Zarządzanie jakością oprogramowania

Autor
Zofia Kruczkiewicz
Programowanie i wdrażanie systemów informatycznych

Główne zagadnienia

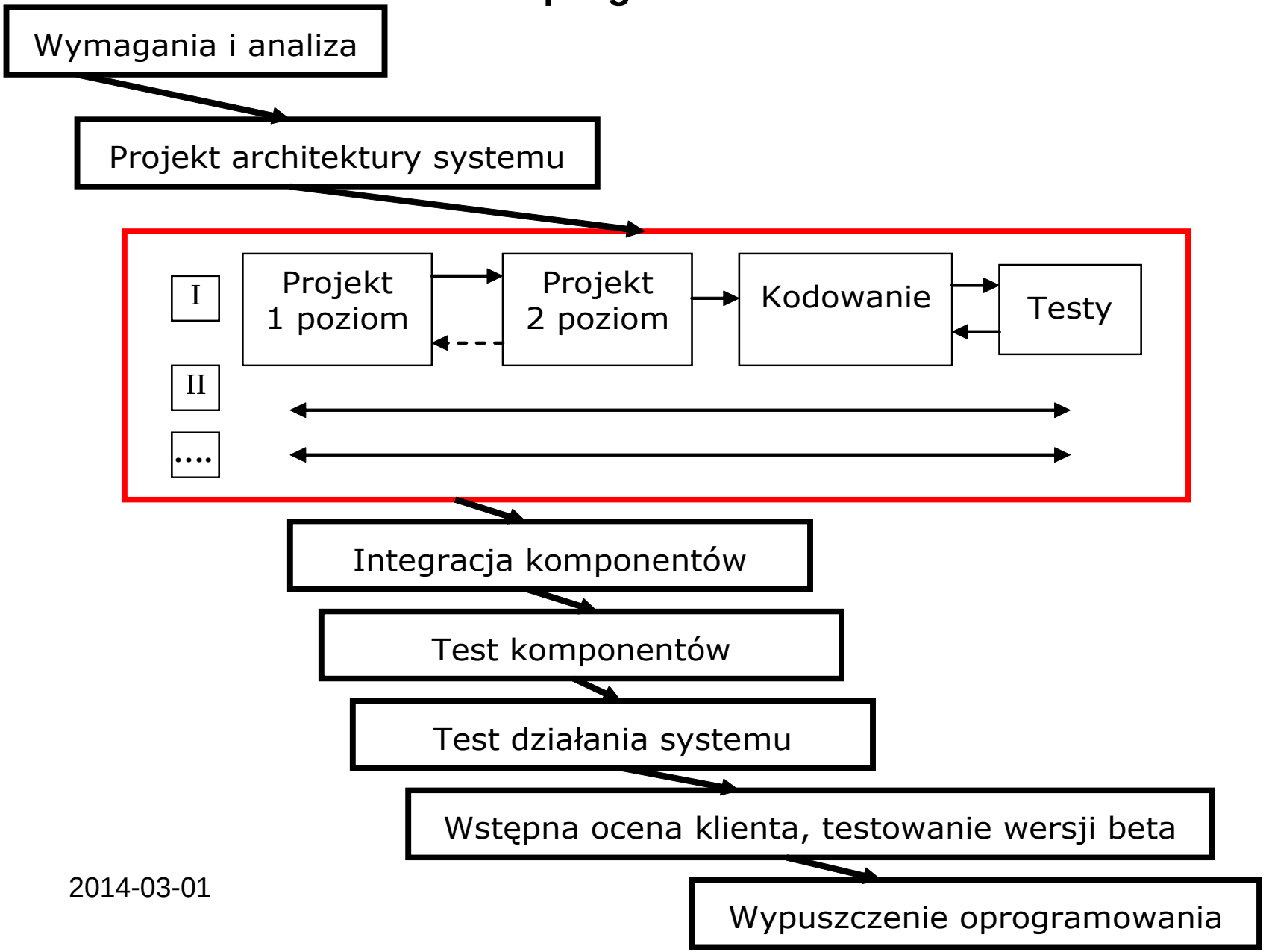
1. Modele procesu produkcji oprogramowania
2. Zapewnianie jakości i standardy [1]
3. Planowanie jakości [1]
4. Kontrolowanie jakości [1]
5. Miernictwo oprogramowania i miary [1]
6. Zalecenia dla projektów obiektowych

[1] Ian Sommerville „Inżynieria oprogramowania”
[2] Stephen H. Kan „Metryki i modele w inżynierii jakości oprogramowania”

Główne zagadnienia

1. **Modele procesu produkcji oprogramowania**

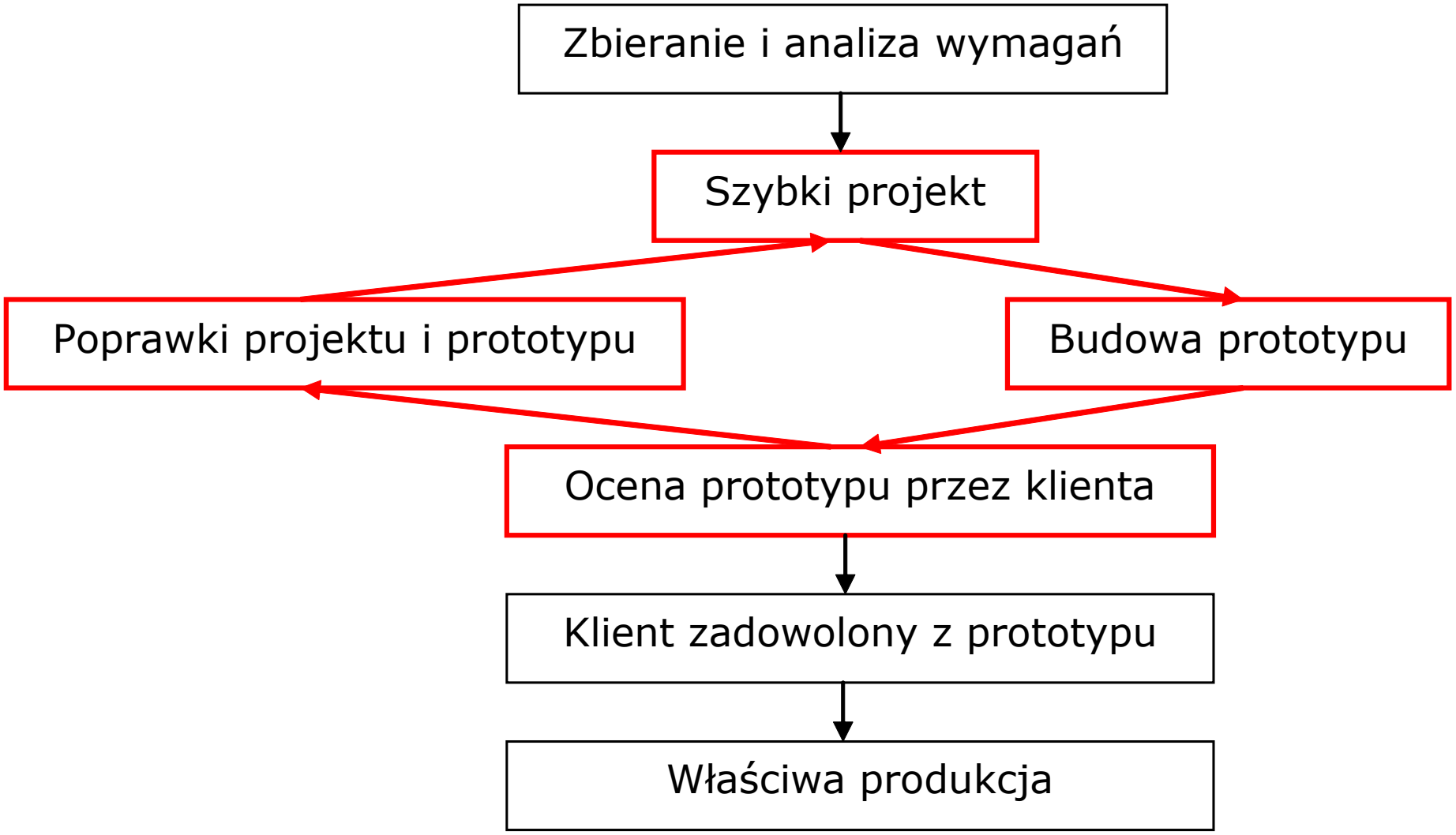
1. Proces - model kaskadowy; produkt – oprogramowanie obiektowe i nieobiektowe



Właściwości modelu kaskadowego – „dziel i rządź”

- Schemat działania: **Wejście –Zadanie – Sprawdzenie – Wyjście** (Entry-Task-Validation_Exit)
- Ułatwienie prowadzenia projektu
- Podstawowe podejście przy tworzeniu oprogramowania metodą strukturalną
- Doświadczenia ostatnich dekad potwierdziły jego przydatność np. podczas tworzenia systemów operacyjnych
- **Brak kontaktów z klientem**
- **Brak odporności na zmianę wymagań klienta**
- **Możliwość strat, ponieważ ostateczna ocena następuje pod koniec cyklu życia oprogramowania**

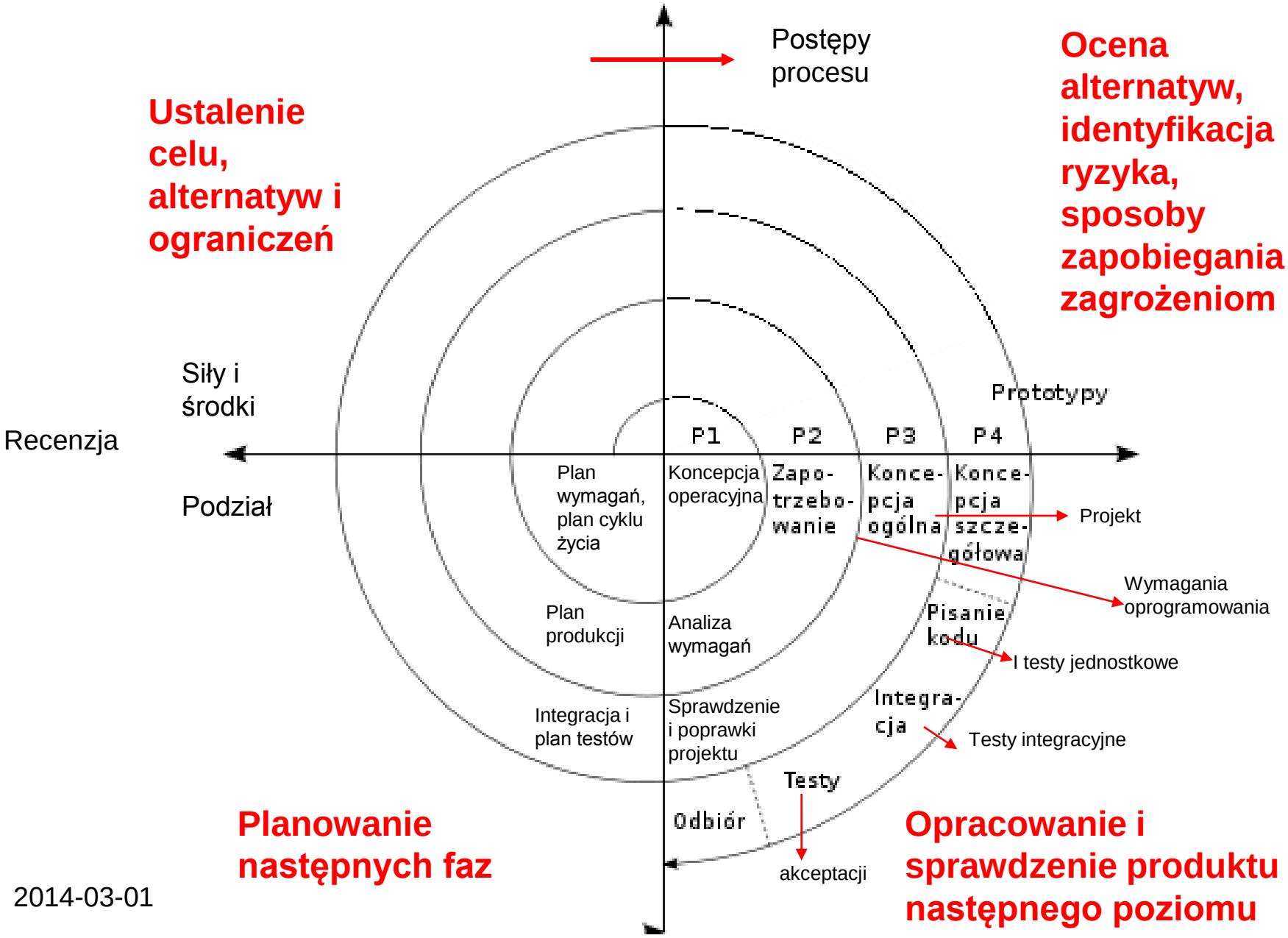
2. Proces - model prototypowy
produkt – oprogramowanie obiektowe i nieobiektowe



