Wykład 7
Złożoność strukturalna programów, metryki
Metryki międzymodułowe

7.1. Podstawowe definicje

Struktura programu to:

- przedstawienie programu na różnych poziomach abstrakcji rozumiane jako odseparowanie danych od bezpośredniej reprezentacji – wynika to z sekwencyjnego przebiegu procesu myślowego i jednocześnie z możliwości wyobrażenia sobie zaledwie ograniczonej liczby pojęć
- podział programu na podsystemy, moduły, klasy, funkcje.

Problem złożoności struktury programu odgrywa kluczową rolę w:

- testowaniu programu, czyli osiąganiu jak największej jego niezawodności
- rozwijaniu programu wynikającego z możliwości zrozumienia programu i stopnia osiągniętej abstrakcji w dziedzinie danych i operacji
- pielęgnacji programu
- wielokrotnemu zastosowaniu elementów programu (biblioteki, moduły).

co opowiada następującym *atrybutom zewnętrznym oprogramowania*, wynikających *ze złożoności psychologicznej:*

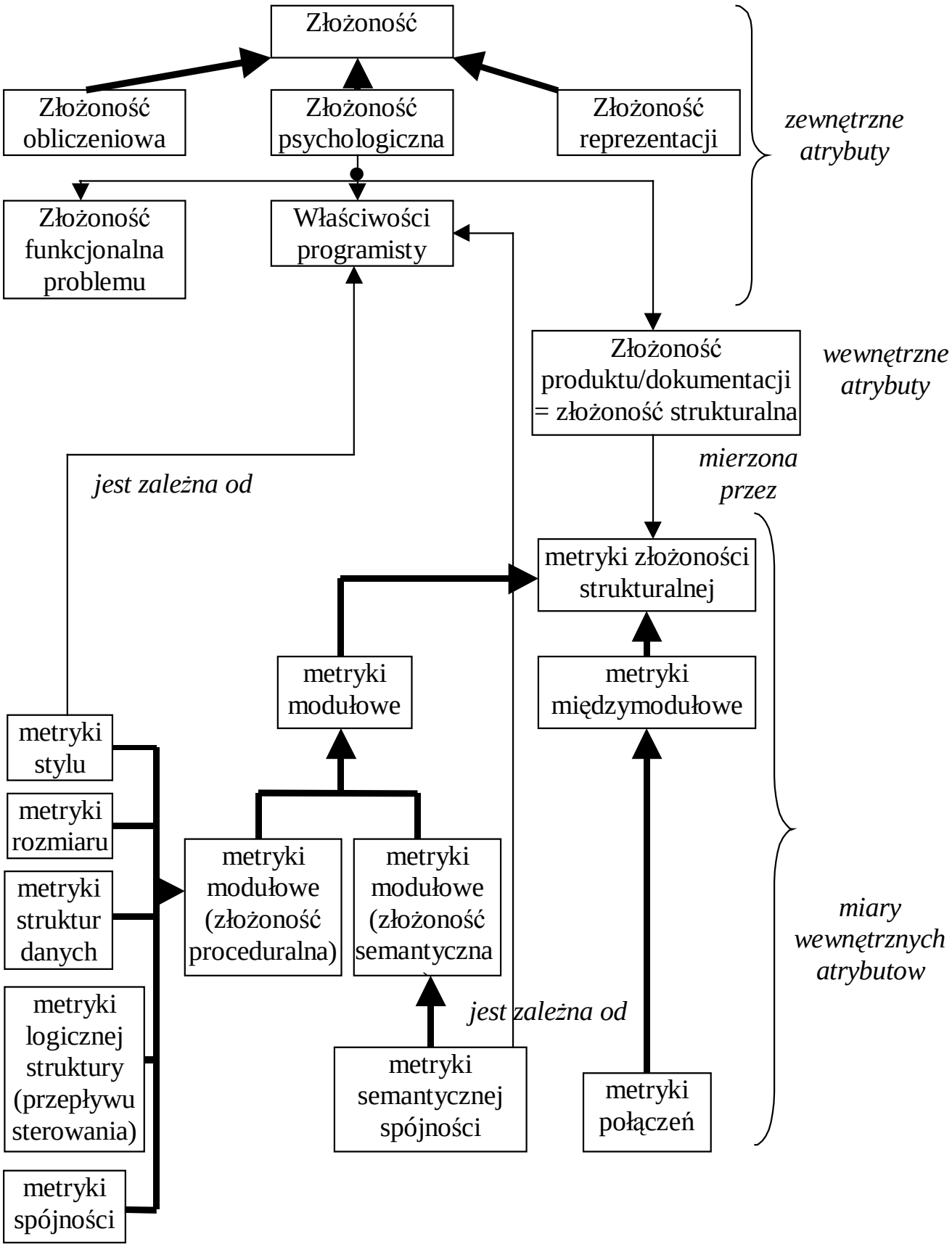
- testowalności, a więc również niezawodności,
- stopnia osiągniętej abstrakcji
- zrozumiałości programu
- stopnia pielęgnacji
- wieloużywalności.

