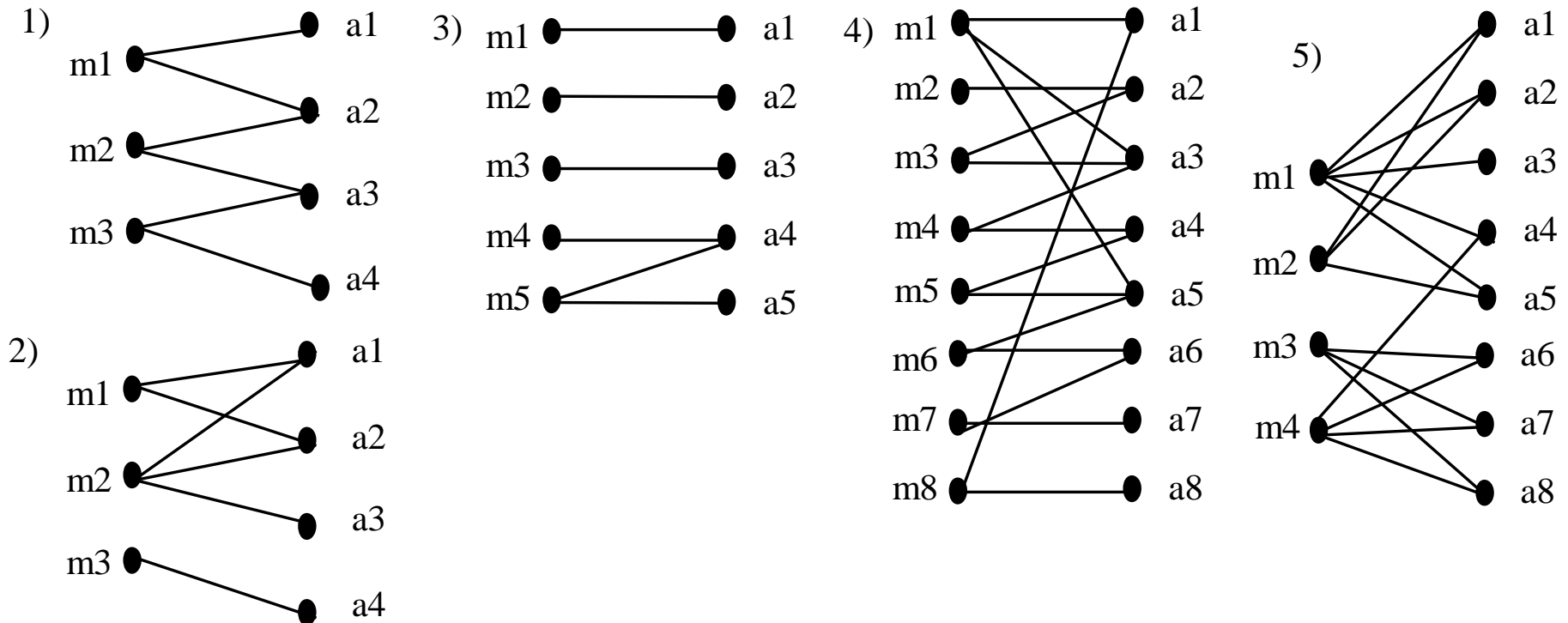
**SLOC=7**

**$S/C=9$**



# Metryki spójności klasy

## Grafy dwudzielne jako modele klasy do wyznaczania metryk spójności LCOM



# Metryka LCOM1

**LCOM1** – metryka wyznacza sumę  $P$  zbioru wszystkich par metod operujących na zbiorach rozłącznych atrybutów oraz sumę  $Q$  zbioru wszystkich par metod operujących na zbiorach spójnych atrybutów. Różnica mocy tych zbiorów jest wartością metryki, gdy moc  $|P|$  jest większa od mocy  $|Q|$ , w przeciwnym wypadku jest równa 0.

Jeśli klasa jest minimalnie spójna (żadna metoda nie jest powiązana z inną metodą i liczba metod jest równa  $n$ ).

Wtedy  $|P| = (n-1)*n/2$  i  $|Q| = 0$ , czyli  $LCOM1 = (n-1)*n/2$

Uwagi:

- 1) Duża wartość metryki oznacza trudność testowania,
- 2) jednak mała wartość lub równa 0 nie zawsze oznacza klasę poprawnie zbudowaną.
- 3) Zbyt wiele różnych klas ma tę samą wartość metryki.
- 4) Brak modelowania property i uwzględnienia wywoływania metody przez metodę

# (1) Przykłady obliczeń metryki LCOM1

## 1) – trzy metody

- Metoda 1 ma zbiór atrybutów  $I_1 = \{a_1, a_2\}$
- Metoda 2 ma zbiór atrybutów  $I_2 = \{a_2, a_3\}$
- Metoda 3 ma zbiór atrybutów:  $I_3 = \{a_3, a_4\}$
- Zbiór rozłącznych par:  $P = \{(I_1, I_3)\} \rightarrow |P| = 1$
- Zbiór spójnych par:  $Q = \{(I_1, I_2), (I_2, I_3)\} \rightarrow |Q| = 2$
- **LCOM = 0 dla  $|P| \leq |Q|$**

## 2) - trzy metody

- Metoda 1 ma zbiór atrybutów  $I_1 = \{a_1, a_2\}$
- Metoda 2 ma zbiór atrybutów  $I_2 = \{a_1, a_2, a_3\}$
- Metoda 3 ma zbiór atrybutów:  $I_3 = \{a_4\}$
- Zbiór rozłącznych par:  $P = \{(I_1, I_3), (I_2, I_3)\} \rightarrow |P| = 2$
- Zbiór spójnych par:  $Q = \{(I_1, I_2)\} \rightarrow |Q| = 1$
- **LCOM =  $|P| - |Q| = 2 - 1 = 1$  dla  $|P| > |Q|$**

## 3) – pięć metod

- Metoda 1 ma zbiór atrybutów  $I_1 = \{a_1\}$
- Metoda 2 ma zbiór atrybutów  $I_2 = \{a_2\}$
- Metoda 3 ma zbiór atrybutów:  $I_3 = \{a_3\}$
- Metoda 4 ma zbiór atrybutów:  $I_4 = \{a_4\}$
- Metoda 5 ma zbiór atrybutów:  $I_5 = \{a_4, a_5\}$
- Zbiór rozłącznych par:  $P = \{(I_1, I_2), (I_1, I_3), (I_1, I_4), (I_1, I_5), (I_2, I_3), (I_2, I_4), (I_2, I_5), (I_3, I_4), (I_3, I_5)\} \rightarrow |P| = 9$
- Zbiór spójnych par:  $Q = \{(I_4, I_5)\} \rightarrow |Q| = 1$
- **LCOM =  $|P| - |Q| = 9 - 1 = 8$  dla  $|P| > |Q|$**

## (2) Przykłady obliczeń metryki LCOM1

### 5) – cztery metody

- Metoda 1 ma zbiór atrybutów  $I_1 = \{a_1, a_2, a_3, a_4, a_5\}$
- Metoda 2 ma zbiór atrybutów  $I_2 = \{a_1, a_2, a_5\}$
- Metoda 3 ma zbiór atrybutów:  $I_3 = \{a_6, a_7, a_8\}$
- Metoda 4 ma zbiór atrybutów:  $I_4 = \{a_4, a_6, a_7, a_8\}$
- Zbiór rozłącznych par:  $P = \{(I_1, I_3), (I_2, I_3), (I_2, I_4)\} \rightarrow |P| = 3$
- Zbiór spójnych par:  $Q = \{(I_1, I_2), (I_1, I_4), (I_3, I_4)\} \rightarrow |Q| = 3$
- **LCOM = 0 dla  $|P| \leq |Q|$**

### 4) – osiem metod

- Metoda 1 ma zbiór atrybutów  $I_1 = \{a_1, a_3, a_5\}$
- Metoda 2 ma zbiór atrybutów  $I_2 = \{a_2\}$
- Metoda 3 ma zbiór atrybutów:  $I_3 = \{a_2, a_3\}$
- Metoda 4 ma zbiór atrybutów:  $I_4 = \{a_3, a_4\}$
- Metoda 5 ma zbiór atrybutów:  $I_5 = \{a_4, a_5\}$
- Metoda 6 ma zbiór atrybutów:  $I_6 = \{a_5, a_6\}$
- Metoda 7 ma zbiór atrybutów:  $I_7 = \{a_6, a_7\}$
- Metoda 8 ma zbiór atrybutów:  $I_8 = \{a_1, a_8\}$
- Zbiór rozłącznych par:  $P = \{(I_1, I_2), (I_1, I_4), (I_1, I_6), (I_1, I_7), (I_2, I_4), (I_2, I_5), (I_2, I_6), (I_2, I_7), (I_2, I_8), (I_3, I_5), (I_3, I_6), (I_3, I_7), (I_3, I_8), (I_4, I_6), (I_4, I_7), (I_4, I_8), (I_5, I_7), (I_5, I_8), (I_6, I_8), (I_7, I_8)\}$   
 $\rightarrow |P| = 20$
- Zbiór spójnych par:  $Q = \{(I_1, I_3), (I_1, I_5), (I_1, I_8), (I_2, I_3), (I_3, I_4), (I_4, I_5), (I_5, I_6), (I_6, I_7), (I_7, I_8)\}$   
 $\rightarrow |Q| = 8$
- **LCOM =  $|P| - |Q| = 20 - 8 = 12$  dla  $|P| > |Q|$**

