INŻYNIERIA OPROGRAMOWANIA

wykład 7:
MIARY OPROGRAMOWANIA

dr inż. Leszek Grocholski
Zakład Inżynierii Oprogramowania
Instytut Informatyki
Uniwersytet Wrocławski

POMIAR OPROGRAMOWANIA

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

Jednostki przydzielane atrybutom nazywane są **miarą** danego atrybutu.

Metryka (*metric*) jest to proponowana (postulowana) miara.

Nie zawsze charakteryzuje ona w sposób obiektywny dany atrybut. Np. ilość linii kodu (LOC) jest metryką charakteryzującą atrybut “długość programu źródłowego”, ale nie jest miarą ani złożoności ani rozmiaru programu (choć występuje w tej roli).

Co mierzyć ?

Proces: każdy określony ciąg działań w ramach analizy, projektowana, wytwarzania lub eksploatacji oprogramowania.

Produkt: każdy przedmiot powstały w wyniku procesu: kod źródłowy, specyfikację projektową, udokumentowaną modyfikację, plan testów, dokumentację, itd.

Zasób: każdy element niezbędny do realizacji procesu: osoby, narzędzia, metody wytwarzania, itd.

Elementy pomiaru oprogramowania - PRODUKTY

Obiekty	**Atrybuty bezpośrednio mierzalne**	**Wskaźniki syntetyczne**
Specyfikacje	rozmiar, ponowne użycie, modularność, nadmiarowość, funkcjonalność, poprawność składniową, ...	zrozumiałość, pielęgnacyjność, ...
Projekty	rozmiar, ponowne użycie, modularność, spójność, funkcjonalność,...	jakość, złożoność, pielęgnacyjność, ...
Kod	rozmiar, ponowne użycie, modularność, spójność, złożoność, strukturalność, ...	niezawodność, używalność, pielęgnacyjność, ...
Dane testowe	rozmiar, poziom pokrycia,...	jakość,...
....	....	....

Elementy pomiaru oprogramowania - PRODUKTY cd ..

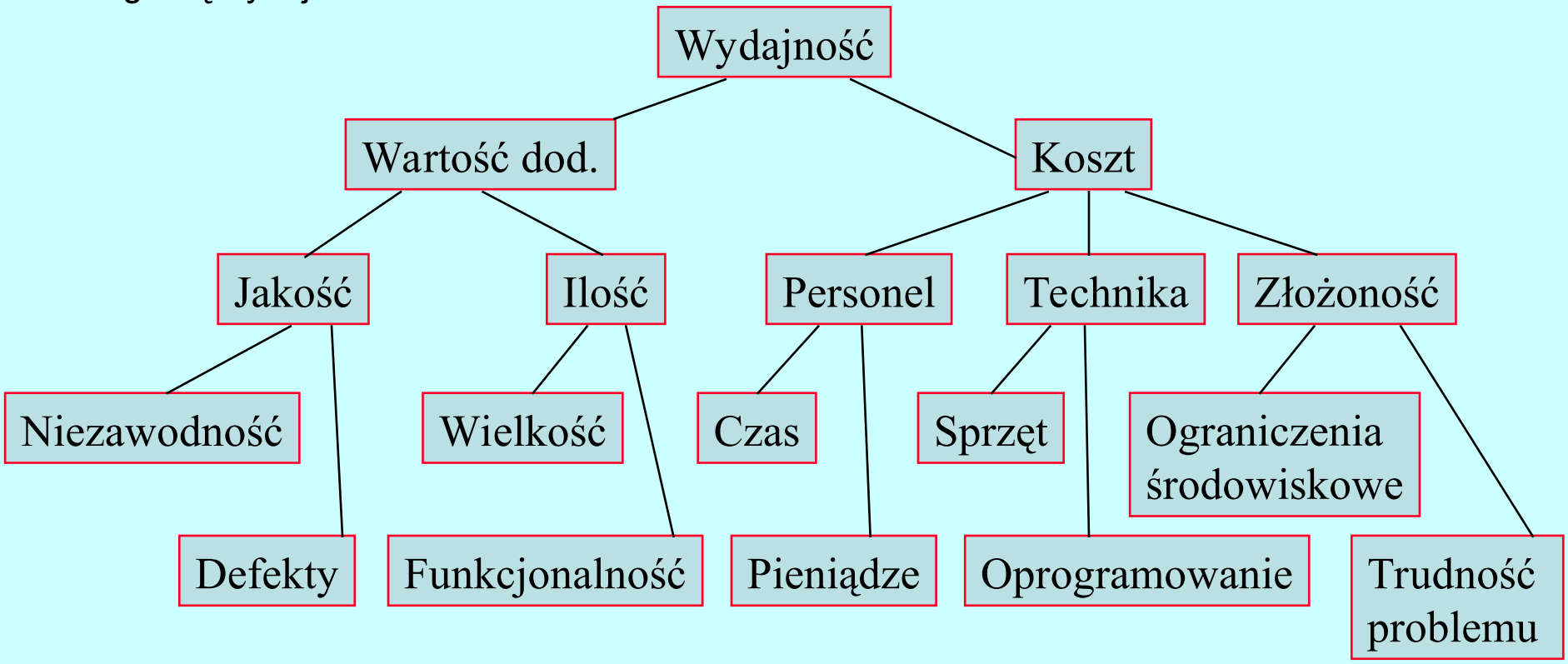
Obiekty	**Atrybuty bezpośrednio mierzalne**	**Wskaźniki syntetyczne**
Specyfikacja architektury	czas, nakład pracy, liczba zmian wymagań, ...	jakość, koszt, stabilność, ...
Projekt szczegółowy	czas, nakład pracy, liczba znalezionych usterek specyfikacji,...	koszt, opłacalność, ...
Testowanie	czas, nakład pracy, liczba znalezionych błędów kodu, ...	koszt, opłacalność, stabilność, ...
....	....	....