Atrybuty zewnętrzne zależą od *atrybutów wewnętrznych oprogramowania* i są wyrażane w postaci obiektywnych miar tych atrybutów.

Wyróżnia się następujące *atrybuty wewnętrzne* oprogramowania:

- *charakterystyki międzymodułowe* czyli wszelkie związki między modułami (przekazywanie sterowania i parametrów, wspólne korzystanie z pól danych, przy projektowaniu jednego modułu uwzględnia się właściwości innego modułu
- *charakterystyki 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.

Te *atrybuty wewnętrzne* są wyrażane za pomocą tzw. *metryk,* czyli prostych wyrażeń, wiążących pewne elementy programu (projektu, kodu źródłowego itp.). Wybór elementów wynika z ich odpowiedzialności za dany atrybut wewnętrzny, a wyrażenie określa wartościowanie atrybutu.



Rys. 1. Klasyfikacja pojęć złożoności oprogramowania i podanie związku między zewnętrznymi atrybutami oprogramowania i miarami wewnętrznych atrybutów [12]

7.2. 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.” [15]

Formalna definicja elementów środowiska pomiarowego metryk [12], 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 jedyny 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 ≡ Q tzn. spełniające te same funkcje, lecz o różnej złożoności.
5. Złożoność programów P i Q połączonych jest niewiększa 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.

7.3. Podstawowe metryki

Poszczególne metryki złożoności strukturalnej mogą być wyznaczane w dowolnym stadium rozwoju oprogramowania, jednak to rzutuje na wybór elementów produktu i rodzaj wyrażenia. Opis dotyczy metryk operujących na elementach programów napisanych w języku C++.

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

7.4. Metryki złożoności międzymodułowej

<u>*Osłabienie powiązań miedzy-modułowych*</u> prowadzi do zmniejszenia oddziaływań między modułami oraz poprawy struktury oprogramowania.

Zbiór połączeń R modułu X z pozostałymi modułami jest sumą relacji R_i zachodzących między zbiorem elementów *łączących wyjściowych x* modułu *X* i elementów *łączących wejściowych y* z modułów $Y_1,...,Y_N$.

$$\{<x, y> : <x, y> \in R_i\} \subseteq X \times Y_i \text{ dla } i \in \{1,\ldots,N\}$$

$$R = \cup_{i \in N} R_i \subseteq \cup_{i \in N} (X \times Y_i)$$

Elementami łączącymi wyjściowymi są:
1) funkcja wywołująca funkcję z innego modułu
2) parametr lub wynik funkcji typu zdefiniowanego w innym module przekazywany przez wartość, wskaźnik i referencję
3) używanie zmiennych automatycznych typu zdefiniowanego poza modułem
4) używanie zmiennych automatycznych typu wskaźnikowego zdefiniowanego poza modułem
5) używanie zmiennych typu **extern** wskaźnikowego zdefiniowanego poza modułem
6) wszystkie deklaracje **extern**
7) każda informacja z poza modułu potrzebna do zdefiniowania ciała funkcji (np. obsługa błędów), definicji typu strukturalnego, definicji dowolnej zmiennej