# Rozszerzenie definicji metryk spójności LCOM (1)

## Metryka LCOM2 (Constantine & Graham, Henderson-Sellers)

$$LCOM2 = 1 - \frac{\left(\frac{1}{a} \sum_{j=1}^a \mu(A_j)\right)}{m} = 1 - \frac{r}{m * a}$$

- gdzie  $m$  jest liczbą wierzchołków zbioru  $M$  metod,  $a$  jest liczbą wierzchołków  $A$  atrybutów, natomiast wyrażenie  $\mu(A_j)$  liczbą krawędzi grafu wiążącą atrybut  $A_j$  z określoną liczbą metod (elementy zbioru  $R$ ).
- Maksymalna i zarazem najlepsza wartość spójności LCOM2 oznacza wartość 0 metryk, co uzyskuje się przy grafie pełnym ( $r = |M| * |A|$  krawędzi).
- Wartość metryki LCOM2 zawarta między „0..mniejszy od 1” oznacza obiektowy model klasy, jednak warta bliska 1 oznacza najgorszy przypadek klasy.
- W metryce LCOM2 muszą przynajmniej istnieć jedna metoda i jeden atrybut.

| Lp | $m$ | $a$ | $r$ | $k$ | $LCOM1$ | $LCOM2$ | $LCOM3$ |
|----|-----|-----|-----|-----|---------|---------|---------|
| 1  | 3   | 4   | 6   | 1   | 0       | 0.5     | 0.75    |
| 2  | 3   | 4   | 6   | 2   | 1       | 0.5     | 0.75    |
| 3  | 5   | 5   | 6   | 4   | 8       | 0.76    | 0.95    |
| 4  | 8   | 8   | 16  | 1   | 0       | 0.75    | 0.8571  |
| 5  | 4   | 8   | 15  | 1   | 12      | 0.5313  | 0.7083  |

## Rozszerzenie definicji metryk spójności LCOM (2)

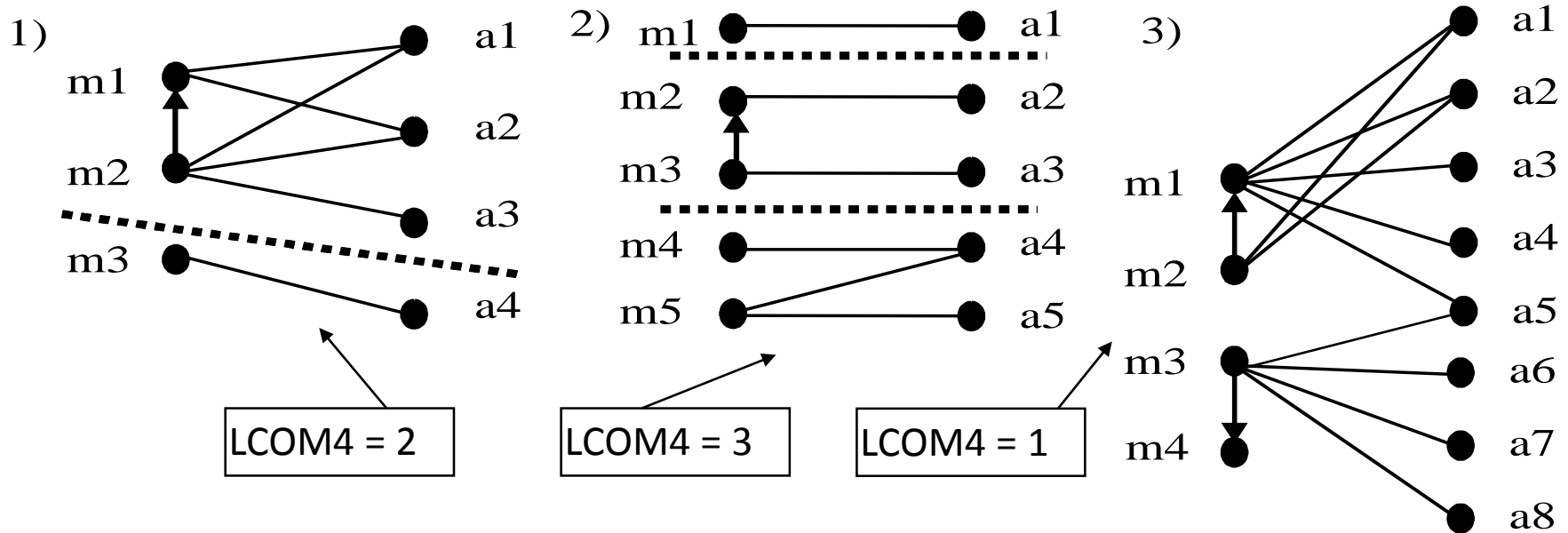
### Metryka LCOM3 (Constantine & Graham, Henderson-Sellers)

Zakres wartości „0..0.2”.

$$LCOM3 = \frac{\left(\frac{1}{a} \sum_{j=1}^a \mu(A_j)\right) - m}{1 - m} = \frac{\frac{r}{a} - m}{1 - m}$$

- gdzie  $m$  jest liczbą wierzchołków zbioru  $M$  metod,  $a$  jest liczbą wierzchołków  $A$  atrybutów, natomiast wyrażenie  $\mu(A_j)$  liczbą krawędzi grafu wiążącą atrybut  $A_j$  z określoną liczbą metod (elementy zbioru  $R$ ).
- Maksymalna i zarazem najlepsza wartość spójności LCOM3 oznacza wartość 0 metryk, co uzyskuje się przy grafie pełnym ( $r=|M|*|A|$  krawędzi).
- Wartość metryki LCOM3 zawarta między „0..1” oznacza obiektowy model klasy (wartość 1 oznacza minimalnie spójną klasę – równa liczby metod i atrybutów). Dopuszczalny zakres „0..0.2”.
- W metryce LCOM3 w klasie nie może istnieć tylko jedna metoda i musi być przynajmniej jeden atrybut.

## LCOM4 - (Hitz & Montazeri)



- **LCOM4** mierzy liczbę „połączonych komponentów” w klasie. „Połączony komponent” jest zbiorem połączonych metod (zbiór takich metod a i b, gdzie metoda a wywołuje metodę b lub metoda b wywołuje metodę a, lub obie metody a i b wywołują ten sam atrybut klasy) i atrybutów, przy czym dopuszcza się jeden taki komponent klasy.
- Jeśli wartość metryki jest równa 2 lub więcej, należy klasę podzielić na dwie klasy lub więcej klas, tak aby posiadała tylko jeden „połączony komponent”.

## **2.6. Przykłady wyników pomiarów metryk złożoności strukturalnej oprogramowania**



## 2.6.1. Przykład metryk trzech systemów

| <b>System analyzed</b> | <b>Java</b> | <b>Java</b> | <b>C++</b> |
|------------------------|-------------|-------------|------------|
| <b>Classes</b>         | 46          | 1000        | 1617       |
| <b>Lines</b>           | 50,000      | 300,000     | 500,000    |
| <b>Quality</b>         | "Low"       | "High"      | "Medium"   |
| <b>CBO</b>             | 2.48        | 1.25        | 2.09       |
| <b>LCOM1</b>           | 447.65      | 78.34       | 113.94     |
| <b>RFC</b>             | 80.39       | 43.84       | 28.60      |
| <b>NOC</b>             | 0.07        | 0.35        | 0.39       |
| <b>DIT</b>             | 0.37        | 0.97        | 1.02       |
| <b>WMC</b>             | 45.7        | 11.10       | 23.97      |

## 2.6.2. Przykład metryk aplikacji typu Java Application z modelem obiektowym opartym na klasach zdefiniowanych przez użytkownika - metryki CK

CKJM Chidamber and Kemerer Java Metrics - Windows Internet Explorer

E:\Dydaktyka\io\Wyklad1pio\ckjm4.html

Google

Google G Go Bookmarks 2 blocked Settings

CKJM Chidamber and K...