Elementy pomiaru oprogramowania - ZASOBY

Obiekty	**Atrybuty bezpośrednio mierzalne**	**Wskaźniki syntetyczne**
Personel	wiek, cena, ...	wydajność, doświadczenie, inteligencja, ...
Zespoły	wielkość, poziom komunikacji, struktura,...	wydajność, jakość, ...
Oprogramowanie	cena, wielkość, ...	używalność, niezawodność, ...
Sprzęt	cena, szybkość, wielkość pamięci	niezawodność, ...
Biura	wielkość, temperatura, oświetlenie,...	wygoda, jakość,...
....	....	....

Model i miara wydajności ludzi

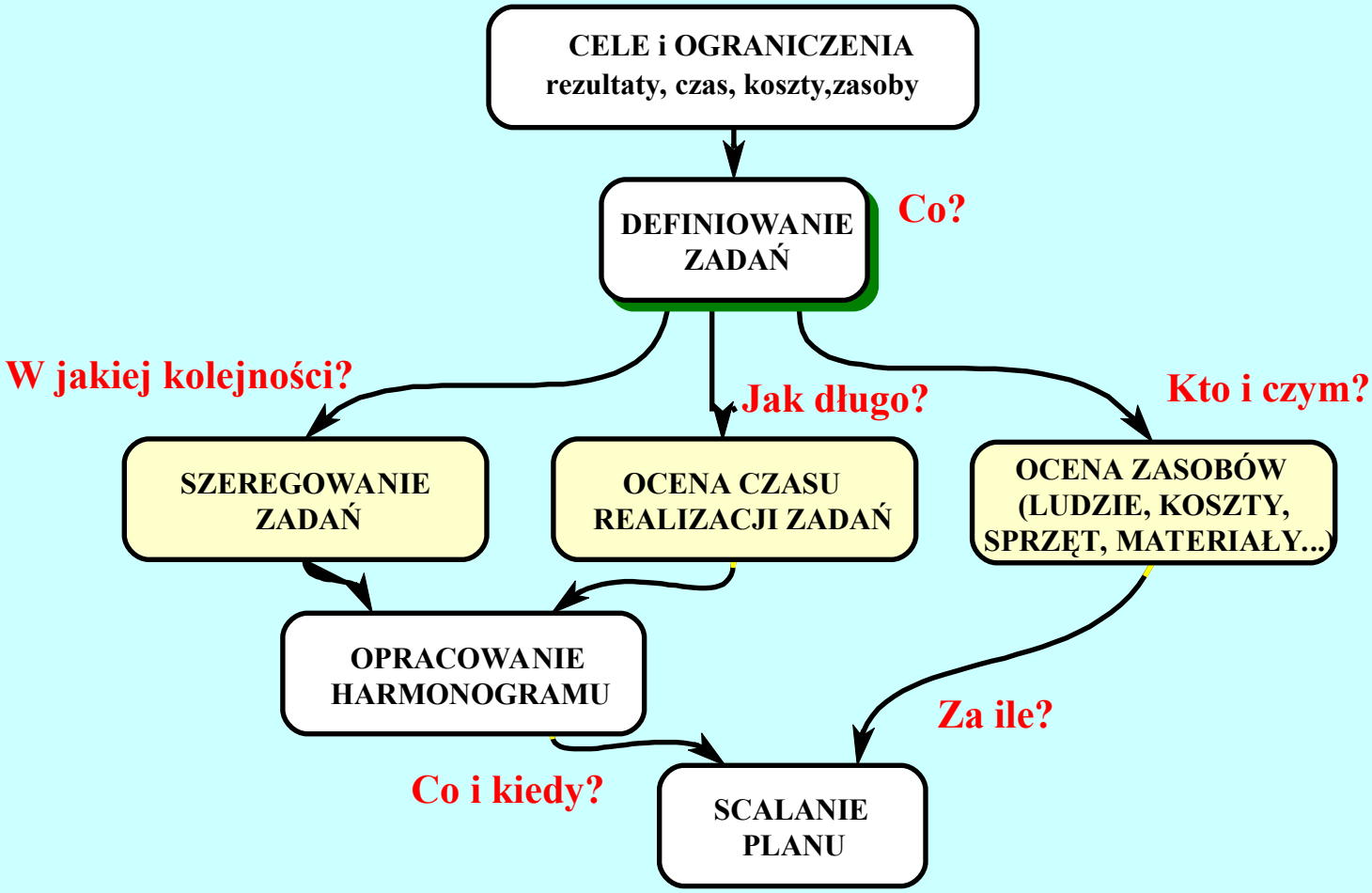
Czynniki wpływające
na ogólną wydajność



Uwaga: Mylące, wręcz niebezpieczne jest zastępowanie wielu miar jedną miarą np. długością wyprodukowanego kodu.