Elementami łączącymi wejściowymi są:
1) funkcja wywoływana przez funkcję z innego modułu
2) typ zdefiniowany w module i zastosowany w innym module do definicji parametru przekazywanego przez wartość, wskaźnik i referencję
3) typ zdefiniowany w module i zastosowany w innym module do definicji zmiennej automatycznej
4) typ zdefiniowany w module i używany w innym module do definicji zmiennej automatyczna typu wskaźnikowego
5) typ zdefiniowany i używany do definicji zmiennej **extern** przez inne moduły
6) typ zdefiniowany i używany do definicji zmiennej **extern** typu wskaźnikowego przez inne moduły
7) informacja zawarta w module potrzebna w innych modułach do dowolnej definicji

Przyklady metryk:

Fan-out

Metryka *Fan-out* wyznaczająca liczbę połączeń *elementów wyjściowych* jednego modułu z *elementami wejściowymi* innych modułów. Uwzględnia się tylko jedno dowolne połączenie wyjściowe-wejściowe z każdym z modułów. Relacja R opisująca tę metrykę zawiera *tylko jedną dowolną* parę jako połączenie z każdym z modułów.

Fan-in

Metryka *Fan-in* wyznaczająca liczbę połączeń *elementów wejściowych* jednego modułu z *elementami wyjściowymi* innych modułów. Uwzględnia się tylko jedno dowolne wejściowo-wyjściowe połączenie z każdym z modułów. Relacja R opisująca tę metrykę zawiera *tylko jedną dowolną parę* jako połączenie z każdym z modułów.

RS

Metryka *RS* jest *sumą zbiorów* wszystkich elementów łączących dany moduł ze wszystkimi pozostałymi. Uwzględnia się zbiór wszystkich elementów wyjściowych własnych modułu oraz zbiory elementów łączących wejściowych z innych modułów, z którymi jest połączony dany moduł. Każdy z elementów wejściowych i wyjściowych wystąpi tylko raz w zbiorze RS.

$$RS = M \cup (\cup_{i \in N} Y_i),$$

gdzie M jest zbiorem elementów wyjściowych połączonych danego modułu oraz Y_i są zbiorami elementów wejściowych połączonych w N modułach

RFC

Metryka *RFC* jest *mocą zbioru RS*, czyli RFC=|RS|

Przykład 7.1.

Przykład wyznaczenia poszczególnych metryk dla modułu A

Moduł A zawiera elementy łączące wyjściowe: A1, A2 ,A3 ,A4. Moduł B 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 B2, C1

A3 łączy się D1

A4 łączy się C1

RS={A1,A2,A3,A4} ∪ {B1,B2} ∪ {D1} ∪ {C1} = {A1,A2,A3,A4,B1,B2,D1,C1}

RFC= |RS|=8

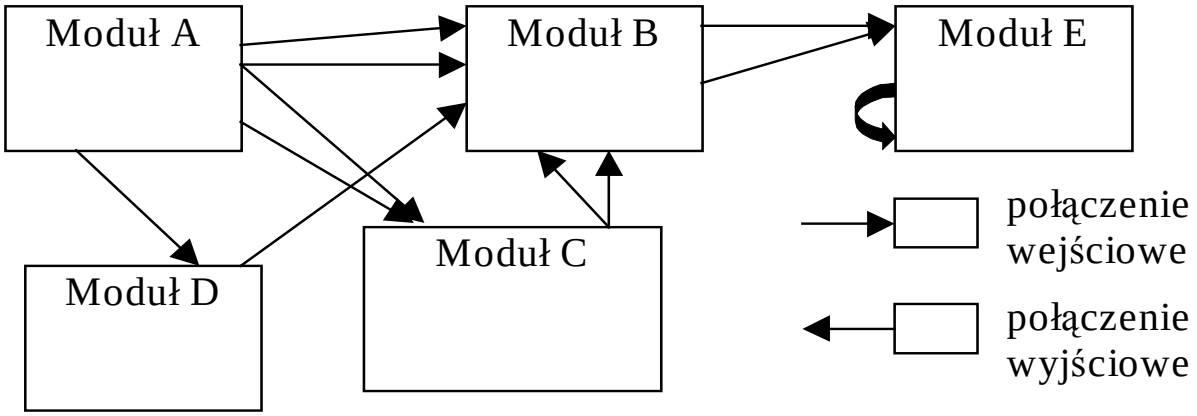
Fan-out= |{<A1,B1>, <A2,C1>, <A3,D1>}| = 3 *//dowolny element wejściowy*

Fan-in=|{}|=0

R={<A1,B1>, <A2,B2>, <A2,C1>,<A3,D1>,<A4,C1>}

|R|=5

Przykład 7.2



Rys. 2. Przykłady połączeń między klasami

Przykłady metryk dla modułów z rys. 2.