Strona Narzędzia

### Top 25: lcom3

[\[wmc\]](#) [\[dit\]](#) [\[noc\]](#) [\[cbo\]](#) [\[rfc\]](#) [\[lcom\]](#) [\[ca\]](#) [\[npm\]](#) [\[lcom3\]](#) [\[explanations\]](#)

| name                 | wmc | dit | noc | cbo | rfc | lcom | ca | npm | lcom3              |
|----------------------|-----|-----|-----|-----|-----|------|----|-----|--------------------|
| tytul1.Tytul_ksiazki | 17  | 1   | 0   | 1   | 29  | 110  | 2  | 17  | 0.84375            |
| ksiazka1.Ksiazka     | 7   | 1   | 0   | 1   | 15  | 9    | 1  | 7   | 0.5833333333333334 |
| uchwy1.Uchwyt        | 7   | 1   | 0   | 1   | 23  | 0    | 0  | 7   | 0.0                |

### Explanations

WMC - Weighted methods per class

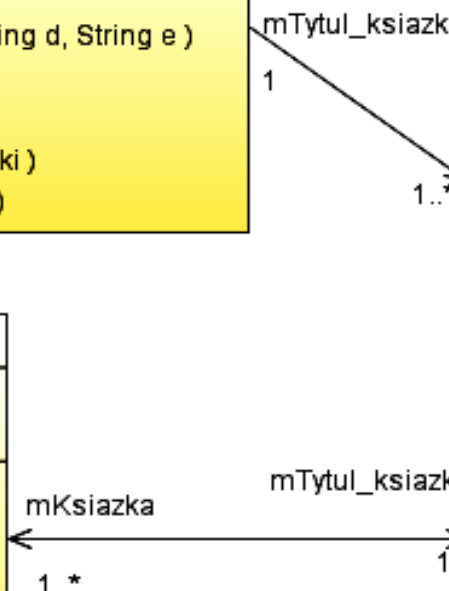
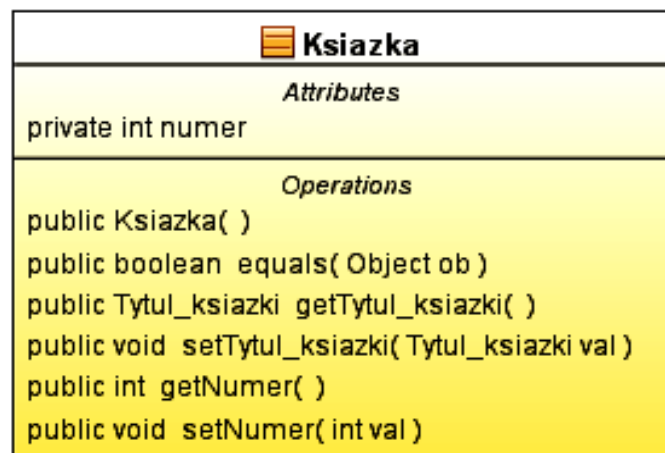
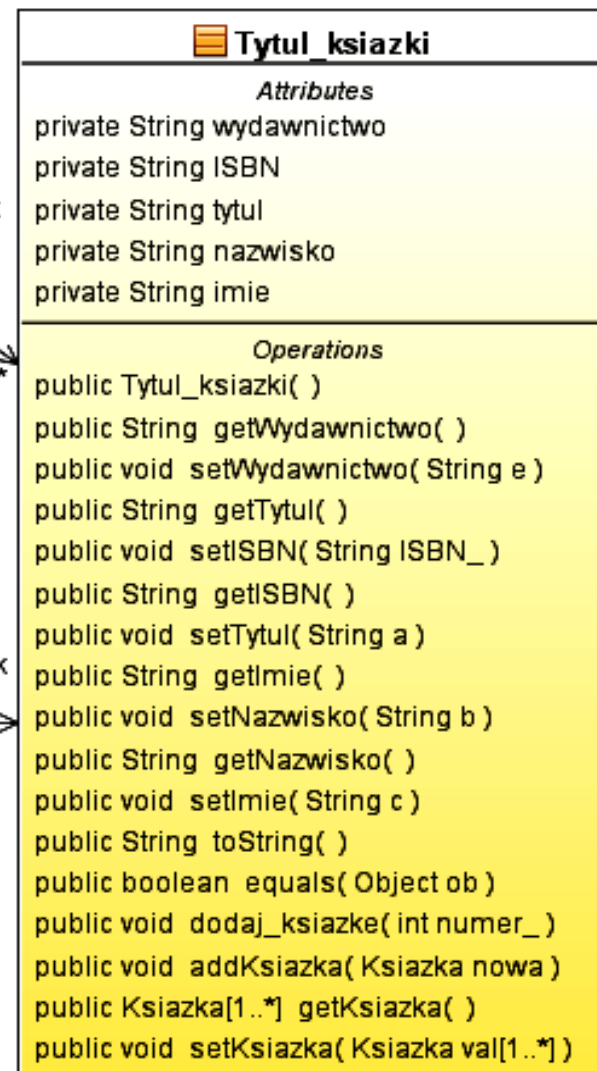
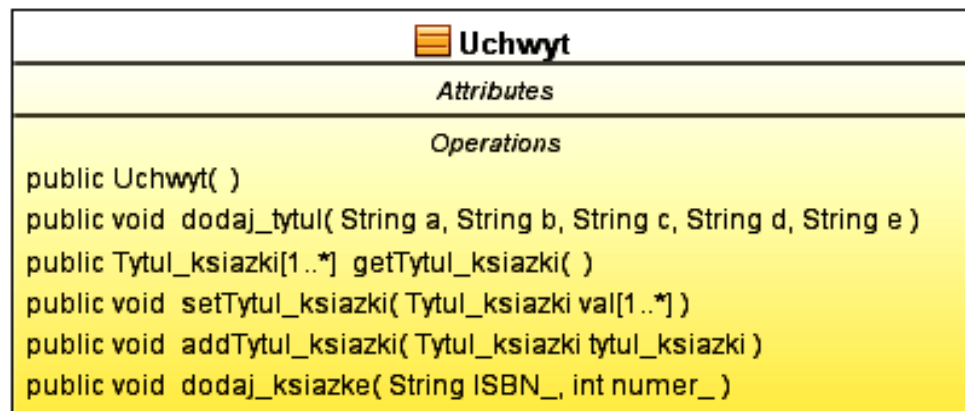
A class's *weighted methods per class* WMC metric is simply the sum of the complexities of its methods. As a measure of complexity we can use the cyclomatic complexity, or we can

2018-11-11

Zofia Kruczkiewicz - Komputer | Tryb chroniony: wyłączony

100%

# Definicja klas `Uchwyty`, `Tytul_książki`, `Książka` po dodaniu przypadku użycia `dodaj_książke`



## 2.6.3. Przykład metryk aplikacji typu Java Application z modelem obiektowym opartym na klasach zdefiniowanych przez użytkownika oraz klasach typu Controller technologii JPA – metryki CK

### Top 25: lcom3

[\[wmc\]](#) [\[dit\]](#) [\[noc\]](#) [\[cbo\]](#) [\[rfc\]](#) [\[lcom\]](#) [\[ca\]](#) [\[npm\]](#) [\[lcom3\]](#) [\[explanations\]](#)

| name                                        | wmc | dit | noc | cbo | rfc | lcom | ca | npm | lcom3              |
|---------------------------------------------|-----|-----|-----|-----|-----|------|----|-----|--------------------|
| wyposzczalnia1app.TFabryka                  | 3   | 1   | 0   | 4   | 20  | 3    | 2  | 3   | 2.0                |
| wyposzczalnia1app.TTytul_ksiazki            | 22  | 1   | 1   | 2   | 36  | 189  | 6  | 22  | 0.9047619047619048 |
| wyposzczalnia1app.TEgzemplarz               | 12  | 1   | 1   | 1   | 20  | 42   | 5  | 12  | 0.8409090909090909 |
| wyposzczalnia1app.TTytul_ksiazki_na_kasecie | 4   | 2   | 0   | 1   | 9   | 4    | 1  | 4   | 0.8333333333333333 |
| wyposzczalnia1app.TEgzemplarz_termin        | 5   | 2   | 0   | 1   | 12  | 0    | 1  | 5   | 0.75               |
| wyposzczalnia1app.TEgzemplarzController     | 9   | 1   | 0   | 2   | 29  | 34   | 0  | 8   | 0.125              |
| wyposzczalnia1app.TTytul_ksiazkiController  | 9   | 1   | 0   | 1   | 34  | 34   | 0  | 8   | 0.125              |
| wyposzczalnia1app.TAplikacja                | 11  | 1   | 0   | 3   | 29  | 13   | 0  | 11  | 0.0                |