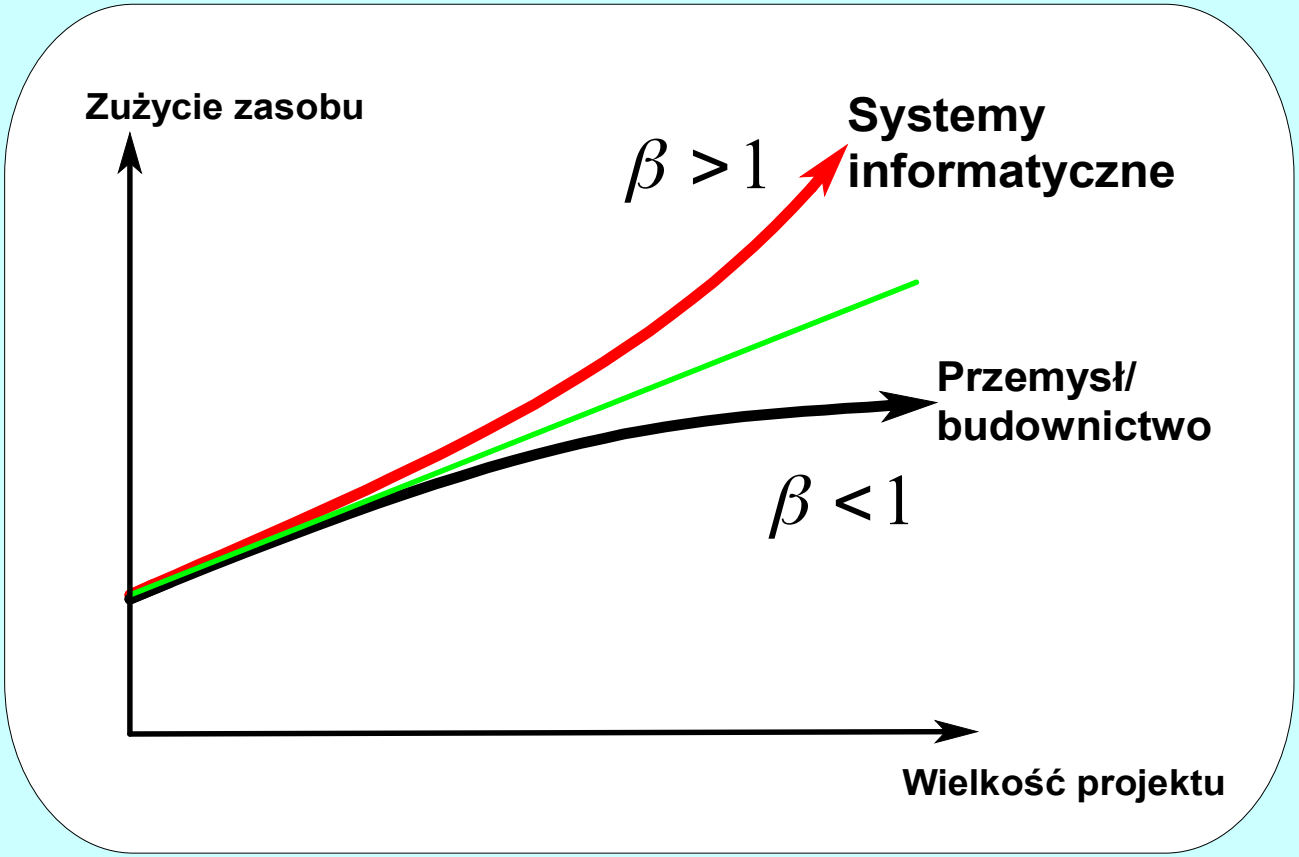
Ocena złożoności w planowaniu projektu

(tak jak odmiana przez przypadki, kto, co ….)



Efekt skali

Zużycie Zasobu = Zużycie Stałe + K * Wielkość Projektu $^{\beta}$



Czynniki wpływające na efekt skali

Czynniki spłaszczenia krzywej:

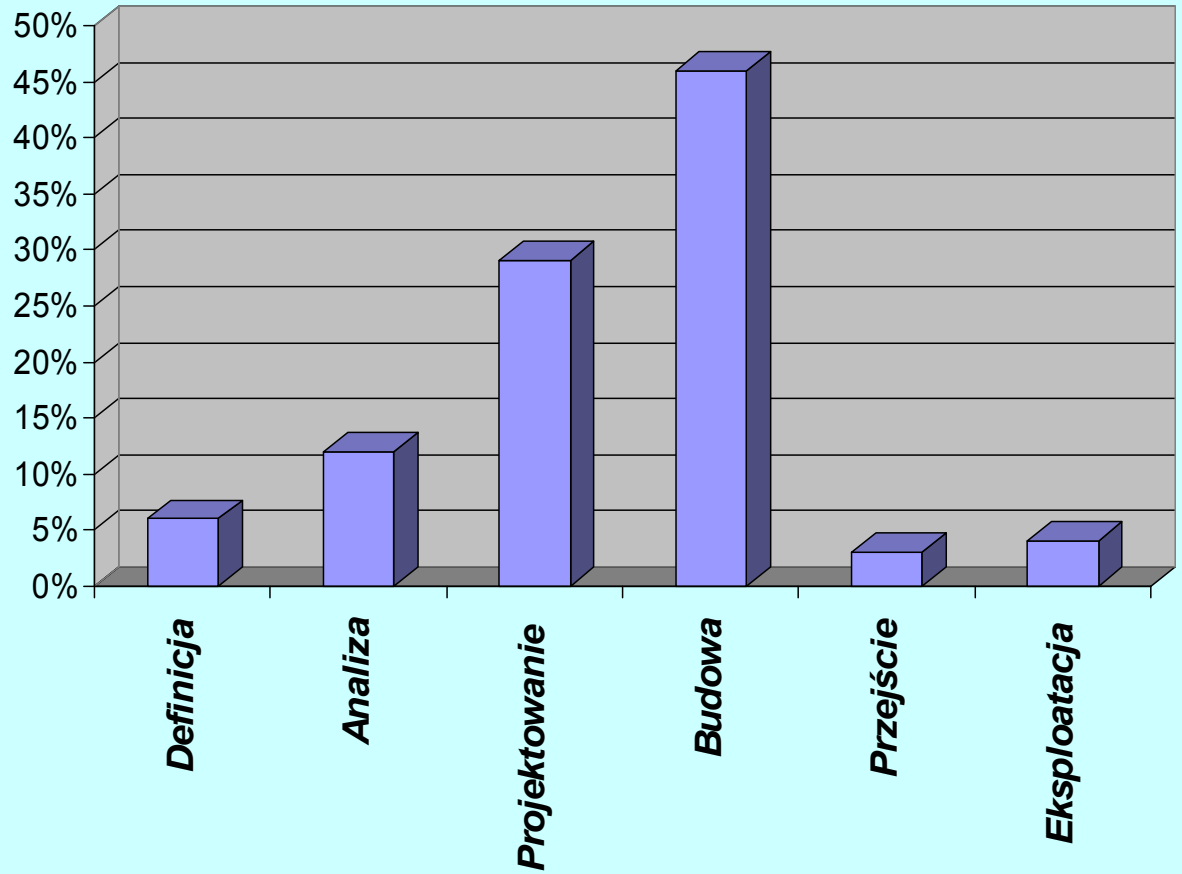
- Specjalizacja
- Uczenie się, doświadczenie
- Narzędzia CASE
- Wspomaganie dokumentowania
- Biblioteki gotowych elementów
- Stałe koszty projektu

Czynniki wzrostu krzywej:

- Koszty zarządzania
  (czas produkcyjny/nie)
- Lawinowy wzrost ilości powiązań
- Komunikacja wewnątrz zespołu
  → czas szybko wzrasta w miarę dodawania nowych członków
- Wzrost złożoności testowania

FAZY a koszty wytwarzania oprogramowania

Empiryczne koszty poszczególnych faz wytwarzania oprogramowania systemów informatycznych:



Źródło: Oracle Corp. Badaniom podlegały realizacje systemów przetwarzania danych, realizowane metodą CDM, prze użyciu narzędzi CASE firmy Oracle.

Metoda szacowania kosztów COCOMO

COnstructive COst MOdel

Wymaga oszacowania liczby instrukcji, z których będzie składał się system. Rozważane przedsięwzięcie jest następnie zaliczane do jednej z klas:

Przedsięwzięć łatwych (organicznych, *organic*). Klasa ta obejmuje przedsięwzięcia wykonywane przez stosunkowo małe zespoły, złożone z osób o podobnych wysokich kwalifikacjach. Dziedzina jest dobrze znana. Przedsięwzięcie jest wykonywane przy pomocy dobrze znanych metod i narzędzi.

Przedsięwzięć niełatwych (pół-oderwanych, *semi-detached*). Członkowie zespołu różnią się stopniem zaawansowania. Pewne aspekty dziedziny problemu nie są dobrze znane.

Przedsięwzięć trudnych (np.. osadzonych, *embedded*). Obejmują przedsięwzięcia realizujące systemy o bardzo złożonych wymaganiach. Dziedzina problemu, stosowane narzędzia i metody są w dużej mierze nieznane. Większość członków zespołu nie ma doświadczenia w realizacji podobnych zadań.

Metoda szacowania kosztów COCOMO (2)

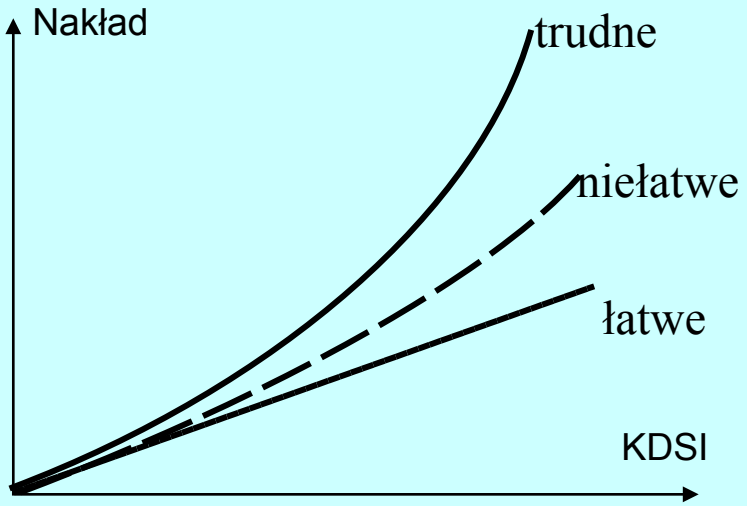
Podstawowy wzór dla oszacowania nakładów w metodzie COCOMO:

$$\text{Nakład [osobomiesiące]} = A * K^{b}$$ (zależność wykładnicza)