Właściwości tworzenia prototypów

- Stosuje się w przypadku braku jasno zdefiniowanych wymagań
- Konieczność zapewnienia szybkości i elastyczności w projektowaniu i budowaniu prototypów
  - Wielokrotne używanie kodu
  - Języki specyfikacji (język wymagań wejścia/ wyjścia – Input/Output Requirements Language (IORL))
- Sprawdzają się w pracy nad prostymi zagadnieniami na poziomie subsystemowym
- Zastosowanie metody *„podziału na okresy”* (time boxing) - brak jasnych kryteriów dotyczących przerwania poprawiania prototypu w kolejnej iteracji
- *Metoda Szybkiego odrzucania prototypu* – po odrzuceniu buduje się od podstaw nowy prototyp (metoda stosowana w oprogramowaniu o wysokim stopniu ryzyka)
- *Metoda Ewolucji prototypu* – poprawa przez kolejne ulepszanie
- **Wysokie koszty tworzenia oprogramowania**
- **Możliwa dezorientacja klienta podczas oceny prototypu**

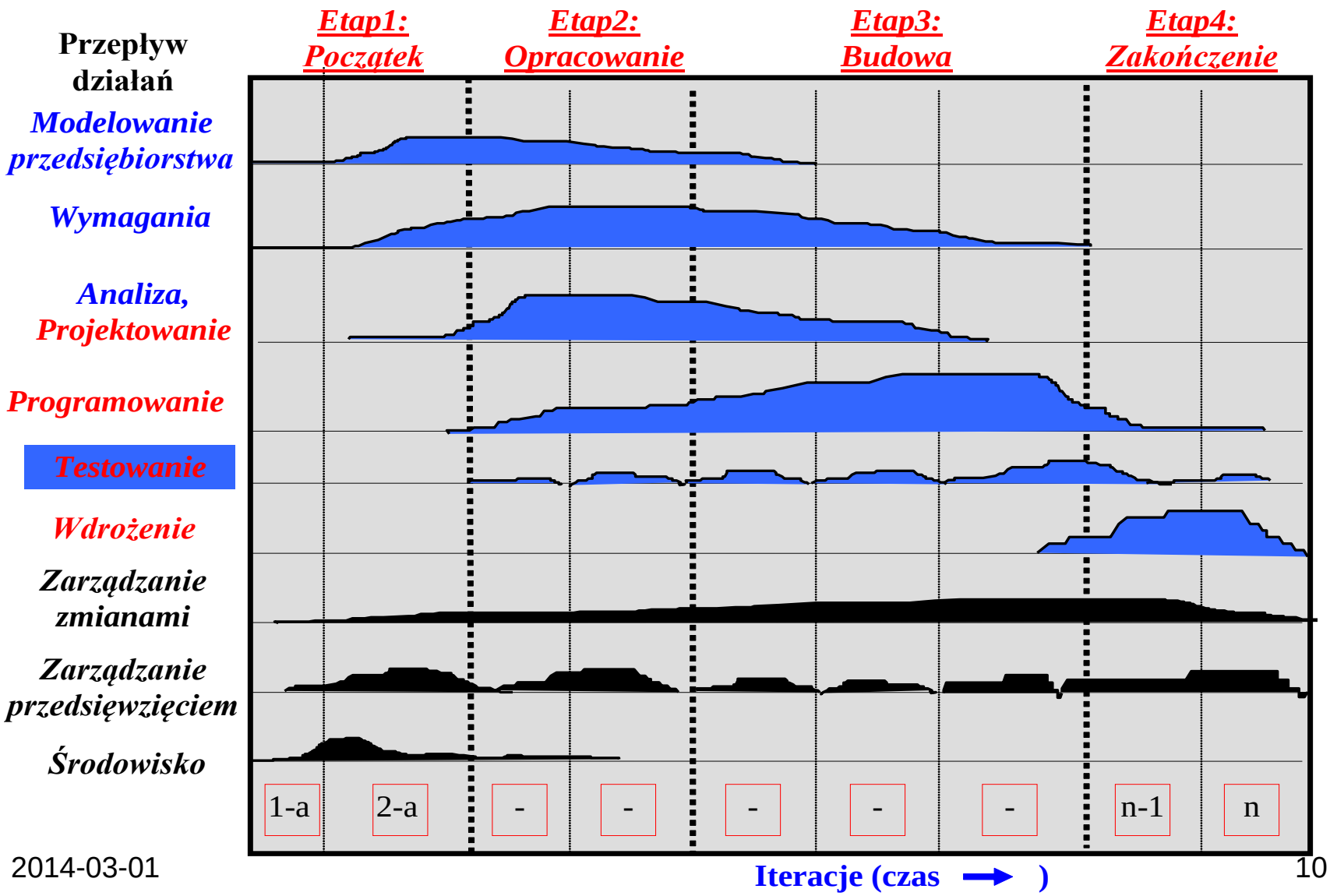
3. Proces - model spiralny; produkt – oprogramowanie obiektowe



Właściwości tworzenia modelu spiralnego

- Posiada zalety modeli kaskadowego i prototypu, natomiast analiza ryzyka pozwala unikać wad tych modeli
- Stosowana do dużych projektów
- Wieloużywalność istniejącego oprogramowania, odporność na zmiany wymagań
- Zastosowanie celów jakości do produkcji oprogramowania
- Systematyczne testowanie podczas całego cyklu życia oprogramowania
- Eliminowanie błędów i niewłaściwych alternatyw rozwoju we wczesnej fazie rozwoju oprogramowania
- Identyczny sposób budowania modelu do produkcji i jego ulepszania
- Solidny funadament do integrowania oprogramowania ze sprzętem
- **Trudność z wywiązania się z warunków kontraktu, lepsza w przypadku budowania oprogramowania do sprzedaży**
- **Duży wpływ analizy ryzyka na przebieg projektowania - błędy w analizie mogą negatywnie wpłynąć na wynik produkcji oprogramowania**
- **Potrzeba opracowania kolejnych kroków przy zachowaniu spójności projektu - może ją stosować jedynie zespół profesjonalistów**

4. Proces - metody iteracyjno- rozwojowe; produkt - oprogramowanie obiektowe



Odmiany metod iteracyjno-rozwojowych

- **Programowanie ekstremalne** (*eXtreme Programming*, *XP*) - wydajne tworzenie małych i średnich "projektów wysokiego ryzyka” oparte na synergii stosowania rozmaitych praktyk zapewniające eliminację wad i wykorzystania zalet tych praktyk.
- **Programowanie zwinne** (*Agile software development*) –

  Pojęcie *zwinnego programowania* zostało zaproponowane w 2001 w Agile Manifesto:
  - osiągnięcie satysfakcji klienta poprzez szybkość wytwarzania oprogramowania,
  - działające oprogramowanie jest dostarczane okresowo (raczej tygodniowo niż miesięcznie),
  - podstawową miarą postępu jest działające oprogramowanie,
  - późne zmiany w specyfikacji nie mają destrukcyjnego wpływu na proces wytwarzania oprogramowania,
  - bliska, dzienna współpraca pomiędzy biznesem a developerem,
  - bezpośredni kontakt, jako najlepsza forma komunikacji w zespole i poza nim,
  - ciągła uwaga nastawiona na aspekty techniczne oraz dobry projekt (design),
  - prostota,
  - samozarządzalność zespołów,
  - regularna adaptacja do zmieniających się wymagań.

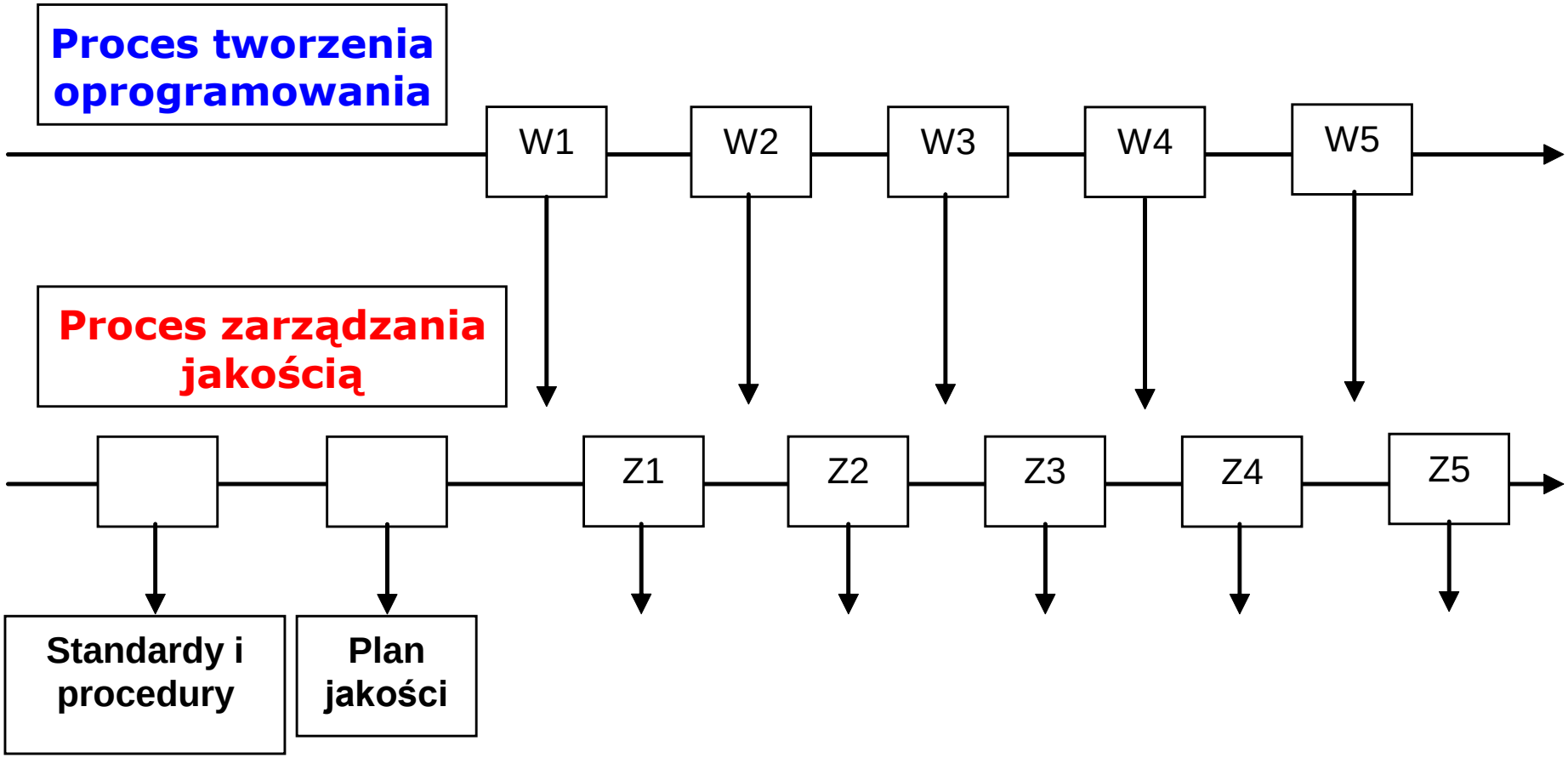
Własności metod iteracyjno - rozwojowych

- Częste kontakty z klientem
- Możliwy niepełny zbiór wymagań
- Odporność na zmiany wymagań
- Wczesne wykorzystanie przez klienta fragmentów systemu
- Utrzymanie terminu – możliwość elastycznego reagowania na opóźnienia realizacji jednej części i przyspieszenie prac nad inną/innymi częściami
- **Dodatkowy koszt związany z niezależną realizacją fragmentów systemu**
- **Potencjalne trudności z wycinaniem podzbioru funkcji w pełni niezależnych**
- **Konieczność implementacji szkieletów (interfejs zgodny z docelowym systemem) – dodatkowy nakład pracy (koszt), ryzyko niewykrycia błędów w fazie testowania**

Główne zagadnienia

1. Modele procesu produkcji oprogramowania
2. **Zapewnianie jakości i standardy** [1]

Zarządzanie jakością i tworzenie oprogramowania



Zarządzanie jakością powinno być oddzielone od zarządzania przedsięwzięciem.