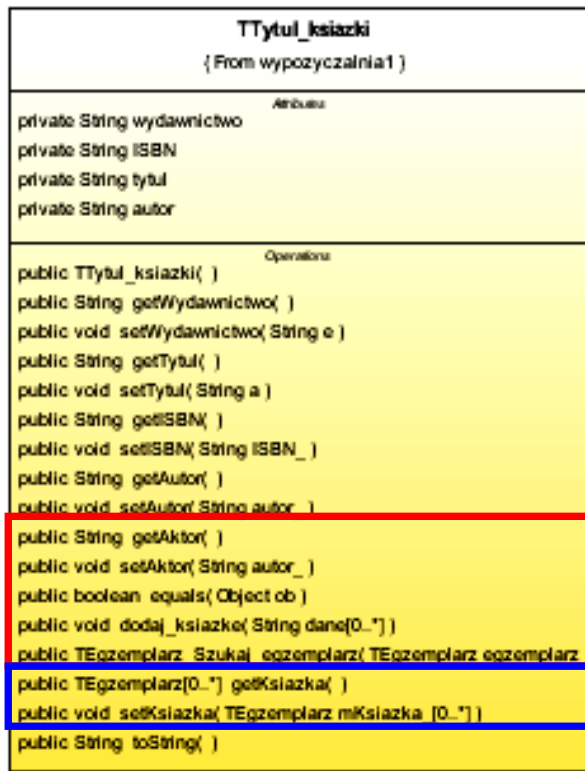
### Explanations

WMC - Weighted methods per class

A class's *weighted methods per class* WMC metric is simply the sum of the complexities of its methods. As a measure of complexity we can use the cyclomatic complexity, or we can arbitrarily assign a complexity value of 1 to each method. The *ckjm* program assigns a complexity value of 1 to each method, and therefore the value of the WMC is equal to the number of methods in the class.

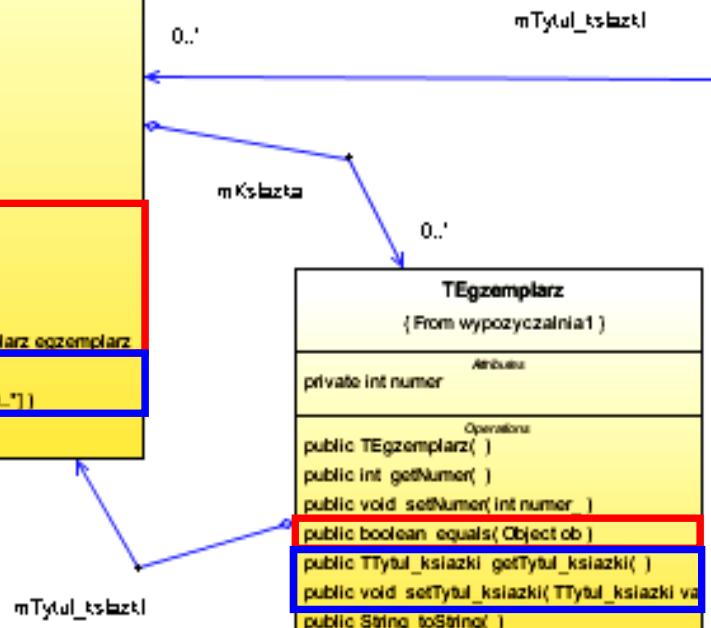
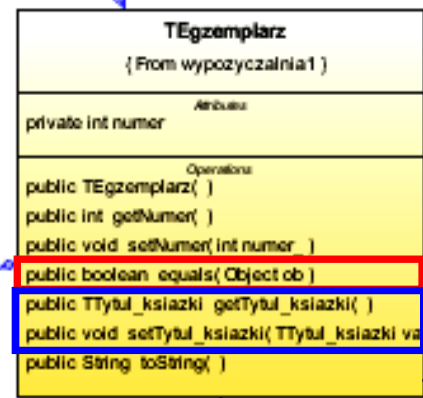
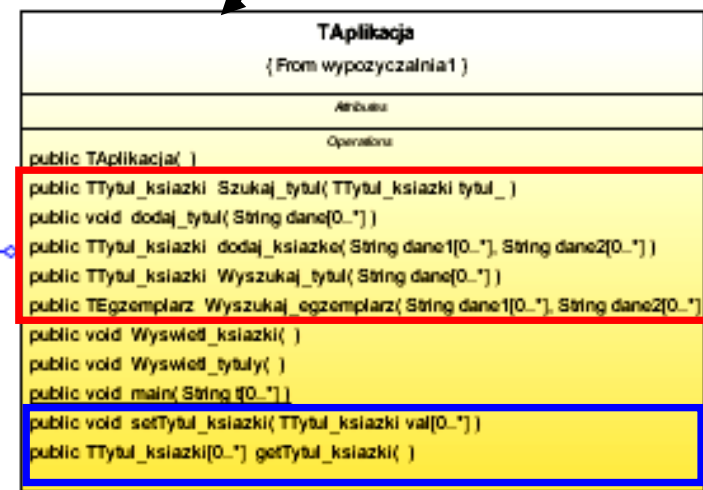
DIT - Depth of Inheritance Tree

The *depth of inheritance tree* (DIT) metric provides for each class a measure of the inheritance



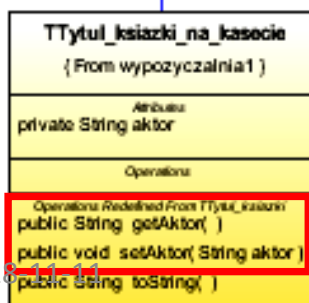
- Implementacja powiązań
- Metody przypadków użycia
- Decyzja projektowa

Wzorzec fasady



Wzorzec strategii

Wzorzec fabryki obiektów



# Dodatek – przykład pomiaru metryk

## 2.6.4. Przykład metryk aplikacji zbudowanej w środowisku Visual Web Java Server Faces - metryki CK: klas typu fasady z warstwy biznesowej (w ramce czerwonej) oraz prezentacji (główne klasy w ramce niebieskiej oraz pomocnicze bez ramki)

### Top 25: lcom3

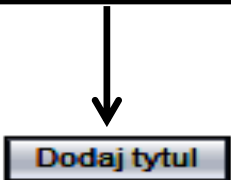
[\[wmc\]](#) [\[dit\]](#) [\[noc\]](#) [\[cbo\]](#) [\[rfc\]](#) [\[lcom\]](#) [\[ca\]](#) [\[npm\]](#) [\[lcom3\]](#) [\[explanations\]](#)

| name                               | wmc | dit | noc | cbo | rfc | lcom | ca | npm | lcom3              |
|------------------------------------|-----|-----|-----|-----|-----|------|----|-----|--------------------|
| webwypożyczalnia2.RequestBean1     | 6   | 3   | 0   | 3   | 11  | 15   | 14 | 3   | 2.0                |
| webwypożyczalnia2.Tytulybaza       | 35  | 3   | 0   | 8   | 44  | 511  | 0  | 31  | 0.9411764705882353 |
| webwypożyczalnia2.Ksiazkibaza      | 27  | 3   | 0   | 8   | 36  | 291  | 0  | 23  | 0.9230769230769231 |
| webwypożyczalnia2.Tytuly           | 26  | 3   | 0   | 15  | 44  | 277  | 0  | 22  | 0.92               |
| webwypożyczalnia2.Menu             | 26  | 3   | 0   | 5   | 32  | 283  | 6  | 22  | 0.92               |
| webwypożyczalnia2.Baza_tytul       | 26  | 3   | 0   | 15  | 45  | 277  | 0  | 22  | 0.92               |
| webwypożyczalnia2.Baza_ksiazki     | 24  | 3   | 0   | 13  | 40  | 234  | 0  | 20  | 0.9130434782608695 |
| webwypożyczalnia2.Baza_tytuly      | 24  | 3   | 0   | 13  | 40  | 234  | 0  | 20  | 0.9130434782608695 |
| webwypożyczalnia2.Ksiazki          | 24  | 3   | 0   | 18  | 49  | 234  | 0  | 20  | 0.9130434782608695 |
| webwypożyczalnia2.Page1            | 22  | 3   | 0   | 12  | 40  | 195  | 0  | 18  | 0.9047619047619048 |
| webwypożyczalnia2.FormTytul        | 31  | 3   | 0   | 6   | 43  | 349  | 2  | 27  | 0.9                |
| webwypożyczalnia2.FormKsiazka      | 19  | 3   | 0   | 6   | 31  | 115  | 1  | 15  | 0.8333333333333334 |
| webwypożyczalnia2.Logo             | 11  | 3   | 0   | 6   | 18  | 43   | 0  | 7   | 0.8                |
| webwypożyczalnia2.ApplicationBean1 | 25  | 3   | 0   | 7   | 52  | 234  | 16 | 24  | 0.7666666666666666 |
| webwypożyczalnia2.SessionBean1     | 9   | 3   | 0   | 2   | 14  | 30   | 15 | 7   | 0.75               |
| webwypożyczalnia2.Ksiazkiaplikacja | 9   | 3   | 0   | 5   | 15  | 30   | 0  | 5   | 0.75               |
| webwypożyczalnia2.Tytulyaplikacja  | 11  | 3   | 0   | 8   | 25  | 43   | 1  | 7   | 0.6                |