K (określane jako KDSI, *Kilo (thousand) of Delivered Source Code Instructions*) oznacza rozmiar kodu źródłowego mierzony w tysiącach linii. KDSI nie obejmuje kodu, który nie został wykorzystany w systemie.
Wartości stałych A i b zależą od klasy, do której zaliczono przedsięwzięcie:

Przedsięwzięcie łatwe:	$\text{Nakład} = 2.4 * K^{1.05}$
Przedsięwzięcie niełatwe:	$\text{Nakład} = 3 * K^{1.12}$
Przedsięwzięcie trudne:	$\text{Nakład} = 3.6 * K^{1.20}$

Dla niewielkich przedsięwzięć są to zależności bliskie liniowym. Wzrost jest szczególnie szybki dla przedsięwzięć trudnych (duży rozmiar kodu).



Metoda szacowania kosztów COCOMO (3)

Metoda COCOMO zakłada, że znając nakład można oszacować czas realizacji przedsięwzięcia, z czego wynika przybliżona wielkość zespołu. Z obserwacji wiadomo, że dla każdego przedsięwzięcia istnieje optymalna liczba członków zespołu wykonawców. Zwiększenie tej liczby może nawet wydłużyć czas realizacji.
Proponowane są następujące wzory:

Przedsięwzięcie łatwe:	Czas [miesiące] = 2.5 * Nakład $^{0.32}$
Przedsięwzięcie niełatwe:	Czas [miesiące] = 2.5 * Nakład $^{0.35}$
Przedsięwzięcie trudne:	Czas [miesiące] = 2.5 * Nakład $^{0.38}$

Otrzymane w ten sposób oszacowania powinny być skorygowane przy pomocy tzw. czynników modyfikujących. Biorą one pod uwagę następujące atrybuty przedsięwzięcia:

- wymagania wobec niezawodności systemu
- rozmiar bazy danych w stosunku do rozmiaru kodu
- złożoność systemu: złożoność struktur danych, złożoność algorytmów, komunikacja z innymi systemami, stosowanie obliczeń równoległych
- wymagania co do wydajności systemu
- ograniczenia pamięci
- zmienność sprzętu i oprogramowania systemowego tworzącego środowisko pracy systemu

Wady metody COCOMO

Liczba linii kodu jest znana dokładnie dopiero wtedy, gdy system jest napisany. Szacunki są zwykle obarczone bardzo poważnym błędem (niekiedy ponad 100%)

Określenie “linii kodu źródłowego” inaczej wygląda dla każdego języka programowania. Jedna linia w Smalltalk’u jest równoważna 10-ciu linii w C. Dla języków 4GL i języków zapytań ten stosunek może być nawet 1000 : 1.

Koncepcja oparta na liniach kodu źródłowego jest całkowicie nieadekwatna dla nowoczesnych środków programistycznych, np. opartych o programowanie wizyjne.

Zły wybór czynników modyfikujących może prowadzić do znacznych rozbieżności pomiędzy oczekiwanym i rzeczywistym kosztem przedsięwzięcia.

Żadna metoda przewidywania kosztów nie jest więc doskonała i jest oparta na szeregu arbitralnych założeń. Niemniej dla celów planowania tego rodzaju metody stają się koniecznością. Nawet niedoskonała metoda jest zawsze lepsza niż “sufit”.

Analiza Punktów Funkcyjnych

Metoda analizy punktów funkcyjnych (FPA), opracowana przez Albrechta z firmy IBM w latach 1979-1983 bada pewien zestaw wartości. Łączy ona własności metody, badającej rozmiar projektu programu z możliwościami metody badającej produkt programowy.

Liczbę nie skorygowanych punktów funkcyjnych wylicza się na podstawie formuły korzystając z następujących danych:

- **Wejścia użytkownika**: obiekty wejściowe wpływających na dane w systemie
- **Wyjścia użytkownika:** obiekty wyjściowe związane z danymi w systemie
- **Zbiory danych wewnętrzne**: liczba wewnętrznych plików roboczych
- **Zbiory danych zewnętrzne:** liczba plików zewnętrznych zapełnianych przez system
- **Zapytania (interfejsy) zewnętrzne:** interfejsy z otoczeniem programu

UFP - Nieskorygowane Punkty Funkcyjne

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} w_{ij} \star n_{ij}$$

UFP- nieskorygowane punkty funkcyjne

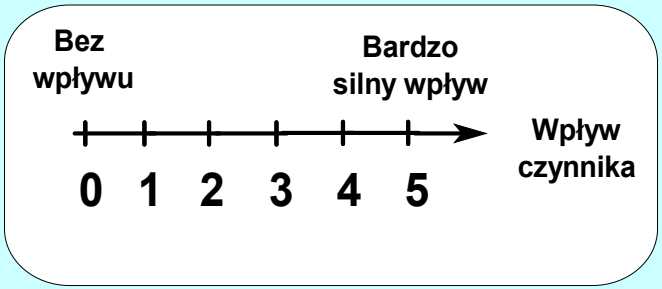
gdzie: w_{ij} - wagi, n_{ij} - ilość elementów