Zarządzanie jakością oprogramowania można podzielić na **trzy zasadnicze czynności**:

1) Zapewnianie jakości
2) Planowanie jakości
3) Kontrola jakości

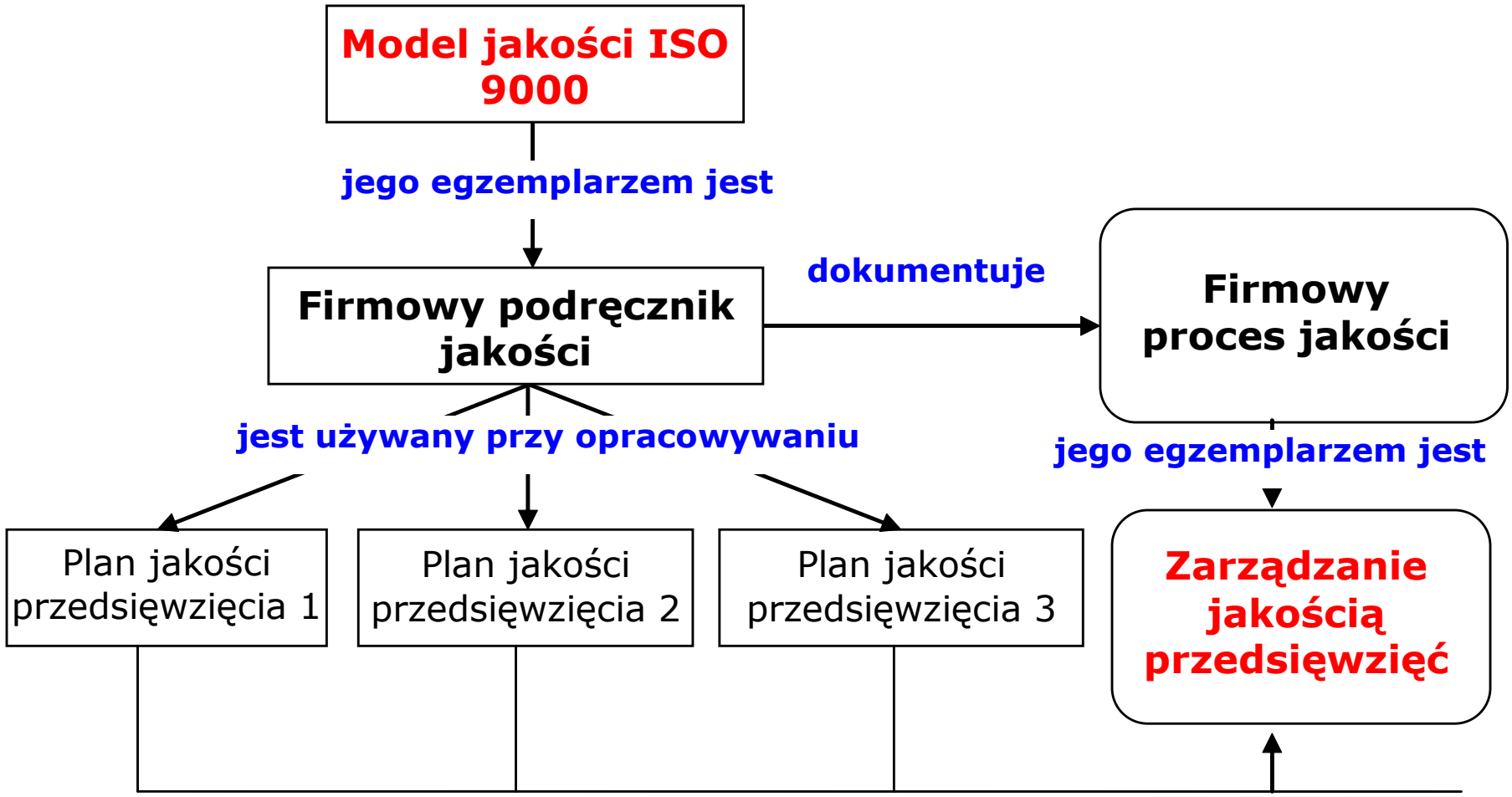
Zarządzanie jakością stanowi niezależne sprawdzenie procesu tworzenia oprogramowania – sprawdza się, czy wyprodukowane oprogramowanie jest zgodne ze:

- standardami i celami firmy (usprawnianie procesu, jakość społeczności pracowników),
- wymaganiami klienta, który zamówił oprogramowanie.

Model zapewnienia jakości ISO 9001 (model procesu jakości niezależny od gałęzi przemysłu)

Zadanie kierownictwa	System jakości
Kontrola niezgodnych produktów	Kontrola projektów
Obsługa, składowanie, pakowanie i dostarczanie	Zakupy
Produkty dostarczane przez dostawców	Identyfikacja i śledzenie produktów
Kontrola procesu	Inspekcja i testowanie
Osprzęt do testowania i kontroli	Stan kontroli i testów
Przegląd umowy	Czynności poprawiające
Kontrola dokumentów	Rejestry jakości
Wewnętrzne kontrole jakości	Szkolenie
Realizacja usług	Metody statystyczne

ISO 9000 – system zarządzania jakością



ISO 9000-3 zawiera interpretację ISO-9000 dla tworzenia oprogramowania

Własności standardów oprogramowania

- Najlepiej dopasowane praktyki dla potrzeb firmy
- Stanowią szkielet jakości
- Standaryzacja działań

Typy standardów

- Standardy produktowe
- Standardy procesowe

Standardy te są ściśle powiązane: celem standardów procesowych często jest przestrzeganiem standardów produktowych.

Przykłady standardów produktowych i procesowych

Standardy produktowe	Standardy procesowe
Formularz przeglądu projektu	Przebieg przeglądu projektu
Struktura dokumentacji wymagań	Zgłaszanie dokumentów do zarządzania strukturą (CM-Configuration Management)
Format nagłówka procedury	Proces rozpowszechniania wersji
Styl programowania w Javie	Proces akceptacji planu przedsięwzięcia
Format planu przedsięwzięcia	Proces panowania nad zmianami
Formularz żądania zmiany	Proces rejestrowania testów

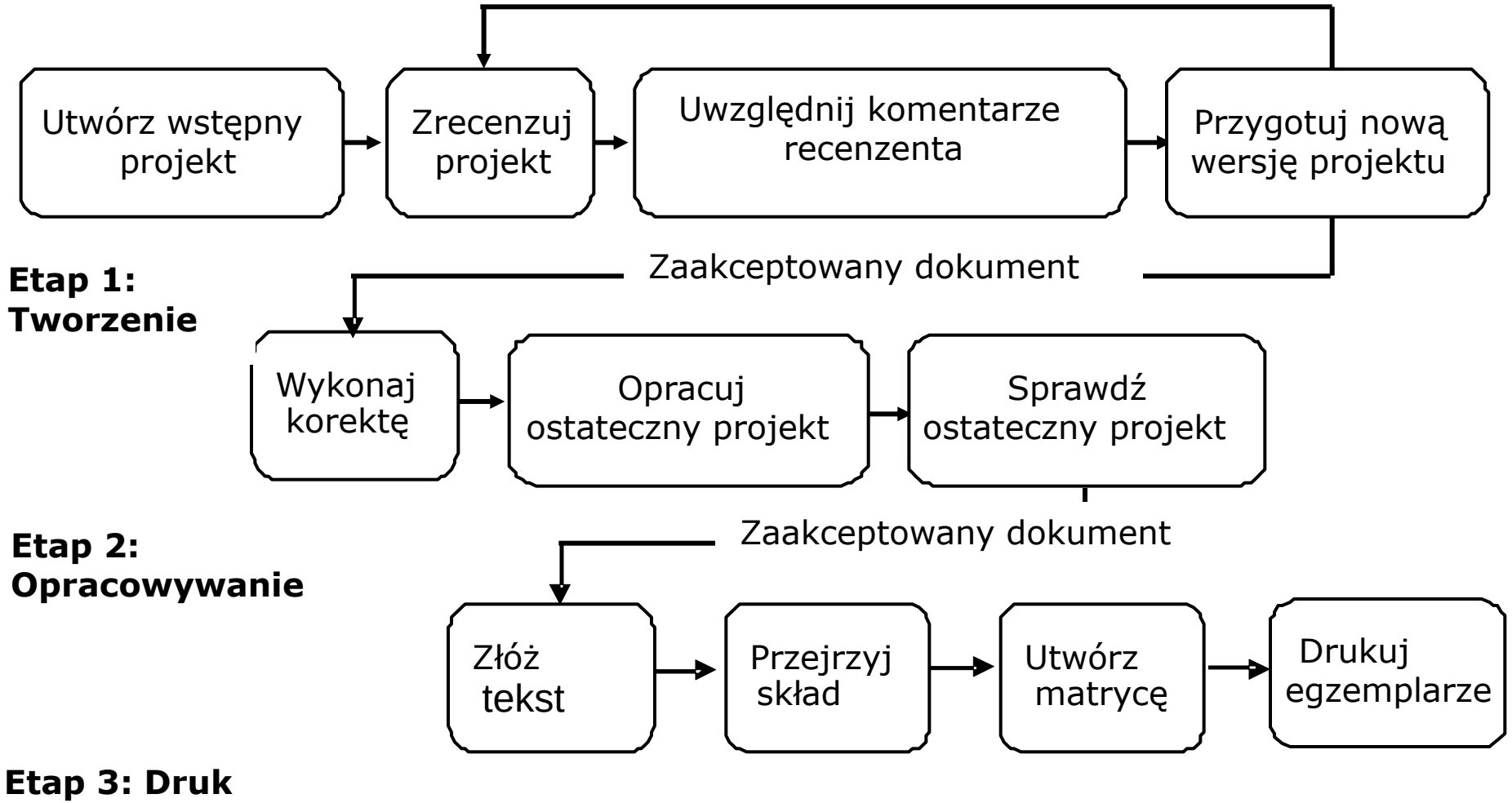
Metody opracowywania standardów

1) Opracowanie standardów również przez inżynierów oprogramowania
2) Dostosowywanie standardów do zmian technologii
3) Zautomatyzowanie procesów za pomocą narzędzi programistycznych wspiera standardy – ułatwia ich zachowanie lub wprowadzanie koniecznych zmian.

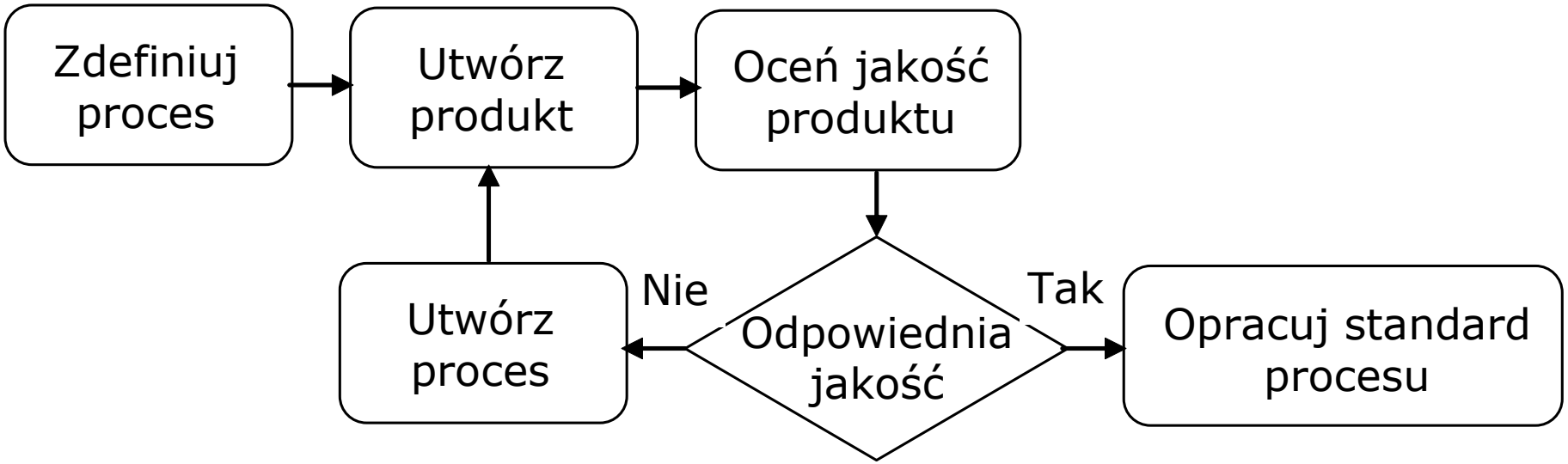
Standardy dokumentowania

1. Standardy procesu dokumentowania
2. Standardy dokumentów:
   1) Standardy identyfikacji dokumentów
   2) Standardy struktury dokumentów
   3) Standardy prezentacji dokumentów
   4) Standardy aktualizacji dokumentów
3. Standardy wymiany dokumentów – elektroniczne przesyłanie dokumentów

Przykład procesu dokumentowania - proces tworzenia dokumentacji z uwzględnieniem kontroli jakości



Jakość procesu i produktu



Główne zagadnienia

1. Modele procesu produkcji oprogramowania
2. Zapewnianie jakości i standardy [1]
3. **Planowanie jakości** [1]

Składowe planu jakości

1) **Określenie produktu** – opis produktu, rynek produktu i oczekiwania wobec produktu
2) **Plany dotyczące produkcji** – daty krytycznych wydań, plany dystrubucji i serwisu
3) **Opisy procesu** – procesy tworzenia i serwisowania produktu,
4) **Cele jakościowe** – atrybuty krytyczne produktu związane z produkcją
5) **Ryzyka i zarządzanie ryzykiem** – czynniki zapobiegające obniżeniu jakości produktu

Podstawowe definicje dotyczące oceny jakości produktu –struktura oprogramowania

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).

Atrybuty (*charakterystyki*) zewnętrzne produktu - wynikające ze złożoności struktury programu

1) jakość oprogramowania:

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

2) funkcjonalność

3) koszt.

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

Charakterystyki (atrybuty) wewnętrzne oprogramowania są wyrażane w postaci obiektywnych miar tych atrybutów czyli 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.

Wyróżnia się następujące **charakterystyki *wewnętrzne*** oprogramowania:

- ***charakterystyki (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)
- ***charakterystyki (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.

Główne zagadnienia

1. Modele procesu produkcji oprogramowania
2. Zapewnianie jakości i standardy [1]
3. Planowanie jakości [1]
4. **Kontrolowanie jakości** [1]

Podejścia do kontroli jakości

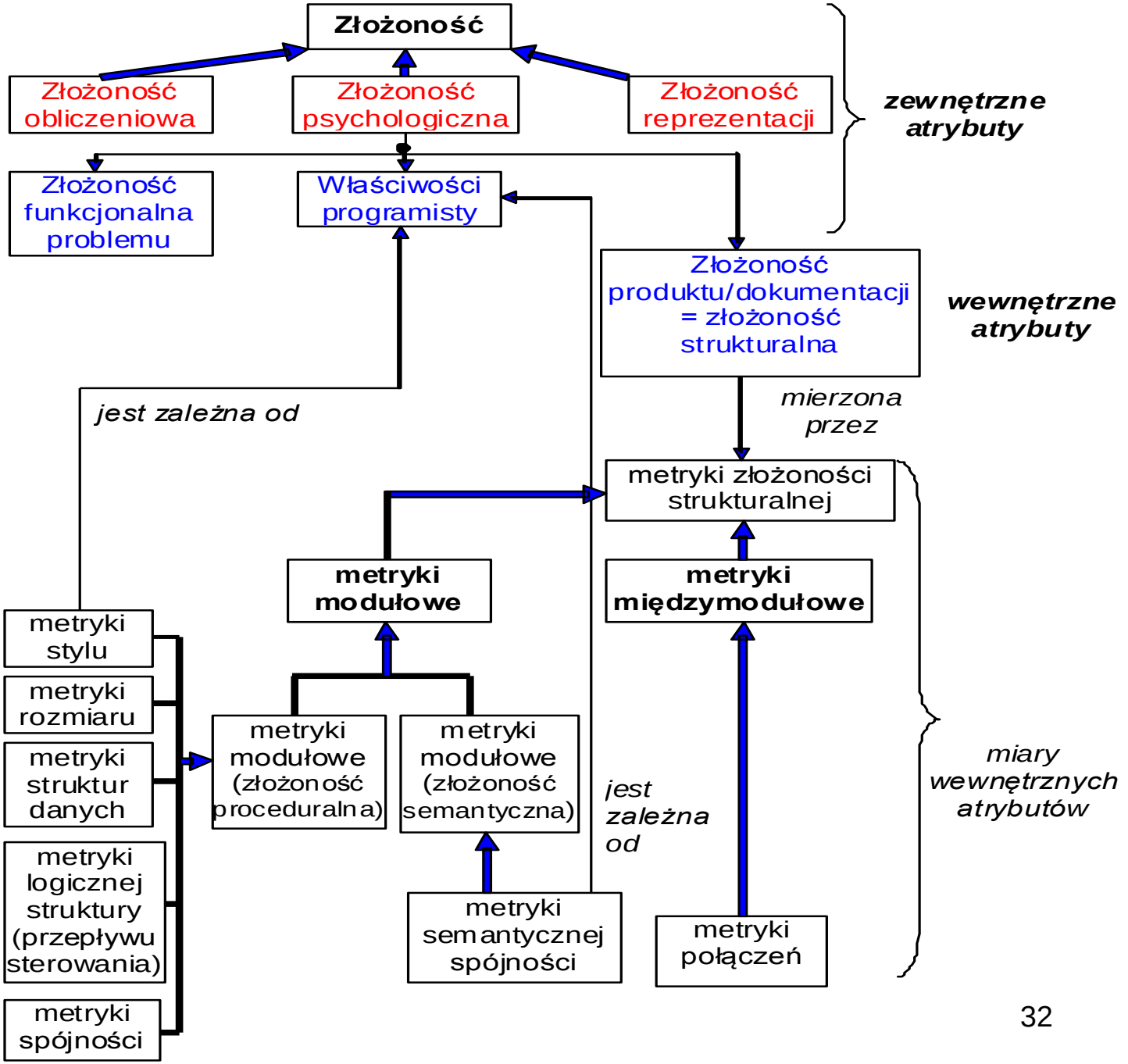
- Przeglądy jakości – badanie zgodności oprogramowania i dokumentacji ze standardami
- Automatyczna ocena oprogramowania za pomocą metryk, badanie zgodności ze standardami

Rodzaje przeglądów	Zasadniczy cel
Kontrola programu lub projektu	Wykrycie szczegółowych błędów w wymaganiach, projekcie lub kodzie na podstawie listy kontrolnej z potencjalnymi błędami
Przeglądy postępu	Przegląd produktu i procesu pod kątem kosztów, planów i harmonogramów
Przegląd jakości	Analiza techniczna komponentów produktu lub dokumentacji w celu wykrycia niezgodności między specyfikacją, projektem, kodem i dokumentacją komponentu i stopnia przestrzegania standardów jakości

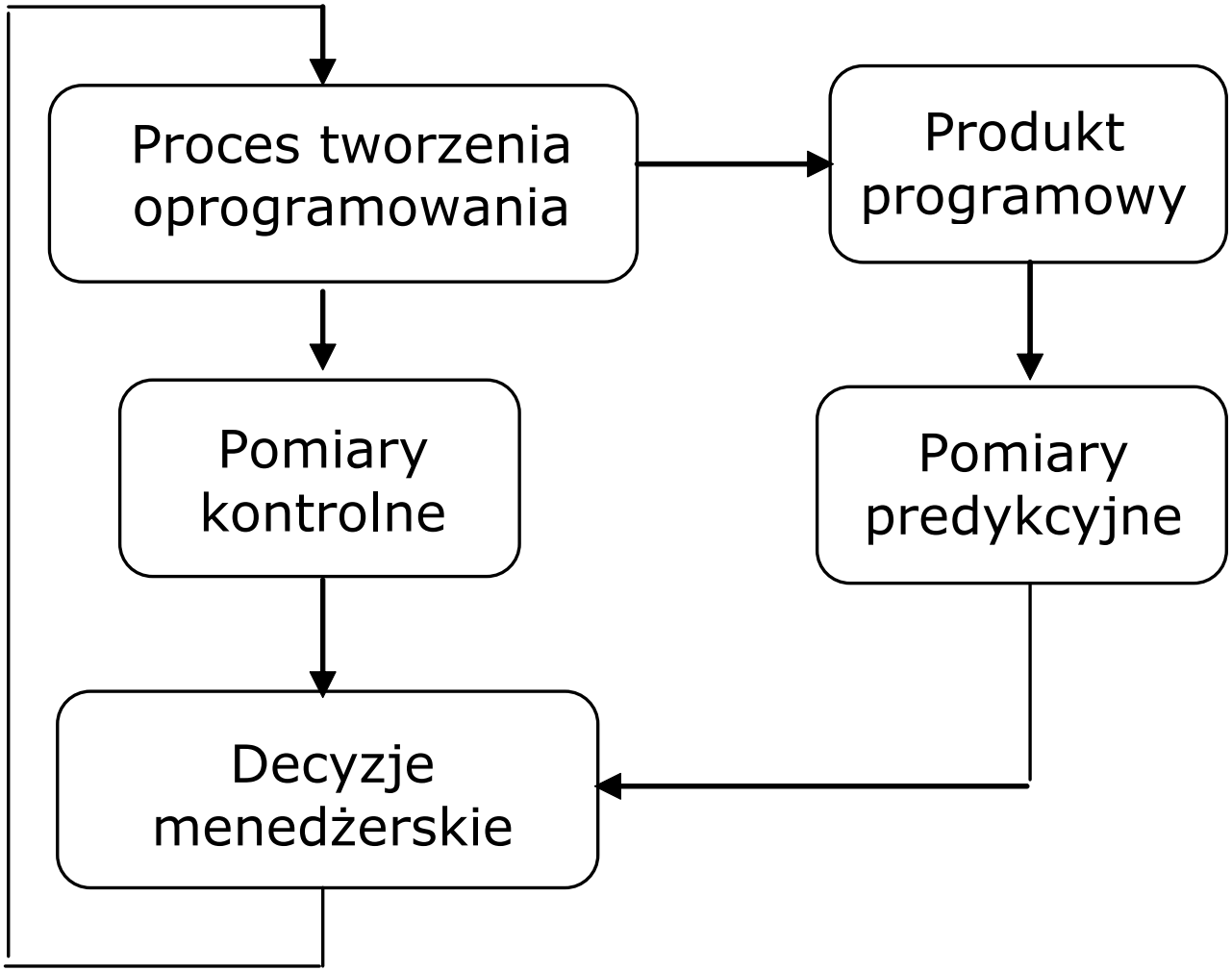
Główne zagadnienia

1. Modele procesu produkcji oprogramowania
2. Zapewnianie jakości i standardy [1]
3. Planowanie jakości [1]
4. Kontrolowanie jakości [1]
5. **Miernictwo oprogramowania i miary**

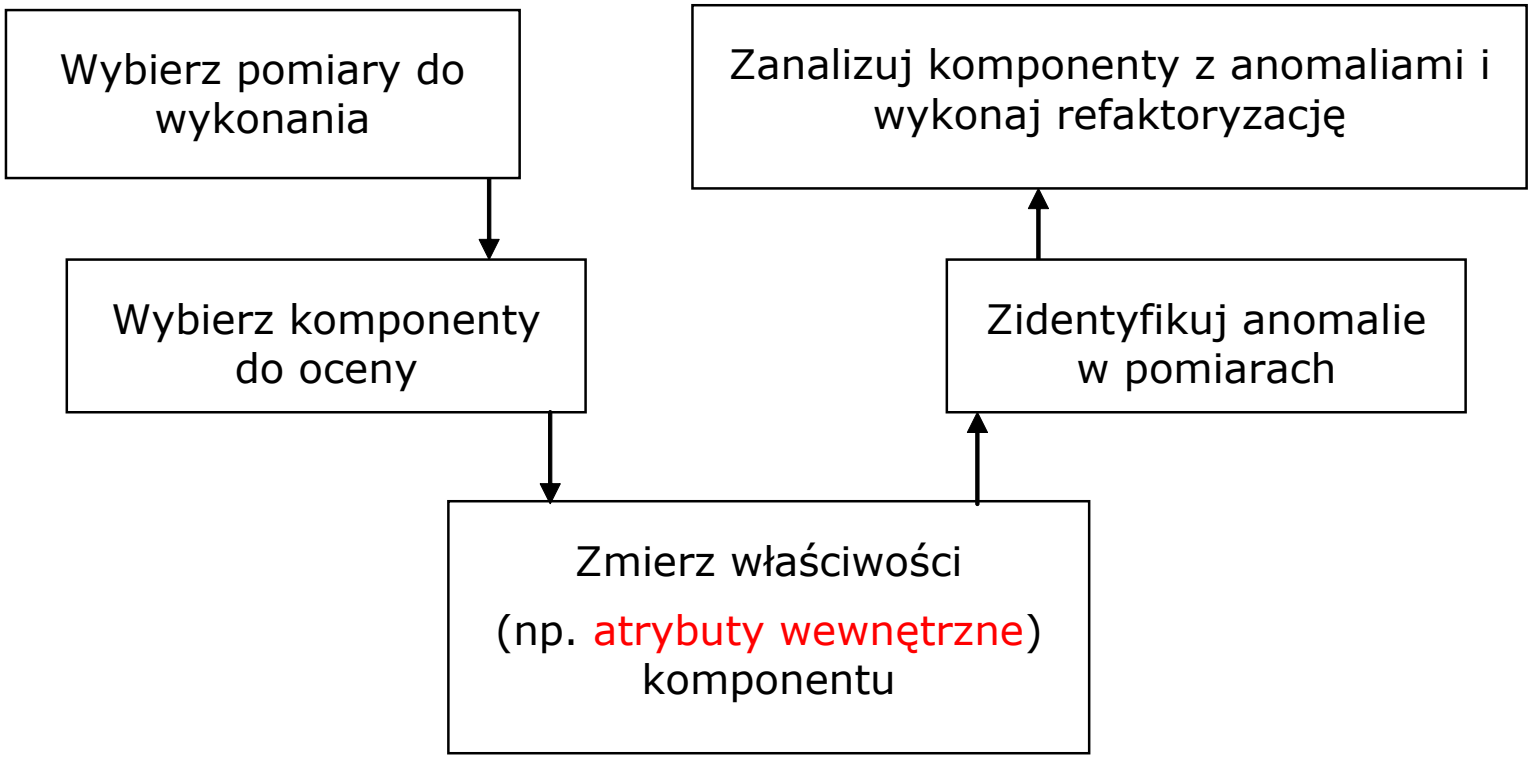
Klasyfikacja złożoności oprogramowania - związki między zewnętrznymi atrybutami oprogramowania i miarami wewnętrznych atrybutów (wg Brian Henderson-Sellers)



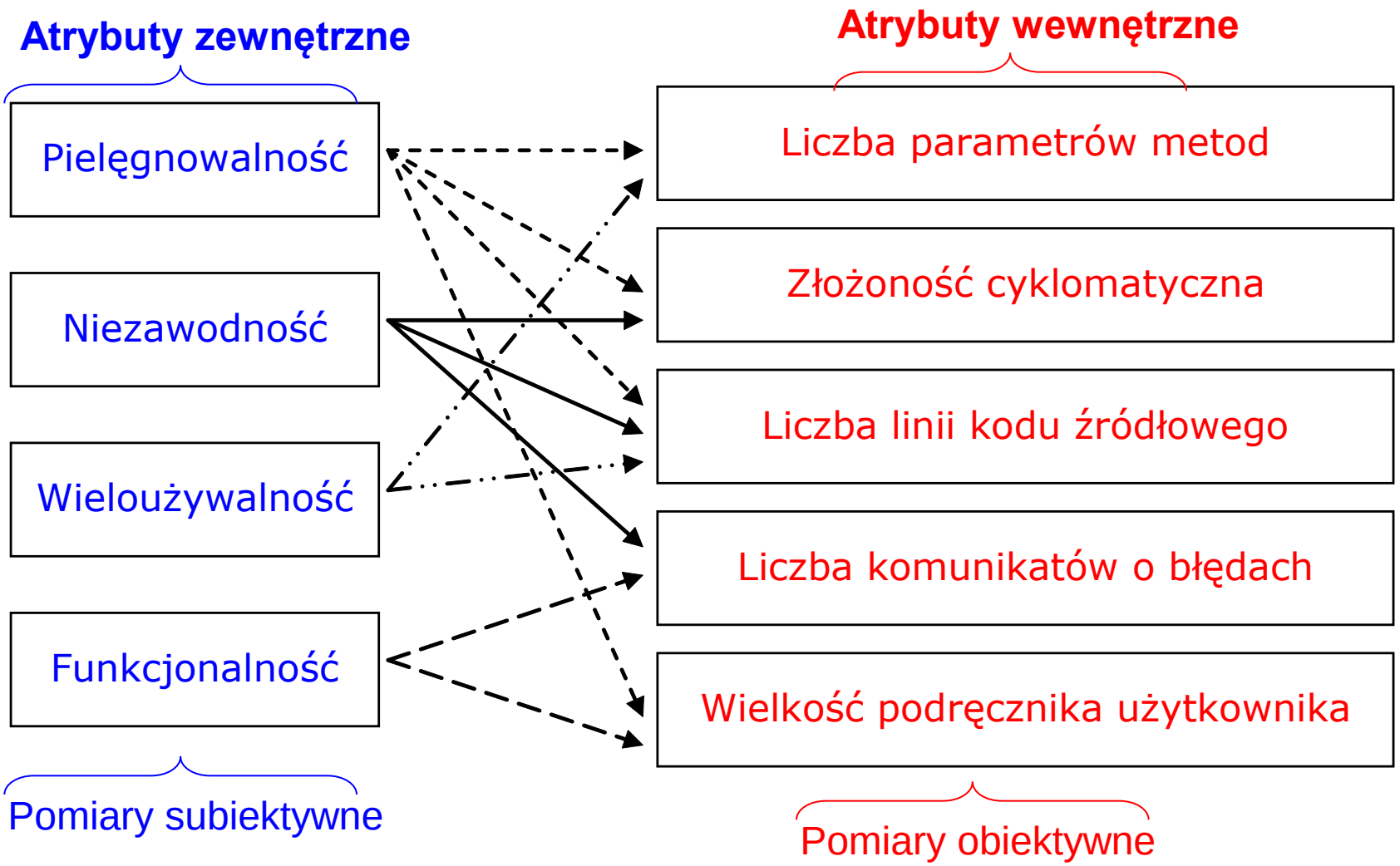
Miary predykcyjne i kontrolne [1]



Proces pomiaru produktu [1]



Związki funkcyjne niefunkcyjne między atrybutami zewnętrznymi i wewnętrznymi oprogramowania [1] – na podstawie atrybutów wewnętrznych ocenia się atrybuty zewnętrzne oprogramowania

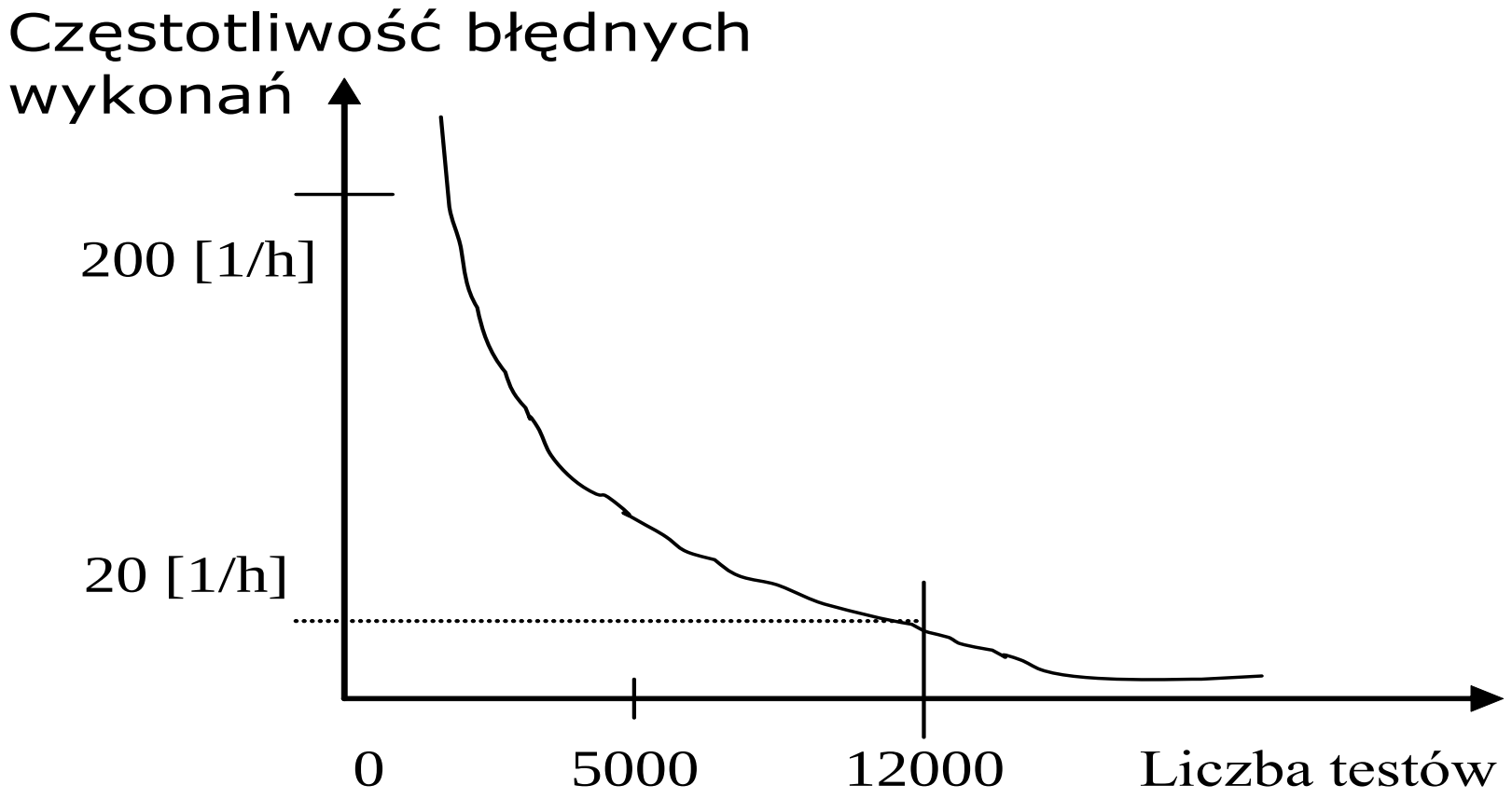


Główne zagadnienia

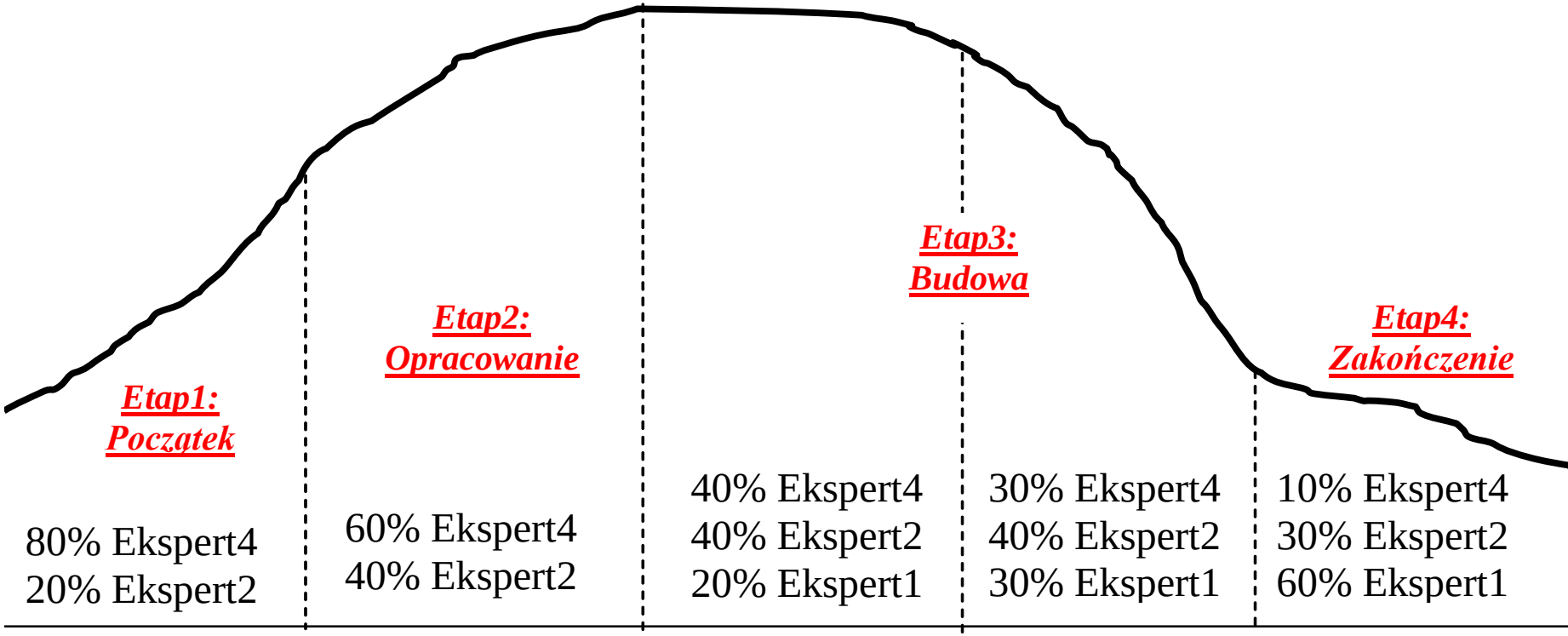
1. Modele procesu produkcji oprogramowania
2. Zapewnianie jakości i standardy [1]
3. Planowanie jakości [1]
4. Kontrolowanie jakości [1]
5. Miernictwo oprogramowania i miary
6. **Zalecenia dla projektów obiektowych**

Związek między liczbą przeprowadzonych testów i niezawodnością

Niezawodność programu jest częstotliwością jego błędnych wykonań. Rośnie ona logarytmicznie w zależności od liczby przeprowadzonych testów.



Wymagany poziom umiejętności w procesie produkcji oprogramowanie obiektowego metodami ieracyjno-rozwojowymi [2]



Ekspert1: Początkujący
Ekspert2: Ekspert technologii obiektowej
Ekspert3: Ekspert domeny projektu
Ekspert4: Ekspert domeny i technologii obiektowej

Poziom zadowolenia klienta [2]

	Metoda 1 (macierz korelacji)	Metoda 2 (analiza wielokrotnej regresji)	Metoda 3 (analiza regresji logistycznej)
Kolejność nierosnąca wartości atyrybutów	Niezawodność	Niezawodność	Niezawodność
	Łatwość użytkowania	Łatwość użytkowania	Łatwość użytkowania
	Łatwość instalacji	Łatwość instalacji	Dostępność
	Dokumentacja	Wydajność	Łatwość instalacji
	Wydajność	Łatwość serwisu	Łatwość serwisu
	Łatwość serwisu	Dokumentacja	Wydajność
	Dostępność	Dostępność	Dokumentacja

Dodatek – powtórzenie z PIO

- Wykaz metryk złożoności struktury kodu

Metryki złożoności międzymodułowej

Osłabienie 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

RFC = M + R oraz RFC' = M + R'

Zakres wartości (1 – 50)

gdzie

M – liczba 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 odwołania 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 wielożywalność

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.

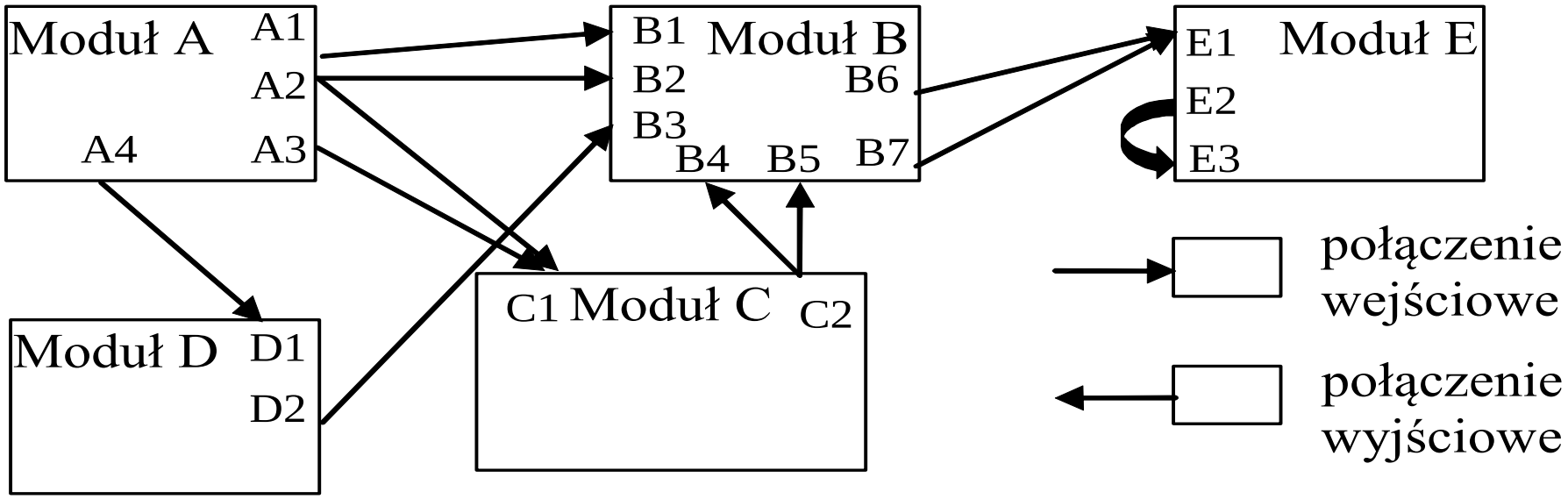
Ca - metryka połączeń wejściowych

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 B2, C1
- A3 łączy się C1
- A4 łączy się 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= |{<A1,B1>, <A2,C1>, <A4,D1>}| = 3 *//dowolny element wejściowy*
- Fan-in=|{}|=0
- R={<A1,B1>, <A2,B2>, <A2,C1>,<A3,C1>,<A4,D1>}
- |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

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 — **int** i=0
- porównanie — i <10
- zwiększanie zmiennej sterującej — i++
- liczba instrukcji w każdym bloku — **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 *ci* jest statyczną złożonością każdej z *i* - metod (złożoność cyklomatyczna materiał podany dalej). Jeżeli *ci* 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 wielożywalność 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)

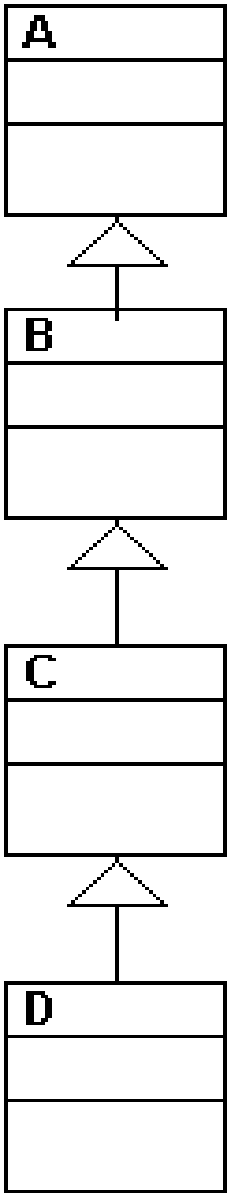
$$DIT = \frac{\sum \ glebokosc\ dziedziczenia}{calkowita\ liczba\ klas}$$

Uwagi:

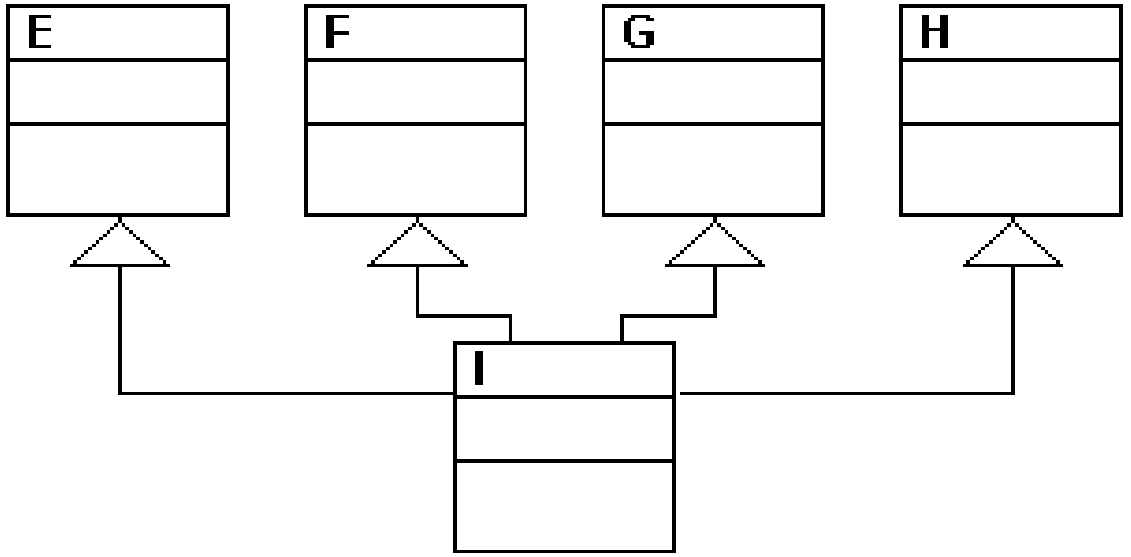
1. Przy głębokim drzewie dziedziczenia rośnie wieloużywalność
2. Przy głębokim drzewie dziedziczenia rośnie też liczba błędów, szczególnie w klasach należących do środkowych poziomów dziedziczenia

(1) Przykłady modeli do pomiaru metryk dziedziczenia

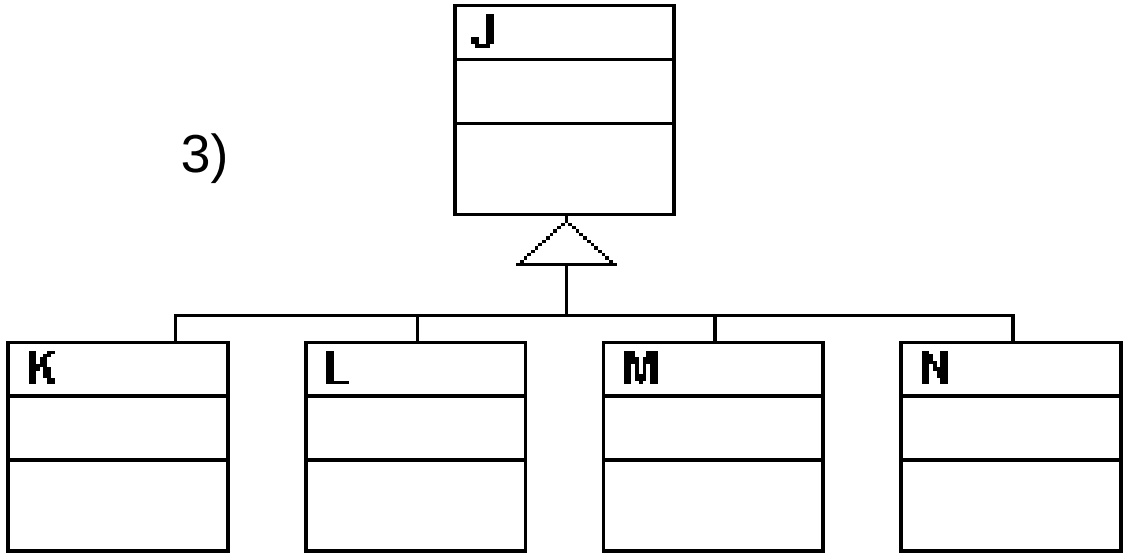
1)



2)

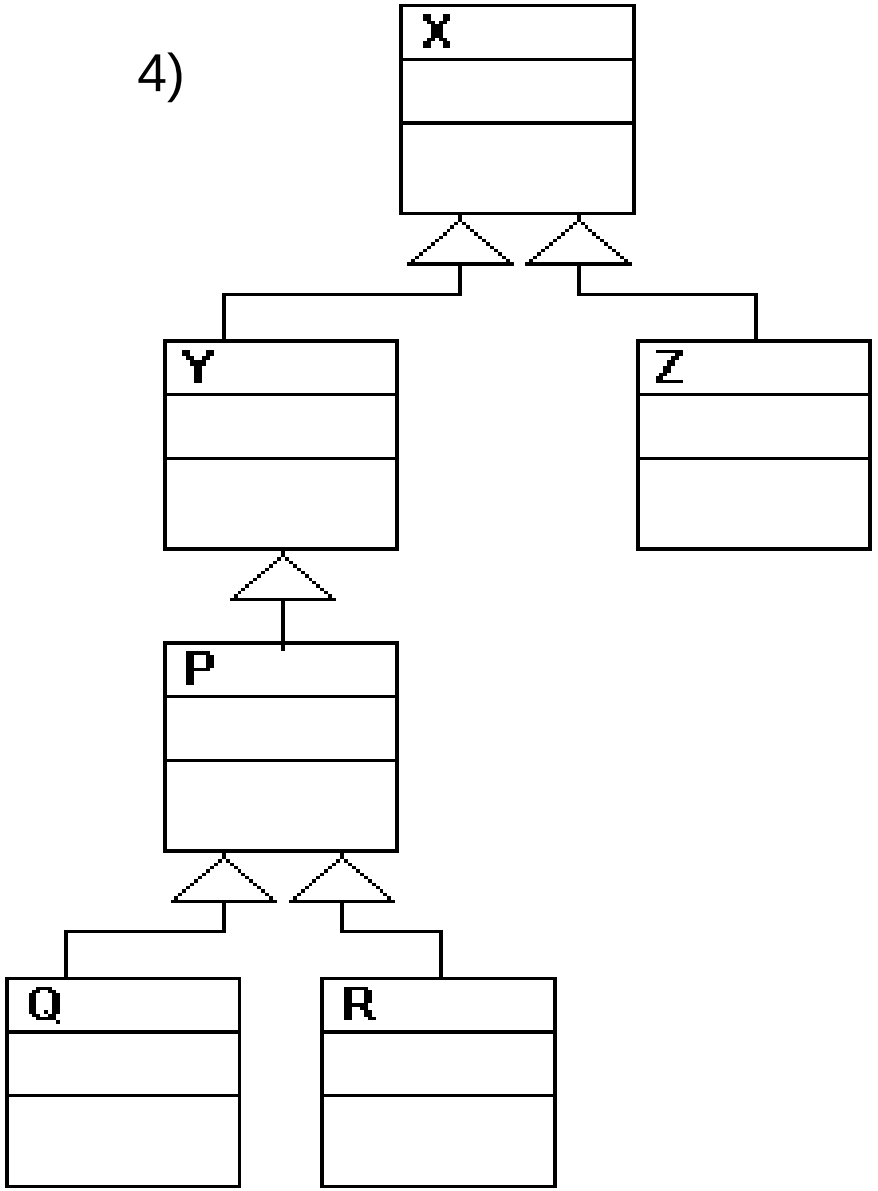


3)

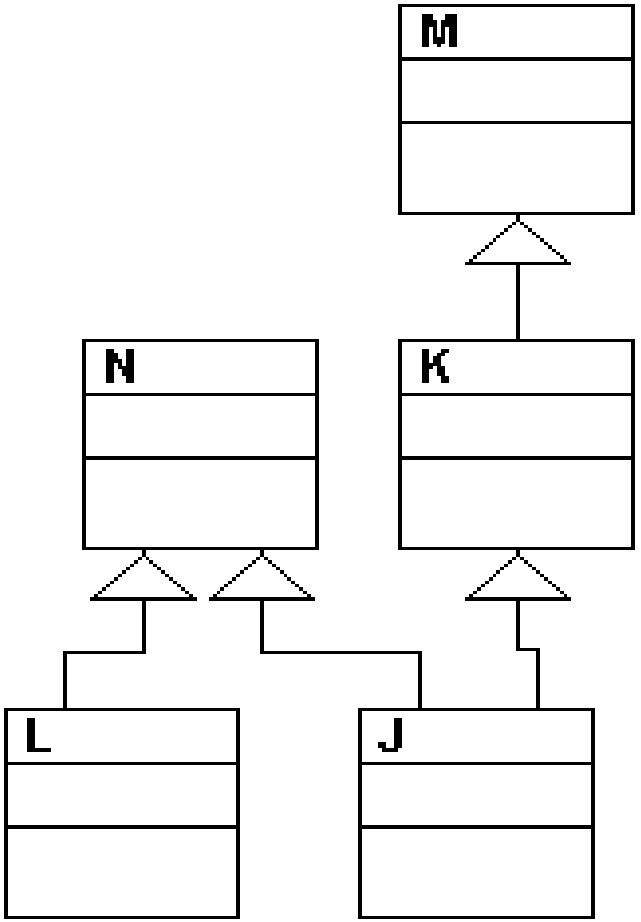


(2) Przykłady modeli do pomiaru metryk dziedziczenia

4)



5)



Metryka / Przykład	S (specialization)	U (reuse)	DIT
1*	1/3 $\rightarrow$ 0	3/4 $\rightarrow$ 1	(0+1+2+3)/4=1.5
2*	1/4 $\rightarrow$ 0	4/5 $\rightarrow$ 1	(0+0+0+0+(1+1+1+1)/4)/5=0.2
3	4/1 $\rightarrow$ μ	1/5 $\rightarrow$ 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

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

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

$$U = \frac{liczba\ \ NadTypów}{całkowita\ \ liczba\ \ klas}$$

Przykłady 1 i 2 reprezentują ubogi schemat dziedziczenia.

Wartości *U* bliska 1 oraz S bliską 0 określają liniowy model dziedziczenia. Wartości U<=1 oraz S >>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

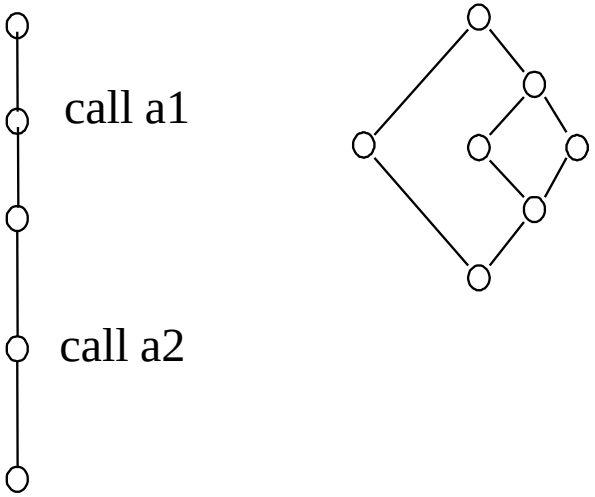
Liczba ta jest wyznaczana na podstawie grafu przedstawiającego drogi sterowania w programie, gdzie *n* jest liczbą wierzchołków grafu reprezentujących poszczególne instrukcje, w tym wywołania funkcji, *e* jest liczbą krawędzi grafu reprezentujących połączenia poszczególnych realizacji instrukcji, *p*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

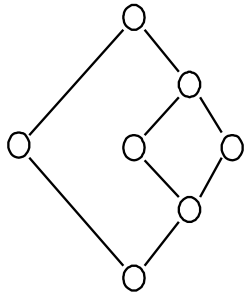
V(G) = e – n + 2*p

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

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

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

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

= e – n + 2 = e – n + 2*p

= 23-20+2=5

Metody a1 i a2 (przypadki a i b):

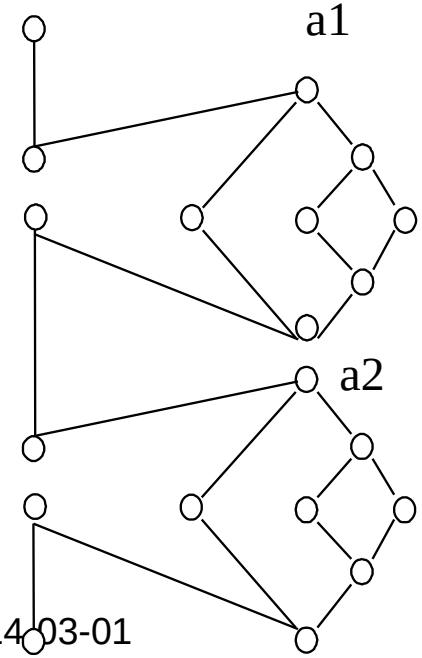
e=8, n=7, p=1

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

= e – n + 2 = e – n + 2*p

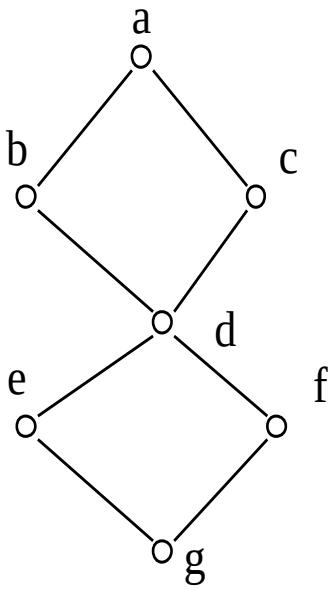
= 8-7+2=3

b)

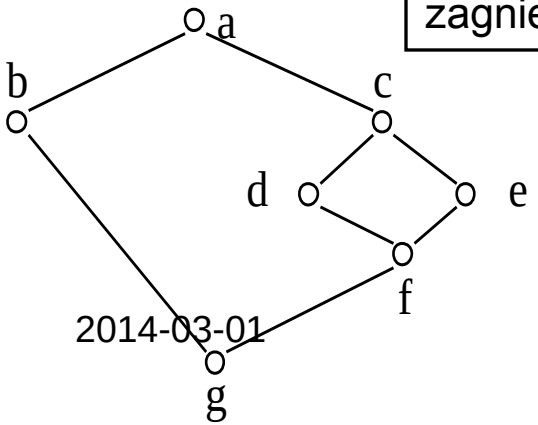


	Metoda a1	Metoda a2	Całość
a			
V(G)	3	3	7
$V_{LI}(G)$	3	3	5
b			
V(G)	3	3	5
$V_{LI}(G)$	3	3	5

(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>= 0)
c: {x=x-y; } *(gdy a==true)*
b: *(gdy a==false)*
d: **while** (y>= 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>= 0)
{ x=x-y; *(gdy a==true)*
c: **while** (y>= 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

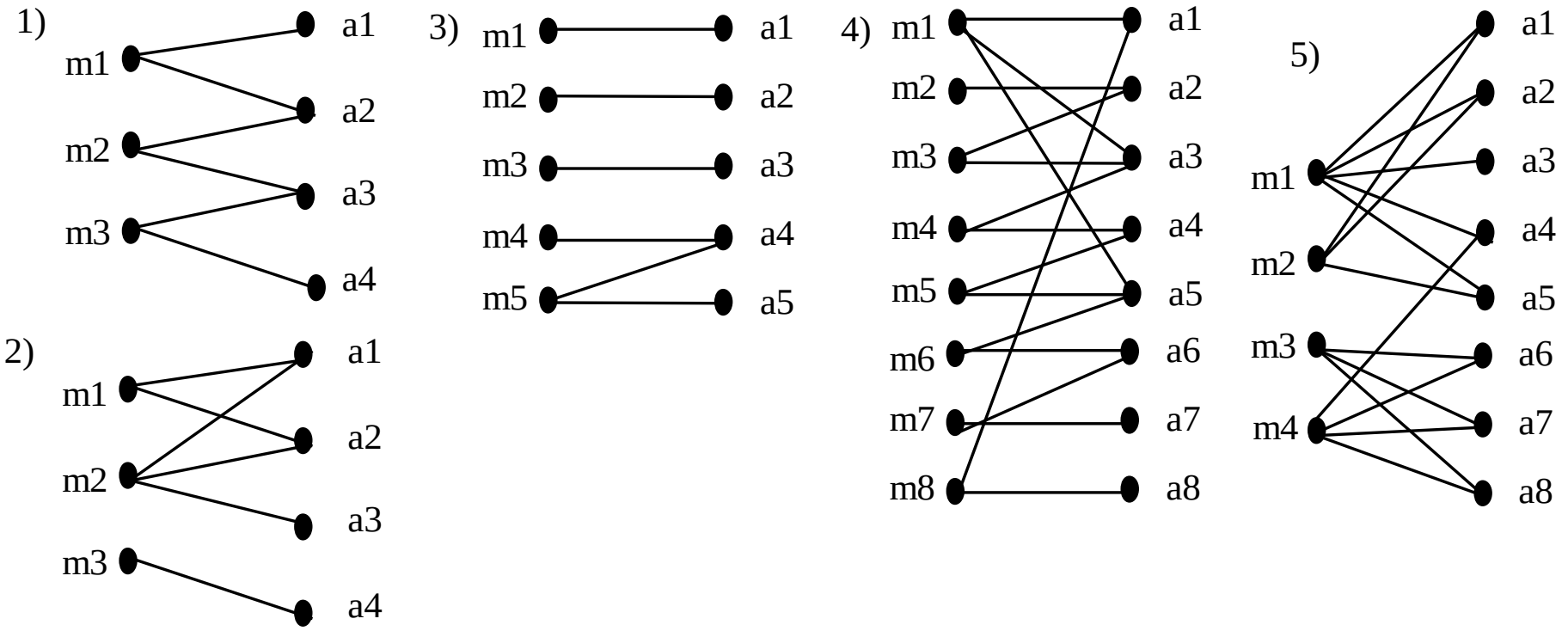
Metryki spójności klasy

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

Grafy dwudzielne jako modele klas do wyznaczania metryki LCOM



(1) Przykłady obliczeń metryki LCOM1

1) – trzy metody

- Metoda 1 ma zbiór atrybutów I1 = {a1, a2}
- Metoda 2 ma zbiór atrybutów I2 = {a2, a3}
- Metoda 3 ma zbiór atrybutów: I3 = {a3, a4}
- Zbiór rozłącznych par: P = {(I1, I3)} -> |P| = 1
- Zbiór spójnych par: Q = {(I1, I2), (I2, I3)} -> |Q| = 2
  - **LCOM = 0 dla |P| <= |Q|**

2) - trzy metody

- Metoda 1 ma zbiór atrybutów I1 = {a1, a2}
- Metoda 2 ma zbiór atrybutów I2 = {a1, a2, a3}
- Metoda 3 ma zbiór atrybutów: I3 = {a4}
- Zbiór rozłącznych par: P = {(I1, I3), (I2, I3)} -> |P| = 2
- Zbiór spójnych par: Q = {(I1, I2)} -> |Q| = 1
  - **LCOM = |P| - |Q| = 2 – 1 = 1 dla |P| > |Q|**

3) – pięć metod

- Metoda 1 ma zbiór atrybutów I1 = {a1}
- Metoda 2 ma zbiór atrybutów I2 = {a2}
- Metoda 3 ma zbiór atrybutów: I3 = {a3}
- Metoda 4 ma zbiór atrybutów: I4 = {a4}
- Metoda 5 ma zbiór atrybutów: I5 = {a4, a5}
- Zbiór rozłącznych par: P = {(I1, I2), (I1,I3), (I1,I4), (I1, I5), (I2, I3), (I2, I4), (I2, I5), (I3, I4), (I3,I5)} -> |P| = 9
- Zbiór spójnych par: Q = {(I4, I5)} -> |Q| = 1
  - **LCOM = |P| - |Q| = 9-1=8 dla |P| > |Q|**

(2) Przykłady obliczeń metryki LCOM1

4) – osiem metod

- Metoda 1 ma zbiór atrybutów I1 = {a1, a3, a5}
- Metoda 2 ma zbiór atrybutów I2 = {a2}
- Metoda 3 ma zbiór atrybutów: I3 = {a2, a3}
- Metoda 4 ma zbiór atrybutów: I4 = {a3, a4}
- Metoda 5 ma zbiór atrybutów: I5 = {a4, a5}
- Metoda 6 ma zbiór atrybutów: I6 = {a5, a6}
- Metoda 7 ma zbiór atrybutów: I7 = {a6, a7}
- Metoda 8 ma zbiór atrybutów: I8 = {a1, a8}
- Zbiór rozłącznych par: P = {(I1, I2), (I1, I7), (I2, I4), (I2, I5), (I2, I6), (I2, I7), (I2, I8), (I3, I5), (I3, I6), (I3, I7), (I3, I8), (I4, I6), (I4, I7), (I4, I8), (I5, I7), (I5, I8), (I6, I8), (I7, I8)}
  -> |P| = 18
- Zbiór spójnych par: Q = {(I1, I3), (I1, I4), (I1, I5), (I1, I6) (I1, I8) (I2, I3), (I3, I4), (I4, I5), (I5, I6), (I6, I7),}
  -> |Q| = 10

- **LCOM = |P| - |Q| = 18-10=8 dla |P| > |Q|**

(3) Przykłady obliczeń metryki LCOM1

5) – cztery metody

- Metoda 1 ma zbiór atrybutów I1 = {a1, a2, a3, a4, a5}
- Metoda 2 ma zbiór atrybutów I2 = {a1, a2, a5}
- Metoda 3 ma zbiór atrybutów: I3 = {a6, a7, a8}
- Metoda 4 ma zbiór atrybutów: I4 = {a4, a6, a7, a8}
- Zbiór rozłącznych par: P = {(I1, I3), (I2, I3), (I2, I4)} -> |P| = 3
- Zbiór spójnych par: Q = {(I1, I2), (I1, I4), (I3, I4)} -> |Q| = 3
  - **LCOM = 0 dla |P| <= |Q|**

Rozszerzenie definicji metryk spójności LCOM (1)

Metryka LCOM2 (Constantine & Graham, Henderson-Sellers)

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

- gdzie *m* jest liczbą wierzchołków zbioru *M* metod, *a* jest liczbą wierzchołków *A* atrybutów, natomiast wyrażenie *μ(Aj)* liczbą krawędzi grafu wiążącą atrybut *Aj* 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.

Kolejność grafów wg malejącej spójności: 5, 1, 4, 2, 3

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	8	0.75	0.8571
5	**4**	**8**	**15**	**1**	**0**	**0.5313**	**0.7083**

Rozszerzenie definicji metryk spójności LCOM (2)

Metryka LCOM3 (Constantine & Graham, Henderson-Sellers)

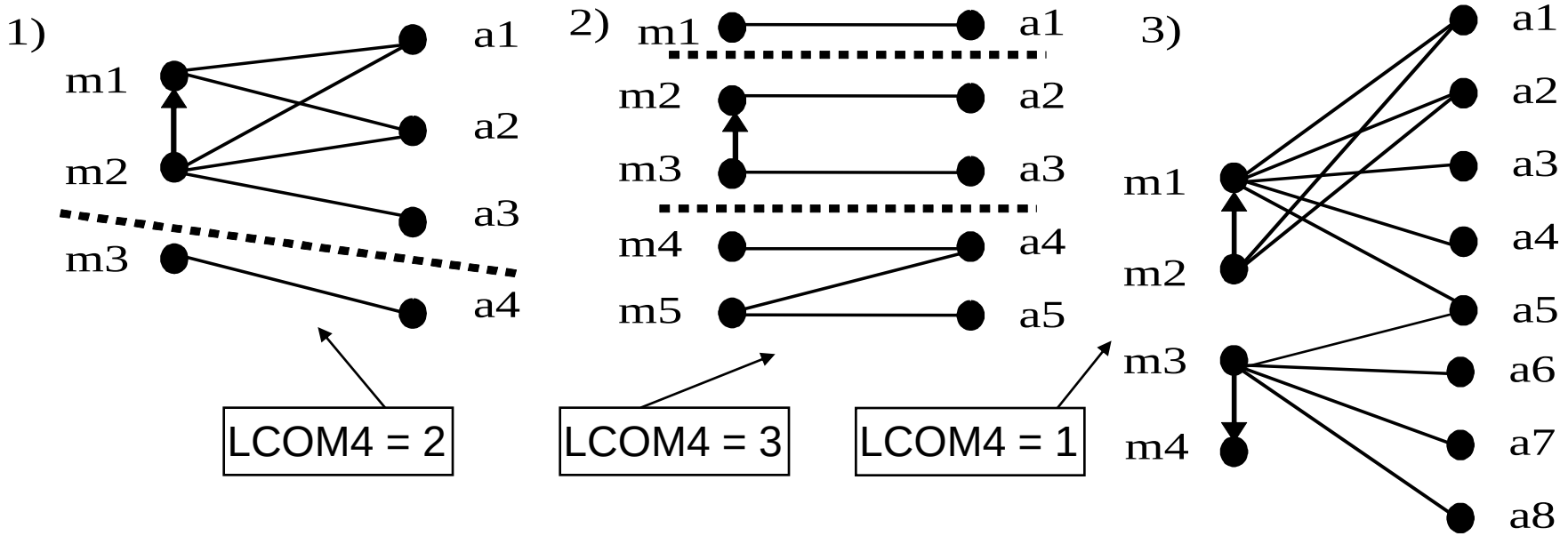
Zakres wartości „0..0.2”.

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

- gdzie *m* jest liczbą wierzchołków zbioru *M* metod, *a* jest liczbą wierzchołków *A* atrybutów, natomiast wyrażenie $\mu(Aj)$ liczbą krawędzi grafu wiążącą atrybut *Aj* 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.

Rozszerzenie definicji metryk spójności LCOM (3)

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”.

Przykład metryk trzech systemów

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