## Przykład wielowarstwowej aplikacji

Generowanie zdarzenia



- Strona główna
- Dodaj tytuły w aplikacji
- Dodaj książki w aplikacji
- Dodaj tytuł do bazy
- Dodaj książki do bazy
- Przepisz tytuły do bazy
- Przepisz książki do bazy

**Tytuł**

**Autor**

**ISBN**

**Wydawnictwo**

**Aktor**

- Tytuł: 1 Autor: 1 ISBN: 1 Wydawnictwo: 1
- Tytuł: 1 Autor: 1 ISBN: 1 Wydawnictwo: 1**
- Tytuł: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2
- Tytuł: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2
- Tytuł: 3 Autor: 3 ISBN: 3 Wydawnictwo: 3
- Tytuł: 3 Autor: 3 ISBN: 3 Wydawnictwo: 3 Aktor: 3
- Tytuł: 4 Autor: 4 ISBN: 4 Wydawnictwo: 4
- Tytuł: 5 Autor: 5 ISBN: 5 Wydawnictwo: 5 Aktor: 5
- Tytuł: 6 Autor: 6 ISBN: 6 Wydawnictwo: 6





# Przykład wielowarstwowej aplikacji

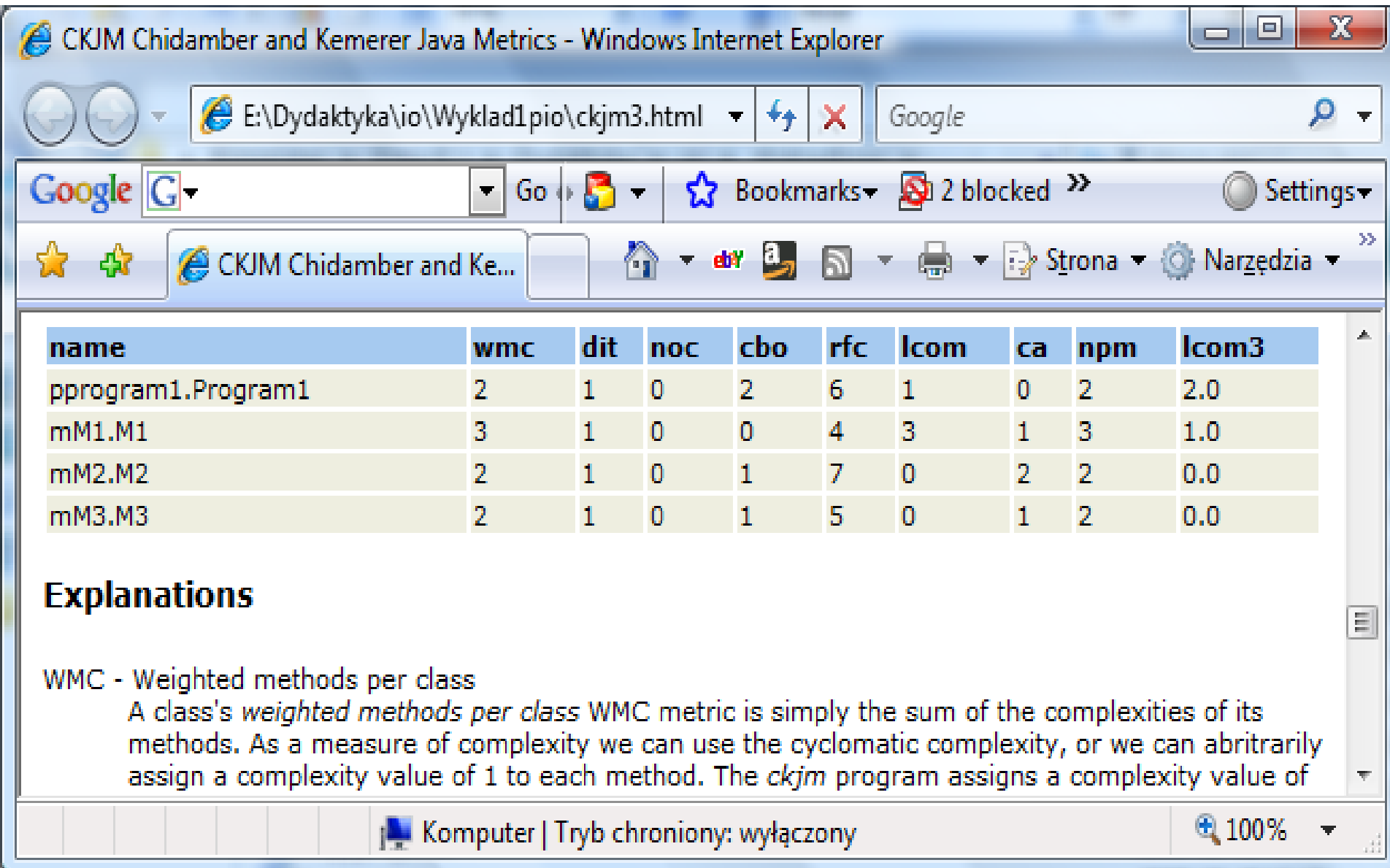
Generowanie zdarzenia

Zapisz tytuły do bazy

- Strona główna
- Dodaj tytuły w aplikacji
- Dodaj książki w aplikacji
- Dodaj tytuł do bazy
- Dodaj książki do bazy
- Przepisz tytuły do bazy
- Przepisz książki do bazy

| Tytuły |       |       |      |             |       |
|--------|-------|-------|------|-------------|-------|
| id     | tytuł | autor | ISBN | wydawnictwo | aktor |
| 2      | 1     | 1     | 1    | 1           |       |
| 4      | 2     | 2     | 2    | 2           | 2     |
| 152    | 2     | 2     | 2    | 2           |       |
| 252    | 3     | 3     | 3    | 3           |       |
| 352    | 3     | 3     | 3    | 3           | 3     |
| 353    | 4     | 4     | 4    | 4           |       |
| 452    | 5     | 5     | 5    | 5           | 5     |
| 453    | 6     | 6     | 6    | 6           |       |

## 2.6.5. Przykład – program typu Java Application do wyznaczania pierwiastków równania kwadratowego – metryki CK



CKJM Chidamber and Kemerer Java Metrics - Windows Internet Explorer

E:\Dydaktyka\io\Wyklad1pio\ckjm3.html

Google

Go

Bookmarks

2 blocked

Settings

CKJM Chidamber and Ke...

Strona

Narzędzia

| name               | wmc | dit | noc | cbo | rfc | lcom | ca | npm | lcom3 |
|--------------------|-----|-----|-----|-----|-----|------|----|-----|-------|
| pprogram1.Program1 | 2   | 1   | 0   | 2   | 6   | 1    | 0  | 2   | 2.0   |
| mM1.M1             | 3   | 1   | 0   | 0   | 4   | 3    | 1  | 3   | 1.0   |
| mM2.M2             | 2   | 1   | 0   | 1   | 7   | 0    | 2  | 2   | 0.0   |
| mM3.M3             | 2   | 1   | 0   | 1   | 5   | 0    | 1  | 2   | 0.0   |

### Explanations

WMC - Weighted methods per class

A class's *weighted methods per class* WMC metric is simply the sum of the complexities of its methods. As a measure of complexity we can use the cyclomatic complexity, or we can arbitrarily assign a complexity value of 1 to each method. The *ckjm* program assigns a complexity value of