		Wagi przypisywane komponentom:		
	Czynnik złożoności	**prosty**	**średni**	**złożony**
		3	4	6
i = 1	Wejścia użytkownika	4	5	7
i = 2	Wyjścia użytkownika	7	10	15
i = 3	Zbiory danych wewnętrzne	5	7	10
i = 4	Zbiory danych zewnętrzne	3	4	6
i = 5	Zapytania zewnętrzne			
		j = 1	j = 2	j = 3

Korekcja Punktów Funkcyjnych

1. występowanie urządzeń komunikacyjnych
2. rozproszenie przetwarzania
3. długość czasu oczekiwania na odpowiedź systemu (nacisk na szybkość)
4. stopień obciążenia istniejącego sprzętu
5. częstotliwość wykonywania dużych transakcji
6. wprowadzanie danych w trybie bezpośrednim
7. wydajność użytkownika końcowego
8. aktualizacja danych w trybie bezpośrednim
9. złożoność przetwarzania danych
10. możliwość ponownego użycia programów w innych zastosowaniach
11. łatwość instalacji
12. łatwość obsługi systemu
13. rozproszenie terytorialne
14. łatwość wprowadzania zmian - pielęgnowania systemu

Skorygowane Punkty Funkcyjne



kompleksowy współczynnik korygujący

$$VAF = \sum_{k=1}^{14} k_k$$

(Skorygowane) punkty funkcyjne (FPs):

$$FP = (0{,}65 + 0{,}01 * VAF) * UFP$$

$$FP = (0{,}65 \ldots 1{,}35) * UFP$$

Kolejność obliczeń Punktów Funkcyjnych

1. Identyfikacja systemu
2. Obliczenie współczynnika korygującego
3. Wyznaczenie ilości zbiorów danych i ich złożoności
4. Wyznaczenie ilości i złożoności elementów funkcjonalnych (we, wy, zbory we, wy)
5. Realizacja obliczeń
6. Weryfikacja
7. Raport, zebranie recenzujące

Przykład obliczania punktów funkcyjnych

Elementy	Proste	Waga	Średnie	Waga	Złożone	Waga	Razem
Wejścia	**2**	3	**5**	4	**3**	6	**44**
Wyjścia	**10**	4	**4**	5	**5**	7	**95**
Zbiory wew.	**3**	7	**5**	10	**2**	15	**101**
Zbiory zew.	**0**	5	**3**	7	**0**	10	**21**
Zapytania	**10**	3	**5**	4	**12**	6	**122**
							Łącznie 383

Aplikacje a Punkty Funkcyjne

1 FP $\approx$ 125 instrukcji w C

10 FPs - typowy mały program tworzony samodzielnie przez jednego „dobrego” programistę przez 1 miesiąc

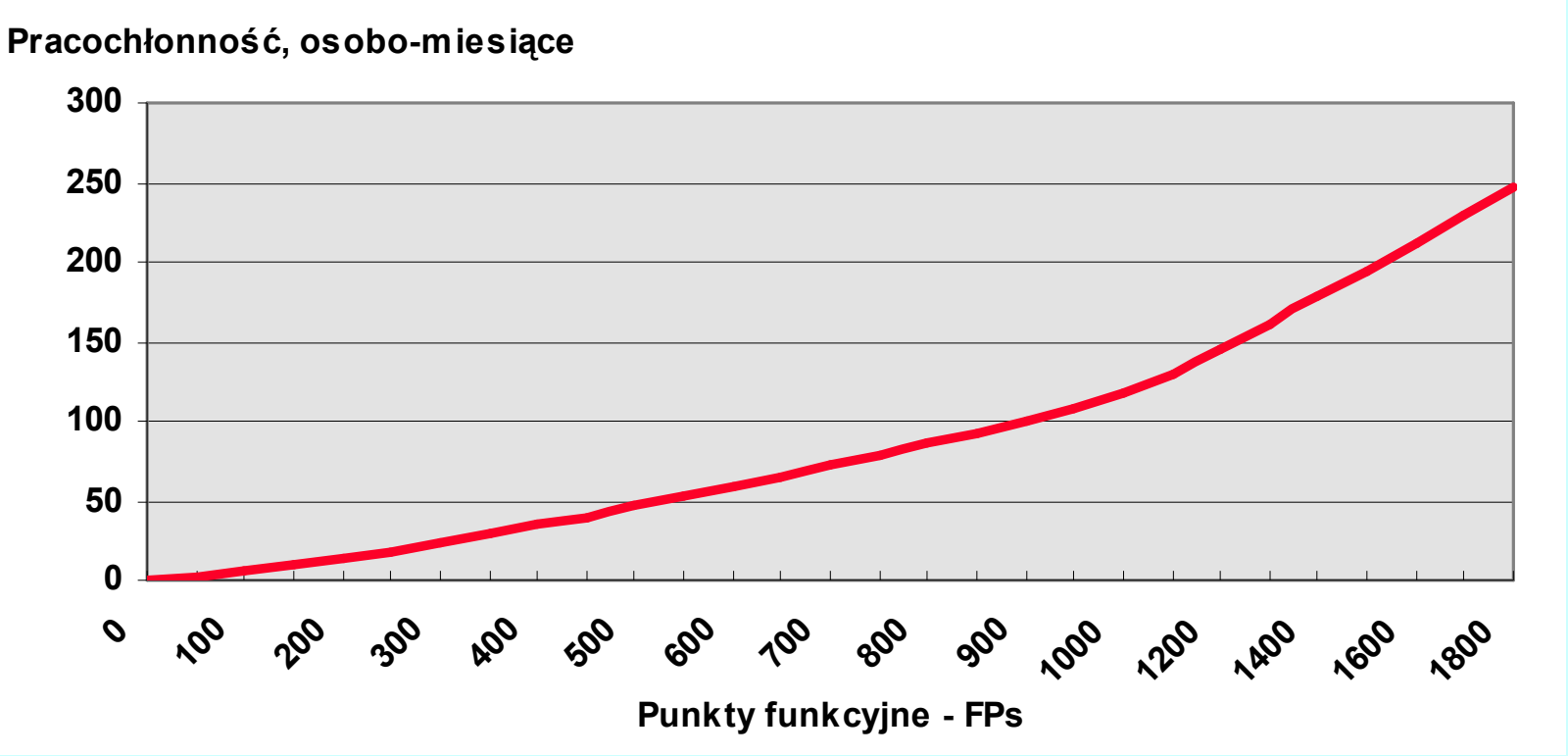
100 FPs - większość popularnych aplikacji; wartość typowa dla aplikacji tworzonych przez klienta samodzielnie (6 m-cy)

1 000 FPs - komercyjne aplikacje w MS Windows, małe aplikacje klient-serwer (10 osób, ponad 12 m-cy)

10 000 FPs - systemy złożone (100 osób, ponad 18 m-cy)

100 000 FPs - MS Windows NT, MVS, systemy militarne

Punkty Funkcyjne a pracochłonność



Wykorzystanie punktów funkcyjnych

- Ocena złożoności realizacji systemów

- Audyt projektów

- Wybór systemów informatycznych funkcjonujących w przedsiębiorstwie do reinżynierii (wg. koszt utrzymania/FPs)

- Szacowanie liczby testów

- Ocena jakości pracy i wydajności zespołów ludzkich

- Ocena stopnia zmian, wprowadzanych przez użytkownika na poszczególnych etapach budowy systemu informatycznego

- Prognozowanie kosztów pielęgnacji i rozwoju systemów

- Porównanie i ocena różnych ofert dostawców oprogramowania pod kątem merytorycznym i kosztowym

Punkty Funkcyjne a języki baz danych

Typ języka lub konkretny język	Poziom języka wg. SPR	Efektywność LOC/FP
Access	8.5	38
ANSI SQL	25.0	13
CLARION	5.5	58
CA Clipper	17.0	19
dBase III	8.0	40
dBase IV	9.0	36
DELPHI	11.0	29
FOXPRO 2.5	9.5	34
INFORMIX	8.0	40
MAGIC	15.0	21
ORACLE	8.0	40
Oracle Developer 2000	14.0	23
PROGRESS v.4	9.0	36
SYBASE	8.0	40

wg. Software Productivity Research

Punkty Funkcyjne a wydajność zespołu

Poziom języka wg. SPR *	Średnia produktywność FPs/osobomiesiąc
1 - 3	5 - 10
4 - 8	10 - 20
9 - 15	16 - 23
16 - 23	15 - 30
24 - 55	30 - 50
>55	40 - 100

** wg. Software Productivity Research*

Uwzglednienie różnorodnych aspektow

Różnorodne metryki uwzględniają m.in. następujące aspekty:

- wrażliwość na błędy,
- możliwości testowania,
- częstotliwość występowania awarii,
- dostępność systemu,
- propagacja błędów,
- ilość linii kodu, złożoność kodu, złożoność programu,
- złożoność obliczeniową, funkcjonalną, modułową,
- łatwość implementacji,
- rozmiar dokumentacji,
- ilość zadań wykonanych terminowo i po terminie,
- współzależność zadań,
- wielkość i koszt projektu,
- czas trwania projektu,
- zagrożenia projektu (ryzyko),
- czas gotowości produktu,
- kompletność wymagań, kompletność planowania,
- stabilność wymagań,
- odpowiedniość posiadanych zasobów sprzętowych, materiałowych i ludzkich,
- efektywność zespołu, efektywność poszczególnych osób.

Przykłady metryk oprogramowania

- Metryki zapisu projektu, kodu programu

- rozmiar projektu, kodu programu (liczba modułów/obiektów, liczba linii kodu, komentarza, średni rozmiar komponentu)
- liczba, złożoność jednostek syntaktycznych i leksykalnych
- złożoność struktury i związków pomiędzy komponentami programu

  (procesy, funkcje, moduły, obiekty itp..)

- Metryki uzyskiwanego produktu

- rozmiar
- architektura
- struktura
- jakość użytkowania i pielęgnacji
- złożoność

Przykłady metryk oprogramowania, cd.