	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

Przykład 7.3

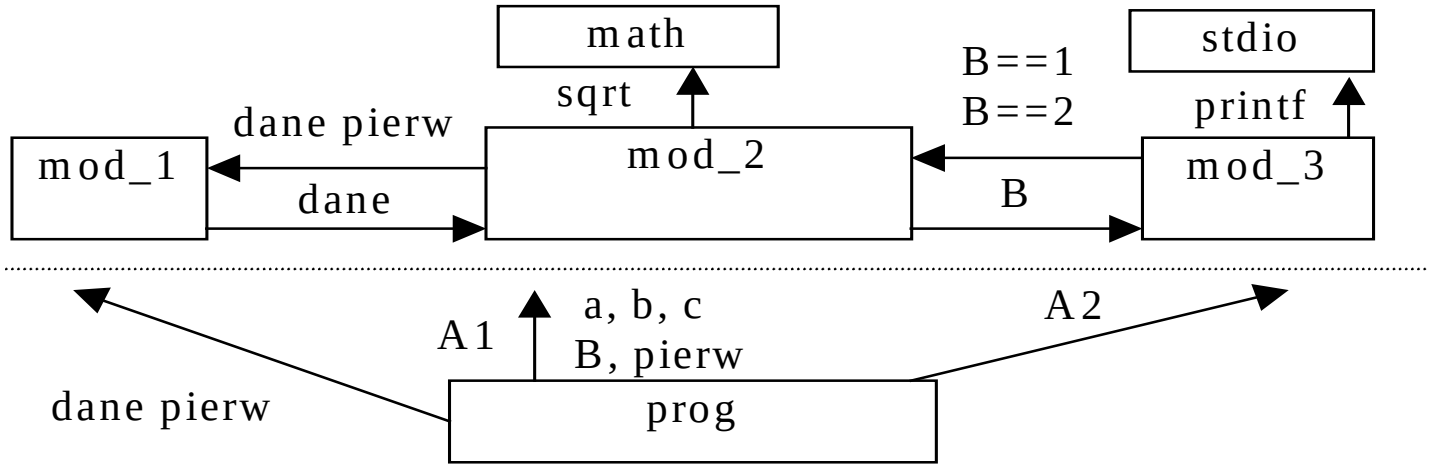
```
//moduł mod_1
#ifndef DANE                    //mod_1.h
#define DANE
struct dane
{ float x1,x2;};
#endif
//koniec mod_1

//moduł mod_2
#include "mod_1.h"              //mod_2.h
void A1 (float a,float b,float c,int& B, dane& pierw);

#include <math.h>               //mod_2.cpp
#include "mod_2.h"
void A1 (float a,float b,float c, int& B, dane& pierw)
 {float pom=2*a, d=b*b-4*a*c;
  if (d<0) B=1;
  else
  {B=2; d=sqrt(d);
   pierw.x1=(-b-d)/pom; pierw.x2=(-b+pom)/pom;
}}//koniec mod_2
```

```
//moduł mod_3
void A2(int B);              //plik mod_3.h
#include <stdio.h>           //plik mod_3.cpp
#include "mod_3.h"
void A2(int B)
 { if (B<1)
       printf("Brak równania kwadratowego\n");
   else
    if (B==1)
       printf("Brak pierwiastków rzeczywistych\n");
    else
       printf("Rownanie ma pierwiastki rzeczywiste\n");}
//koniec mod_3

//#include "mod_1.h"
#include "mod_2.h"
#include "mod_3.h"
void main()
 {float a=1,b=2,c=1;
  dane pierw;
  int B;
  if (a==0) B=0;
  else A1(a,b,c,B,pierw);
  A2(B);}
```



	mod_1	mod_2	mod_3	stdio	math
Fan-out	1	3	2	-	-
Fan-in	1	2	1	1	1
RFC	2	6	4	-	-
|R|	1	3	2	-	-