Komputer | Tryb chroniony: wyłączony

100%

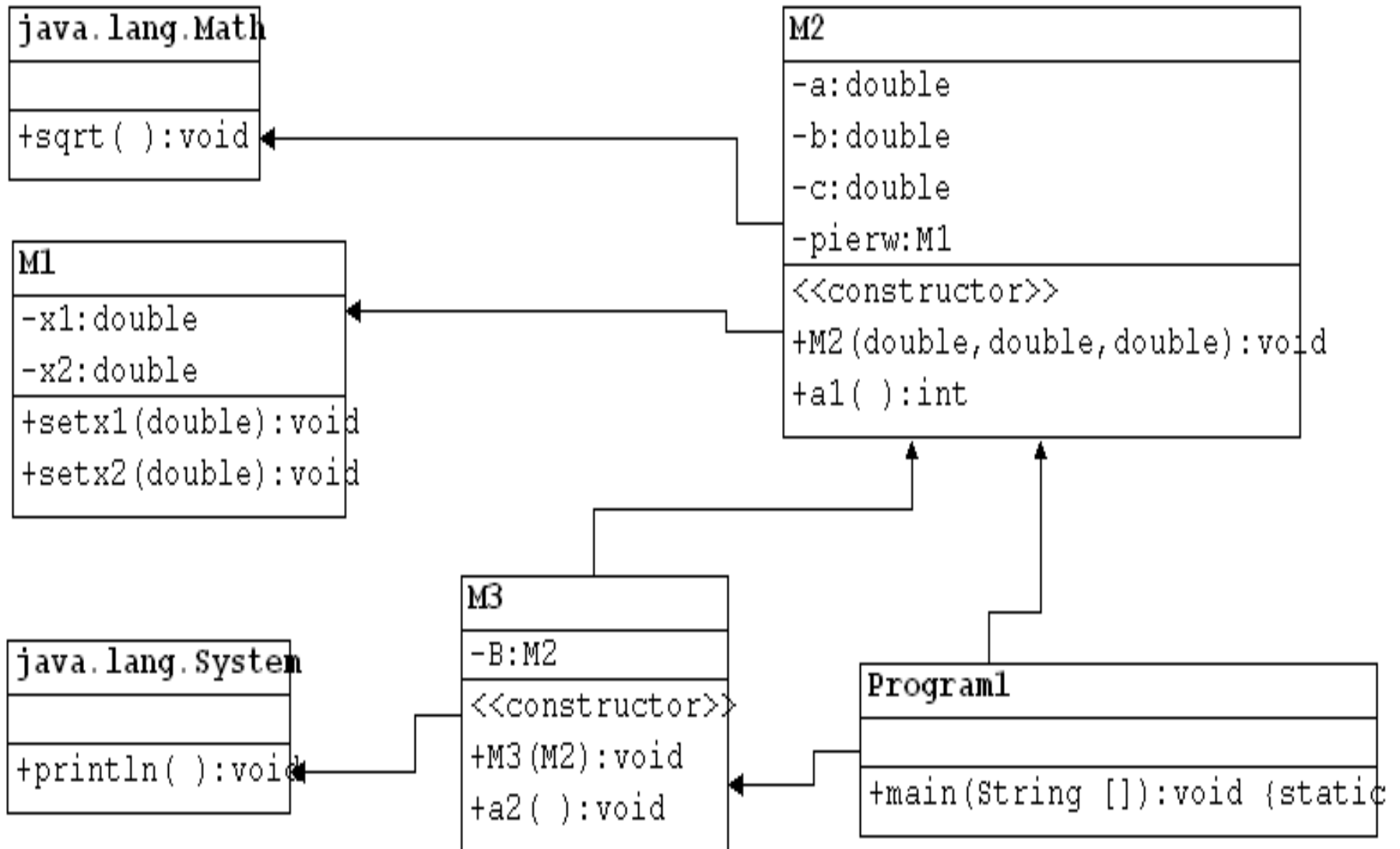
# Schemat powiązań międzymodułowych do pomiaru metryk międzymodułowych

The screenshot displays the NetBeans IDE 5.5.1 interface with the 'Draw Dependence...' window open. The diagram shows the following structure:

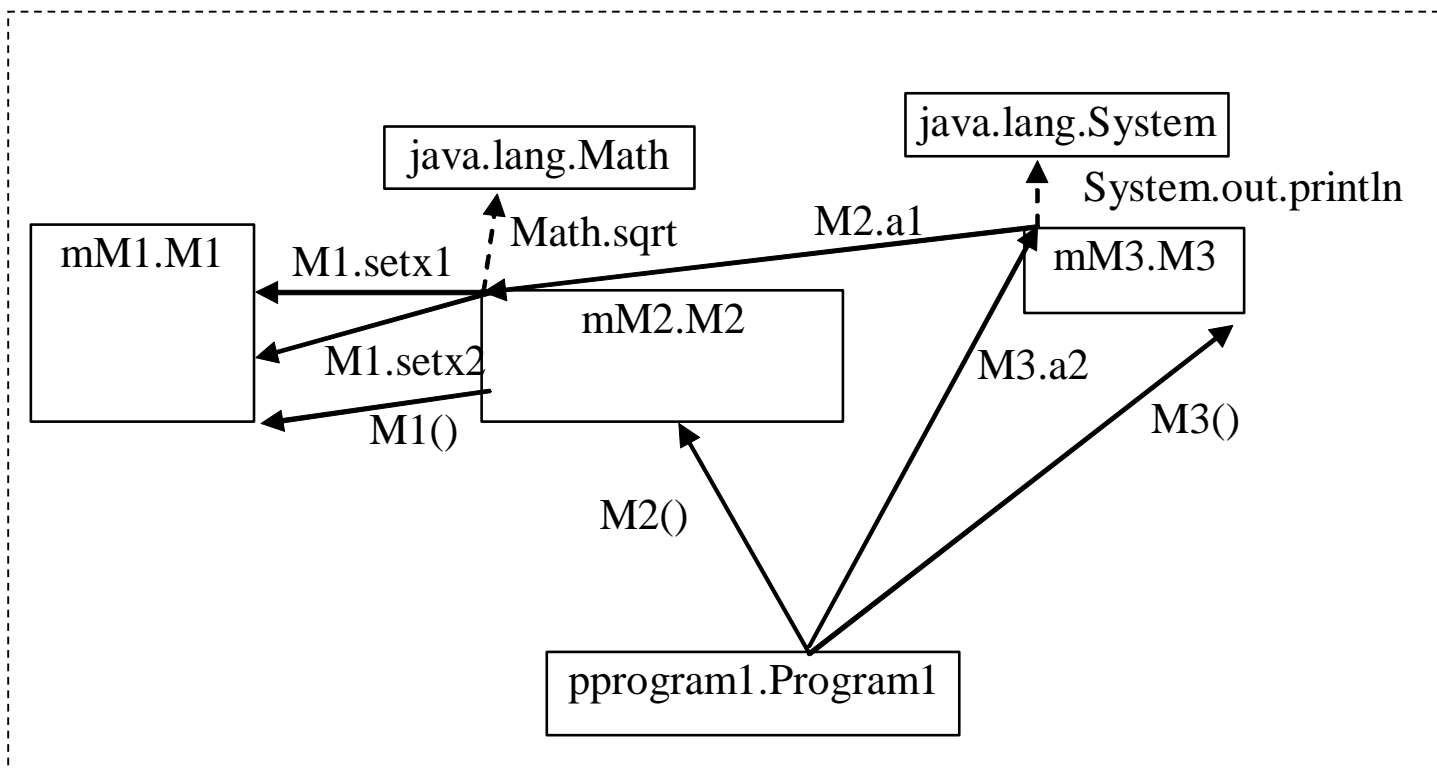
- Modules:** mM1, mM2, and mM3 are represented as green rounded rectangles at the top. Below them are M1, M2, and M3, represented as blue rounded rectangles with a gear icon.
- Program:** 'Program1' is a blue rounded rectangle with a gear icon, positioned below 'pprogram1'.
- Dependencies:** Blue vertical lines connect mM1 to M1, mM2 to M2, and mM3 to M3. Red arrows point from 'Program1' to M1, M2, and M3.

The 'Draw Dependence...' window includes a toolbar with options: Freeze (checked), More Details (checked), Show cycles (unchecked), and Zoom. The status bar at the bottom shows the date '2018-11-11', the name 'Zofia Kruczkiewicz', and the slide number 'Wyklad\_INP002017\_11'.