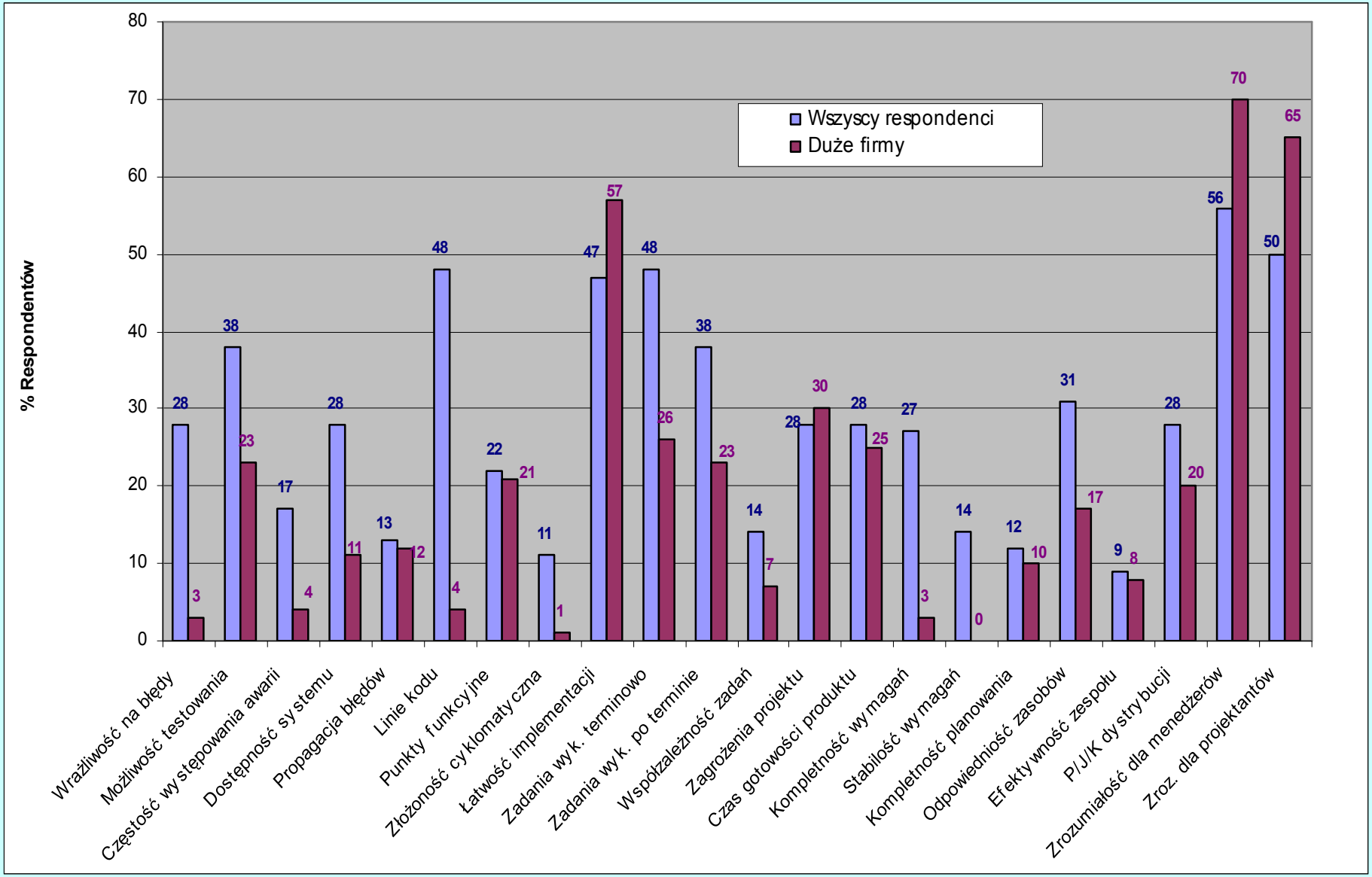
- Metryki procesu wytwarzania

- dojrzałość procesow systemu !
- ocena zarządzanie wytwarzaniem oprogramowania w odniesieniu do cyklu życia oprogramowania

- Metryki zasobów realizacyjnych

- w odniesieniu do personelu „zaangażowanego” w realizację
- narzędzia software’owe, wykorzystywane przy realizacji
- sprzęt, jakim dysponuje wykonawca,
- inne, np. samochody, lokale …..

Wykorzystanie metod estymacyjnych



Podsumowanie

- **Metryki są tworzone i stosowane na bazie doświadczenia i zdrowego rozsądku, co obniża ich wartość dla tzw. „teoretyków informatyki”!**

- Metryki powinny być wykorzystywane jako metody wspomagania ekspertów. Bo metryki stosowane formalistycznie mogą być groźne.
- Najlepiej jest stosować zestawy metryk, co pozwala zmniejszyć błędy pomiarowe.
- Przede wszystkim zdrowy rozsądek i doświadczenie.
- Pomimo pochodzenia empirycznego, metryki skutecznie pomagają w szybkiej i mniej subiektywnej ocenie oprogramowania.
- Specjalizacja metryk w kierunku konkretnej klasy oprogramowania powinna dawać lepsze i bardziej adekwatne oceny niż metryki uniwersalne.
- Wskazane jest wspomaganie metod opartych na metrykach narzędziami programistycznymi.

INŻYNIERIA OPROGRAMOWANIA

Dziękuję za uwagę