# Diagram klas badanego programu

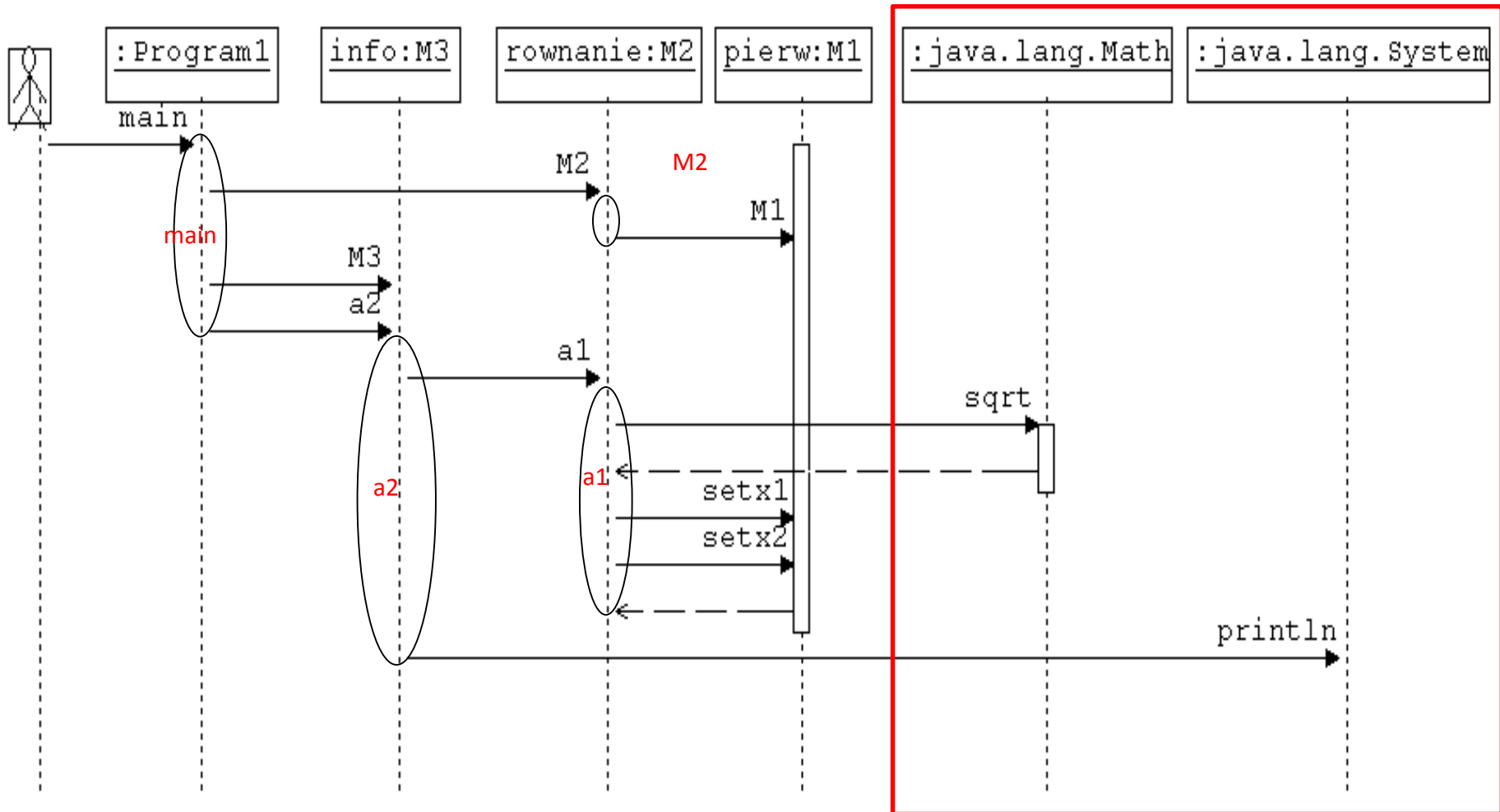


# Schemat do pomiaru metryk połączeń międzymodułowych – Java SE

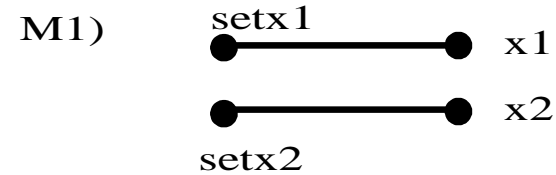
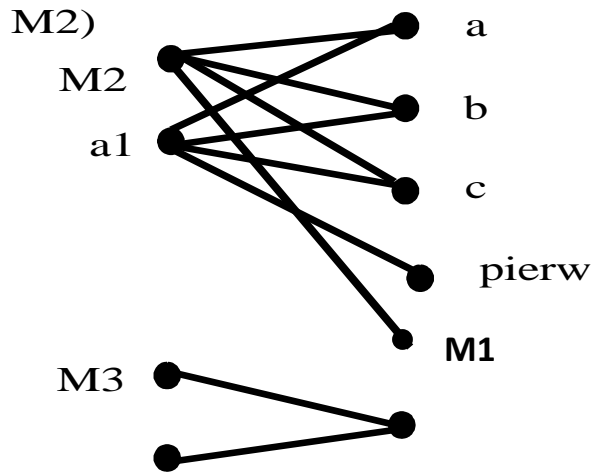


|         | mM1 | mM2       | mM3       | pprogram1 | java.lang.System | java.lang.Math |
|---------|-----|-----------|-----------|-----------|------------------|----------------|
| Fan-out | -   | 1 + (1)   | 1+ (1)    | 2         | -                | -              |
| Fan-in  | 1   | 2         | 1         | -         | (1)              | (1)            |
| RFC     | 2+2 | 2+1+3+(1) | 2+1+1+(1) | 2+1+3     | -                | -              |
| R       | -   | 3+(1)     | 1+(1)     | 3         | -                | -              |

# Korzystanie z diagramu sekwencji do pomiaru metryk powiązań międzymodułowych – równoważny model do wyznaczania metryki RFC



# Obliczanie metryk spójności LCOM



## Spójność klasy M1

- $a=2, m=2, r=2$   $LCOM1 = 1$

$$LCOM4 = 2$$

$$LCOM2 = 1 - \frac{r}{m * a} = 0.5$$

$$LCOM3 = \frac{\frac{r}{a} - m}{1 - m} = 1$$

## Spójność klasy M3

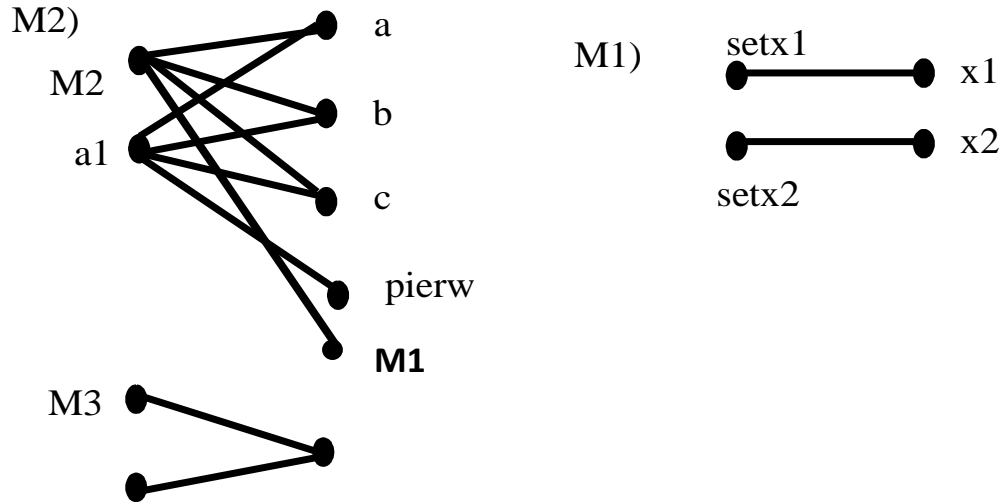
- $a=1, m=2, r=2$   $LCOM1 = 0$

$$LCOM4 = 1$$

$$LCOM2 = 1 - \frac{r}{m * a} = 0$$

$$LCOM3 = \frac{\frac{r}{a} - m}{1 - m} = \frac{\frac{2}{1} - 2}{1 - 2} = 0$$

# Obliczanie metryk spójności LCOM cd



## Spójność klasy M2

$a=4, m=2, r=8$

$$LCOM1 = 1$$

$$LCOM4 = 1$$

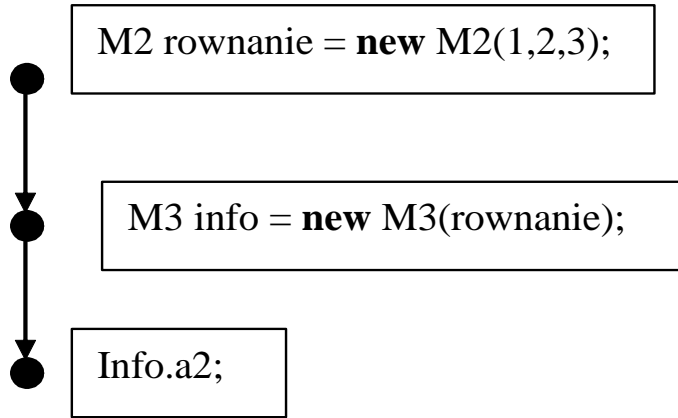
$$LCOM2 = 1 - \frac{r}{m * a} = 0$$

$$LCOM3 = \frac{r - m}{1 - m} = \frac{8 - 2}{1 - 2} = 0$$



# Wyznaczanie metryk MC Cabe

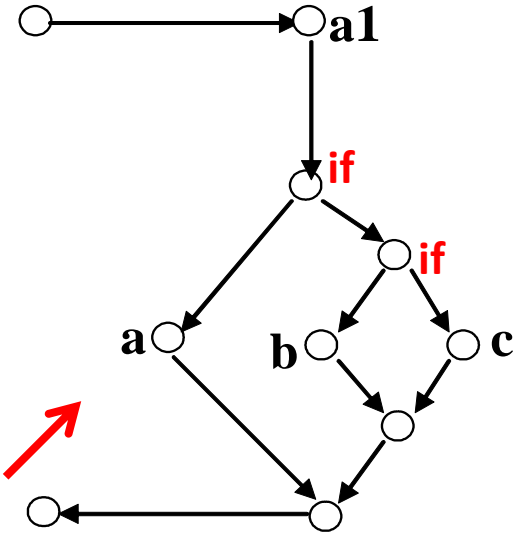
**Program1::main:  $Vli(G)=V(G)=1$**



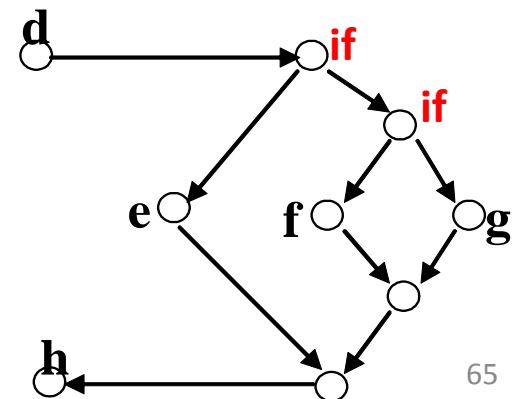
- **a1:** int b=B.a1();
- **a:** System.out.println("Brak równania kwadratowego\n");
- **b:** System.out.println("Brak pierwiastków rzeczywistych\n");
- **c:** System.out.println("Równanie ma pierwiastki rzeczywiste\n");

- **d:** int B; double pom=2\*a, d=b\*b-4\*a\*c;
- **e:** B=0;
- **f:** B=1;
- **g:** B=2;
- **d=** Math.sqrt(d);  
 pierw.setx1((-b-d)/pom);  
 pierw.setx2((-b+d)/pom);
- **h:** return B;

**M3::a2:  $Vli(G)=V(G)=3$**



**M2::a1:  $Vli(G)=V(G)=3$**



# Kod źródłowy klasy M1

```
package Rownanie;
```

```
public class M1 {
```

```
 double x1, x2;
```

```
 /* public M1()
```

```
 { super(); }*/ // wartość 2 dodawana do metryki RFC w każdej klasie
```

```
 public void setx1(double x1_)
```

```
 { x1 = x1_; }
```

```
 public void setx2(double x2_)
```

```
 { x2 = x2_; }
```

```
}
```

# SMC - Metryki kodu źródłowego klasy M1

- **LOC** | Total LOC: 11, Classes LOC: M1: 11, Packages LOC:
- **Lines with imports** | Total imports: 0, Classes imports:nM1: 0, Packages imports:
- **Blank lines** | Total blank lines:0, Classes blank lines:M1:0, Packages blank lines:
- **Classes count** | Total classes: 1, Packages with the biggest number of classes:
- **Methods count** | Total methods: 2,Classes with the biggest number of methods: M1: 2
- **Cyclomatic complexity** | Average cyclomatic complexity: 1.0  
Methods with the highest cyclomatic complexity: M1::setx2: 1, M1::setx1: 1
- **LCOM**  
Average LCOM 1: 1,  
Classes with the highest LCOM 1:M1: 1, Packages with the highest average LCOM 1:  
Average LCOM 2: 0.5  
Classes with the highest LCOM 2: M1: 0.5, Packages with the highest average LCOM 2:  
Average LCOM 3: 1.0  
Classes with the highest LCOM 3: M1: 1.0, Packages with the highest average LCOM 3:  
Average LCOM 4: 2  
Classes with the highest LCOM 4: M1: 2, Packages with the highest average LCOM 4:

# Kod źródłowy klasy M2

```
package mM2;
import java.lang.Math;
import mM1.M1;
public class M2
{
 private double a, b, c;
 private M1 pierw = new M1();
 public M2(double a_, double b_, double c_)
 { a=a_; b=b_; c=c_; }
 public int a1 ()
 { int B;
 double pom=2*a, d=b*b-4*a*c;
 if (a==0) B=0;
 else
 if (d<0) B=1;
 else
 { B=2;
 d=Math.sqrt(d);
 pierw.setx1((-b-d)/pom);
 pierw.setx2((-b+d)/pom);
 }
 return B;
 }
}
```

2018-11-11

# SMC - Metryki kodu źródłowego klasy M2

- **LOC** | Total LOC: 27, Classes LOC: M2: 27, Packages LOC:
- **Lines with imports** | Total imports:2, Classes imports: M2: 2, Packages imports:
- **Blank lines** | Total blank lines:1, Classes blank lines:M2:1, Packages blank lines:
- **Classes count** | Total classes: 1, Packages with the biggest number of classes:
- **Methods count** | Total methods: 1, Classes with the biggest number of methods: M2: 1
- **Cyclomatic complexity** | Average cyclomatic complexity: 3.0  
Methods with the highest cyclomatic complexity: M2::a1: 3
- **LCOM**
- Average LCOM 1: 0  
Classes with the highest LCOM 1: M2: 0,  
Packages with the highest average LCOM 1:
- Average LCOM 2: 0.0  
Classes with the highest LCOM 2:M2:0.0,  
Packages with the highest average LCOM 2:
- Average LCOM 3: 0.0  
Classes with the highest LCOM 3:M2:0.0,  
Packages with the highest average LCOM 3:
- Average LCOM 4: 1  
Classes with the highest LCOM 4:M2: 1,  
Packages with the highest average LCOM 4:

# Kod źródłowy klasy M3

```
package mM3;
import mM2.M2;

public class M3
{
 M2 B;
 public M3 (M2 B_)
 { B=B_; }

 public void a2()
 {
 int b=B.a1();
 if (b<1)
 System.out.println("Brak równania kwadratowego\n");
 else
 if (b==1)
 System.out.println("Brak pierwiastków rzeczywistych\n");
 else
 System.out.println("Równanie ma pierwiastki rzeczywiste\n");
 }
}
```

# SMC - Metryki kodu źródłowego klasy M3

- **LOC** | Total LOC: 21, Classes LOC: M3: 21, Packages LOC:
- **Lines with imports** | Total imports: 1, Classes imports: M3: 1, Packages imports:
- **Blank lines** | Total blank lines: 2, Classes blank lines: M3: 2, Packages blank lines:
- **Classes count** | Total classes: 1, Packages with the biggest number of classes:
- **Methods count** | Total methods: 1, Classes with the biggest number of methods: M3: 1
- **Cyclomatic complexity** | Average cyclomatic complexity: 3.0  
Methods with the highest cyclomatic complexity: M3::a2: 3
- **LCOM**
- Average LCOM 1: 0,  
Classes with the highest LCOM 1: M3: 0,  
Packages with the highest average LCOM 1:
- Average LCOM 2: 0.0  
Classes with the highest LCOM 2: M3: 0.0,  
Packages with the highest average LCOM 2:
- Average LCOM 3: 0.0  
Classes with the highest LCOM 3: M3: 0.0,  
Packages with the highest average LCOM 3:
- Average LCOM 4: 1  
Classes with the highest LCOM 4: M3: 1,  
Packages with the highest average LCOM 4:

# Kod źródłowy klasy Program1

```
package pprogram1;

import mM3.M3;
import mM2.M2;
public class Program1
{
 public static void main(String arg[])
 {
 M2 rownanie = new M2(1,2,3);
 M3 info= new M3(rownanie);
 info.a2();
 }
}
```



# SMC - Metryki kodu źródłowego klasy Program1

- **LOC** | Total LOC: 15, Classes LOC: Program1: 15, Packages LOC:
- **Lines with imports** | Total imports: 2, Classes imports:Program1: 2, Packages imports:
- **Blank lines** | Total blank lines:3 | Classes blank lines:Program1: 3, Packages blank lines:
- **Classes count** | Total classes: 1,Packages with the biggest number of classes:
- **Methods count**| Total methods: 1,Classes with the biggest number of methods:Program1: 1
- **Cyclomatic complexity** | Average cyclomatic complexity: 1.0,
  - Methods with the highest cyclomatic complexity: Program1::main: 1
- **LCOM**
- Average LCOM 1: 0,
  - Classes with the highest LCOM 1:Program1: 0,
  - Packages with the highest average LCOM 1:
- Average LCOM 2: 0.0
  - Classes with the highest LCOM 2:Program1:0.0,
  - Packages with the highest average LCOM 2:
- Average LCOM 3: 0.0
  - Classes with the highest LCOM 3:Program1:0.0,
  - Packages with the highest average LCOM 3:
- Average LCOM 4: 1
  - Classes with the highest LCOM 4:Program1: 1,
  - Packages with the highest average LCOM 4: