

# Wprowadzenie do inżynierii oprogramowania

dr Jarosław Skaruz

<http://ii3.uph.edu.pl/~jareks>

[jaroslaw@skaruz.com](mailto:jaroslaw@skaruz.com)






# Plan wykładu

- ✦ **Przedmiot i zagadnienia inżynierii oprogramowania**
- ✦ **Kryzys oprogramowania**
- ✦ **Złożoność projektu oprogramowania**
- ✦ **Modelowanie pojęciowe**
- ✦ **Pojęcie metodyki; metodyki i notacje**
- ✦ **Modele cyklu życiowego oprogramowania**

# Przedmiot inżynierii oprogramowania

**Inżynieria oprogramowania jest wiedzą techniczną dotyczącą wszystkich faz cyklu życia oprogramowania. Traktuje oprogramowanie jako produkt, który ma spełniać potrzeby techniczne, ekonomiczne lub społeczne.**

**Dobre oprogramowanie powinno być:**

-  zgodne z wymaganiami użytkownika,
-  niezawodne,
-  efektywne,
-  łatwe w konserwacji,
-  ergonomiczne.

Produkcja oprogramowania jest procesem składającym się z wielu faz. Kodowanie (pisanie programów) jest tylko jedną z nich, niekoniecznie najważniejszą.

**Inżynieria oprogramowania jest wiedzą empiryczną, syntezą doświadczenia tysięcy ośrodków zajmujących się budową oprogramowania.**

Praktyka pokazała, że w inżynierii oprogramowania nie ma miejsca stereotyp „od teorii do praktyki”. Teorie, szczególnie zmatematyzowane teorie, okazały się dramatycznie nieskuteczne w praktyce.

# Zagadnienia inżynierii oprogramowania

- ✦ Sposoby prowadzenia przedsięwzięć informatycznych.
- ✦ Techniki planowania, szacowania kosztów, harmonogramowania i monitorowania przedsięwzięć informatycznych.
- ✦ Metody analizy i projektowania systemów.
- ✦ Techniki zwiększania niezawodności oprogramowania.
- ✦ Sposoby testowania systemów i szacowania niezawodności.
- ✦ Sposoby przygotowania dokumentacji technicznej i użytkowej.
- ✦ Procedury kontroli jakości.
- ✦ Metody redukcji kosztów konserwacji (usuwania błędów, modyfikacji i rozszerzeń)
- ✦ Techniki pracy zespołowej i czynniki psychologiczne wpływające na efektywność pracy.

# Kryzys oprogramowania (1)

- ✦ Sprzeczność pomiędzy odpowiedzialnością, jaka spoczywa na współczesnych SI, a ich zawodnością wynikającą ze złożoności i ciągle niedojrzałych metod tworzenia i weryfikacji oprogramowania.
- ✦ Ogromne koszty utrzymania oprogramowania.
- ✦ Niska kultura ponownego użycia wytworzonych komponentów projektów i oprogramowania; niski stopień powtarzalności poszczególnych przedsięwzięć.
- ✦ Długi i kosztowny cykl tworzenia oprogramowania, wysokie prawdopodobieństwo niepowodzenia projektu programistycznego.
- ✦ Długi i kosztowny cykl życia SI, wymagający stałych (często globalnych) zmian.
- ✦ Eklektyczne, niesystematyczne narzędzia i języki programowania.

# Kryzys oprogramowania (2)

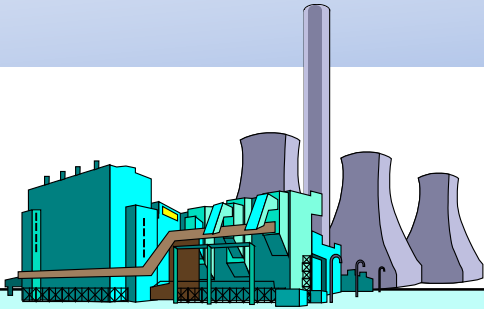
- ✦ Frustracje projektantów oprogramowania i programistów wynikające ze zbyt szybkiego postępu w zakresie języków, narzędzi i metod oraz uciążliwości i długotrwałości procesów produkcji, utrzymania i pielęgnacji oprogramowania.
- ✦ Uzależnienie organizacji od systemów komputerowych i przyjętych technologii przetwarzania informacji, które nie są stabilne w długim horyzoncie czasowym.
- ✦ Problemy współdziałania niezależnie zbudowanego oprogramowania, szczególnie istotne przy dzisiejszych tendencjach integracyjnych.
- ✦ Problemy przystosowania istniejących i działających systemów do nowych wymagań, tendencji i platform sprzętowo-programowych.

# Walka z kryzysem oprogramowania

- ✦ Stosowanie technik i narzędzi ułatwiających pracę nad złożonymi systemami;
- ✦ Korzystanie z metod wspomagających analizę nieznanych problemów oraz ułatwiających wykorzystanie wcześniejszych doświadczeń;
- ✦ Usystematyzowanie procesu wytwarzania oprogramowania, tak aby ułatwić jego planowanie i monitorowanie;
- ✦ Wytworzenie wśród producentów i nabywców przekonania, że budowa dużego systemu wysokiej jakości jest zadaniem wymagającym profesjonalnego podejścia.

**Podstawowym powodem kryzysu oprogramowania jest złożoność produktów informatyki i procesów ich wytwarzania.**

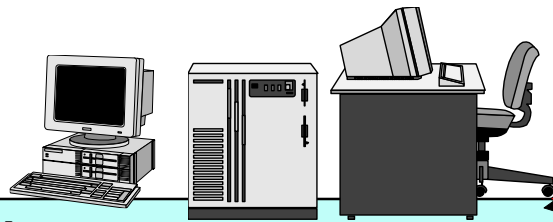
# Źródła złożoności projektu oprogramowania



**Dziedzina problemowa,**  
obejmująca ogromną liczbę  
wzajemnie uzależnionych  
aspektów i problemów.



**Zespół projektantów**  
podlegający ograniczeniom  
pamięci, percepcji, wyrażania  
informacji i komunikacji.



**Środki i technologie  
informatyczne:**  
sprzęt, oprogramowanie, sieć,  
języki, narzędzia, udogodnienia.

**Oprogramowanie:**  
decyzje strategiczne,  
analiza,  
projektowanie,  
konstrukcja,  
dokumentacja,  
wdrożenie,  
szkolenie,  
eksploatacja,  
pielęgnacja,  
modyfikacja.



**Potencjalni użytkownicy:**  
czynniki psychologiczne,  
ergonomia, ograniczenia pamięci  
i percepcji, skłonność do błędów  
i nadużyć, tajność, prywatność.



# Jak walczyć ze złożonością ?



## ***Zasada dekompozycji:***

rozdzielenie złożonego problemu na podproblemy, które można rozpatrywać i rozwiązywać niezależnie od siebie i niezależnie od całości.



## ***Zasada abstrakcji:***

eliminacja, ukrycie lub pominięcie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji; wyodrębnianie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzaniu pojęć lub symboli oznaczających takie cechy.



## ***Zasada ponownego użycia:***

wykorzystanie wcześniej wytworzonych schematów, metod, wzorców, komponentów projektu, komponentów oprogramowania, itd.



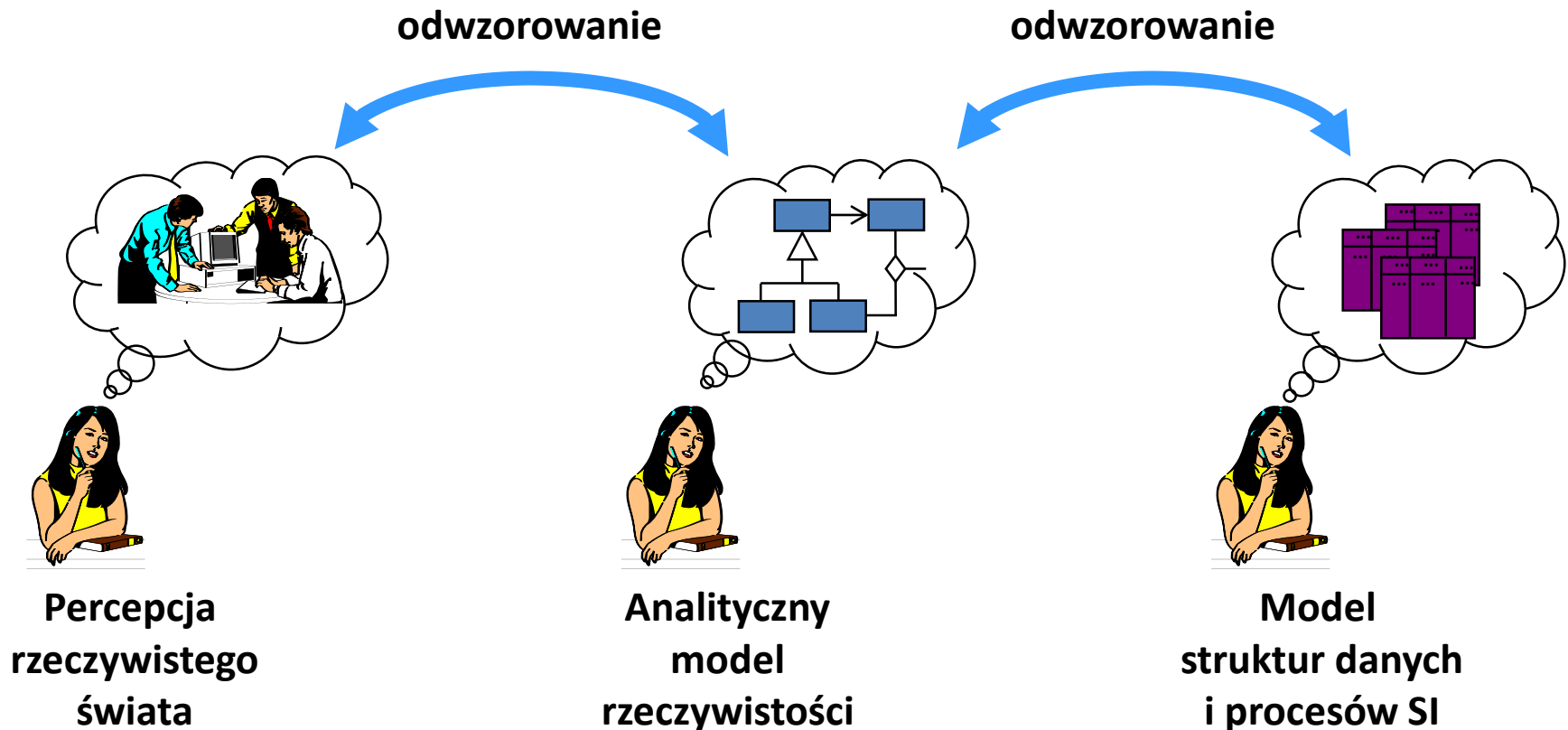
## ***Zasada sprzyjania naturalnym ludzkim własnościom:***

dopasowanie modeli pojęciowych i modeli realizacyjnych systemów do wrodzonych ludzkich własności psychologicznych, instynktów oraz mentalnych mechanizmów percepcji i rozumienia świata.

# Modelowanie pojęciowe

- ✦ Projektant i programista muszą dokładnie wyobrazić sobie problem oraz metodę jego rozwiązania. Zasadnicze procesy tworzenia oprogramowania zachodzą w ludzkim umyśle i nie są związane z jakimkolwiek językiem programowania.
- ✦ Pojęcia *modelowania pojęciowego* (*conceptual modeling*) oraz *modelu pojęciowego* (*conceptual model*) odnoszą się do procesów myślowych i wyobrażeń towarzyszących pracy nad oprogramowaniem.
- ✦ Modelowanie pojęciowe jest wspomagane przez środki wzmacniające ludzką pamięć i wyobraźnię. Służą one do przedstawienia rzeczywistości opisywanej przez dane, procesów zachodzących w rzeczywistości, struktur danych oraz programów składających się na konstrukcję systemu.

# Perspektywy w modelowaniu pojęciowym



Trwałą tendencją w rozwoju metod i narzędzi projektowania oraz konstrukcji SI jest dążenie do minimalizacji luki pomiędzy myśleniem o rzeczywistym problemie a myśleniem o danych i procesach zachodzących na danych.

# Co to jest metodyka (metodologia)?

**Metodyka jest to zestaw pojęć, notacji, modeli, języków, technik i sposobów postępowania** służący do analizy dziedziny stanowiącej przedmiot projektowanego systemu oraz do projektowania pojęciowego, logicznego i/lub fizycznego.

Metodyka jest powiązana z **notacją** służącą do dokumentowania wyników faz projektu (pośrednich, końcowych), jako środek wspomagający ludzką pamięć i wyobraźnię i jako środek komunikacji w zespołach oraz pomiędzy projektantami i klientem.

**Metodyka  
ustala:**

- fazy projektu, role uczestników projektu,
- modele tworzone w każdej z faz,
- scenariusze postępowania w każdej z faz,
- reguły przechodzenia od fazy do następnej fazy,
- notacje, których należy używać,
- dokumentację powstającą w każdej z faz.

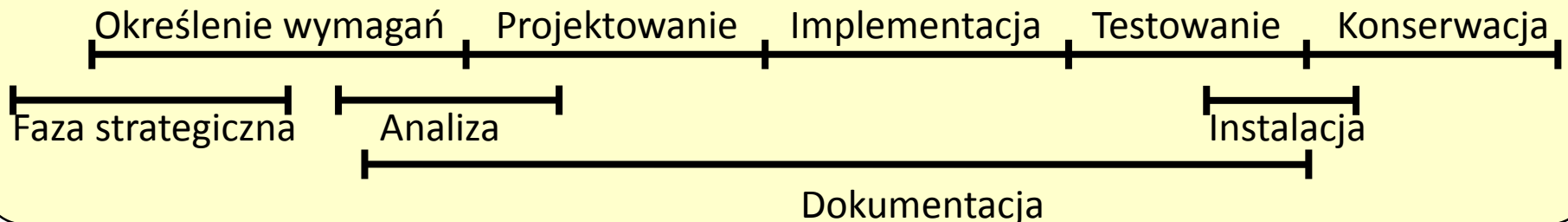
# Cykl życiowy oprogramowania

- ✦ Faza strategiczna: określenie strategicznych celów, planowanie i definicja projektu
- ✦ Określenie wymagań
- ✦ Analiza: dziedziny przedsiębiorczości, wymagań systemowych
- ✦ Projektowanie: projektowanie pojęciowe, projektowanie logiczne
- ✦ Implementacja/konstrukcja: rozwijanie, testowanie, dokumentacja
- ✦ Testowanie
- ✦ Dokumentacja
- ✦ Instalacja
- ✦ Przygotowanie użytkowników, akceptacja, szkolenie
- ✦ Działanie, włączając wspomaganie tworzenia aplikacji
- ✦ Utrzymanie, konserwacja, pielęgnacja

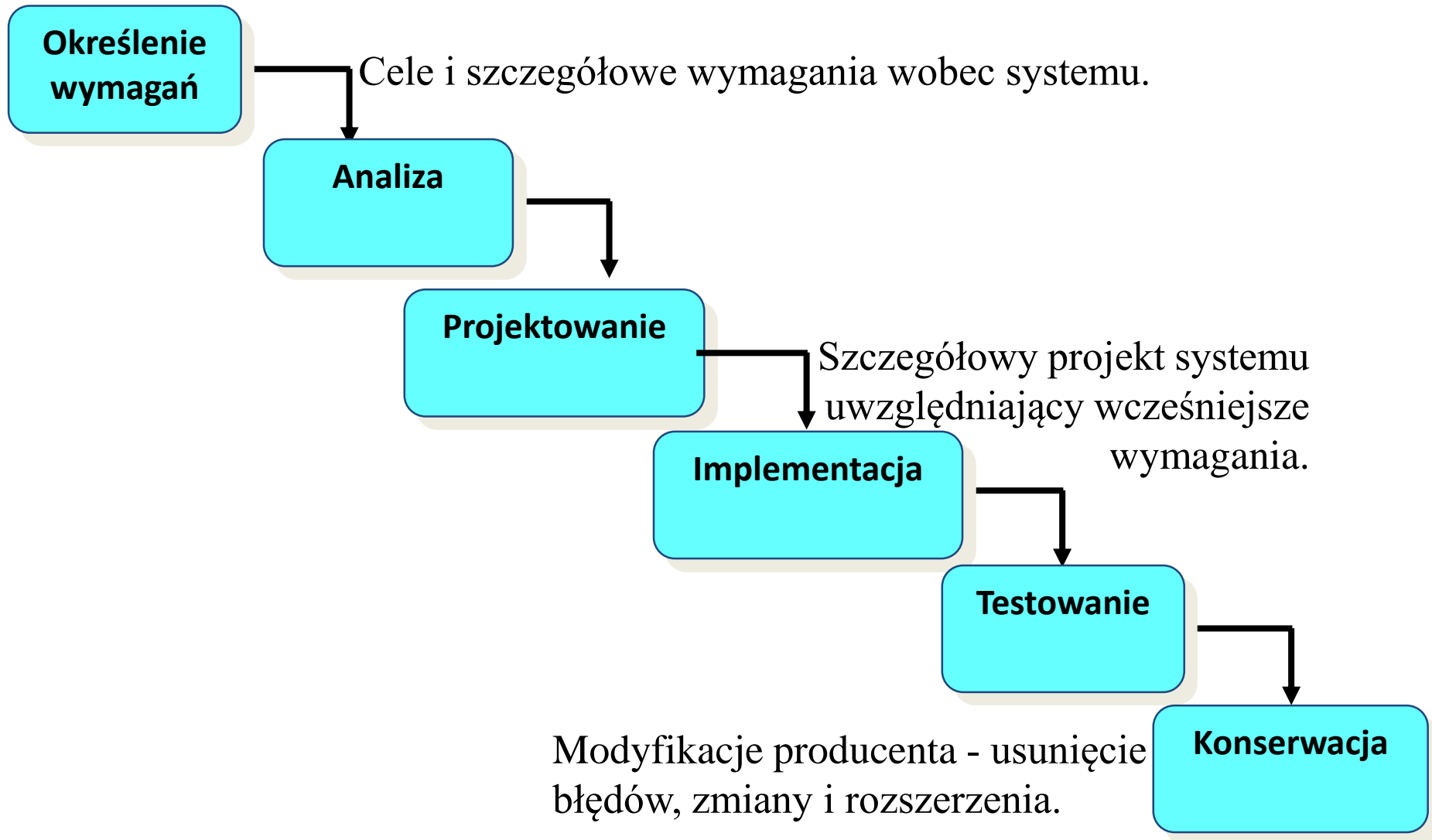
# Modele cyklu życia oprogramowania

- ✦ Model kaskadowy (wodospadowy)
- ✦ Model spiralny
- ✦ Prototypowanie
- ✦ Montaż z gotowych komponentów

Tego rodzaju modeli (oraz ich mutacji) jest bardzo dużo.



# Model kaskadowy (wodospadowy)

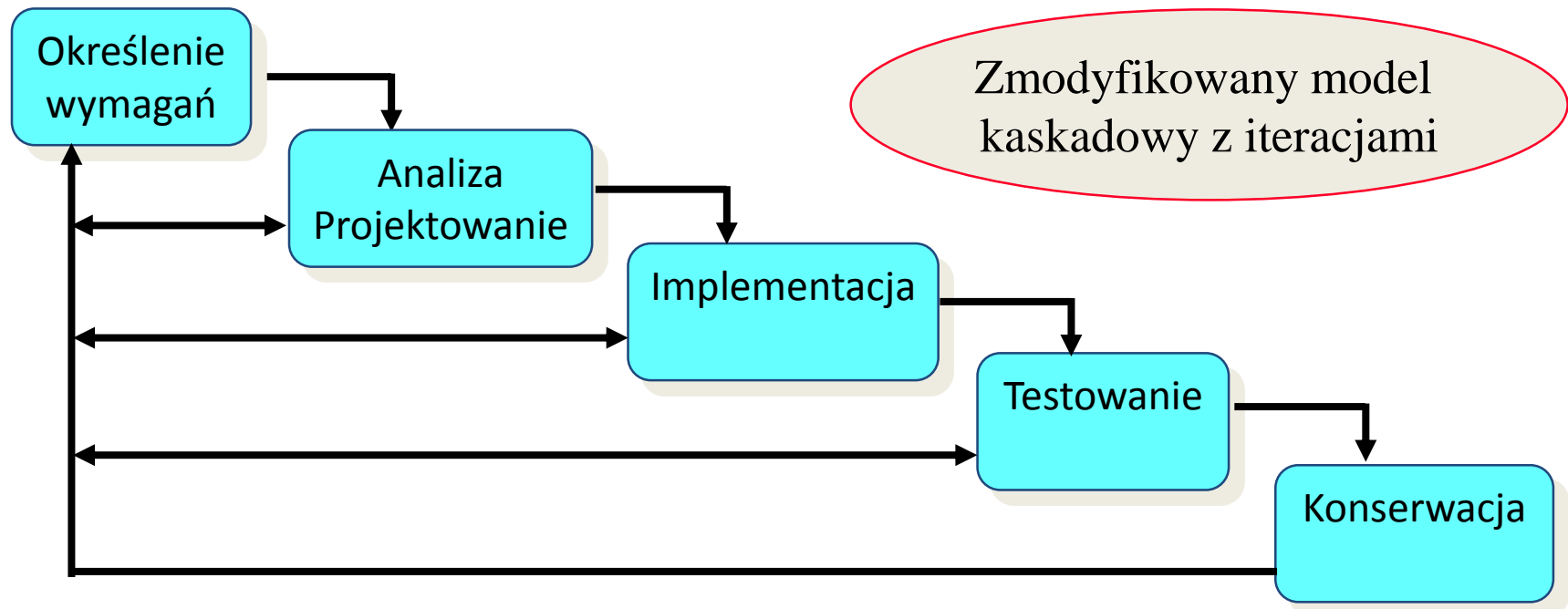


# Ocena modelu kaskadowego

Istnieją zróżnicowane poglądy co do przydatności praktycznej modelu kaskadowego. Podkreślane są następujące wady:

- ☐ Narzucenie twórcom oprogramowania ścisłej kolejności wykonywania prac
- ☐ Wysoki koszt błędów popełnionych we wczesnych fazach
- ☐ Długa przerwa w kontaktach z klientem

Z drugiej strony, jest on do pewnego stopnia niezbędny dla planowania, harmonogramowania, monitorowania i rozliczeń finansowych.





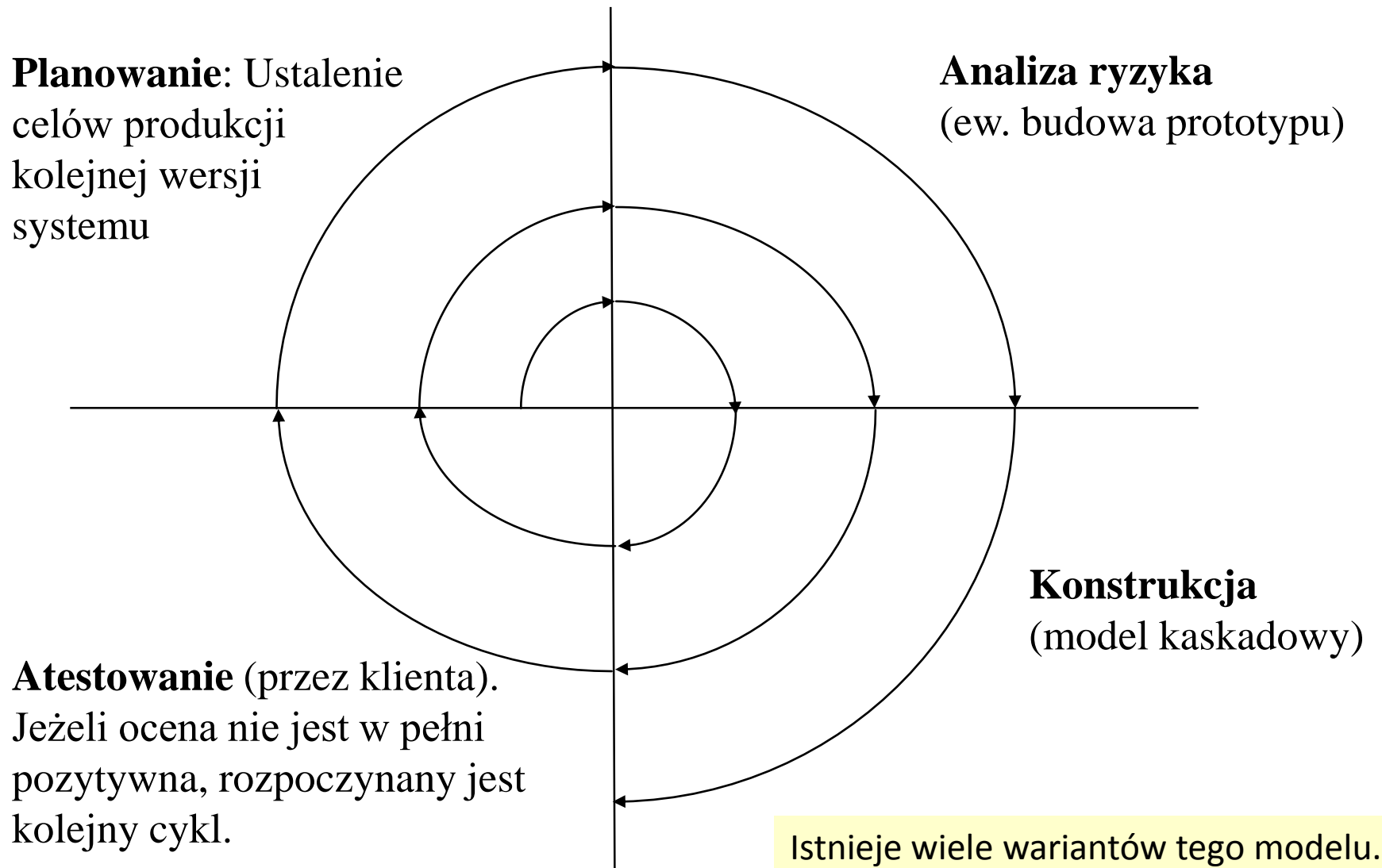
# Realizacja kierowana dokumentami

- ☐ Przyjęty przez armię amerykańską dla realizacji projektów w języku Ada.
- ☐ Jest to odmiana modelu kaskadowego.
- ☐ Każda faza kończy się sporządzeniem szeregu dokumentów, w których opisuje się wyniki danej fazy.
- ☐ Łatwe planowanie, harmonogramowanie oraz monitorowanie przedsięwzięcia.  
Dodatkowa zaleta: (teoretyczna) możliwość realizacji dalszych faz przez inną firmę.

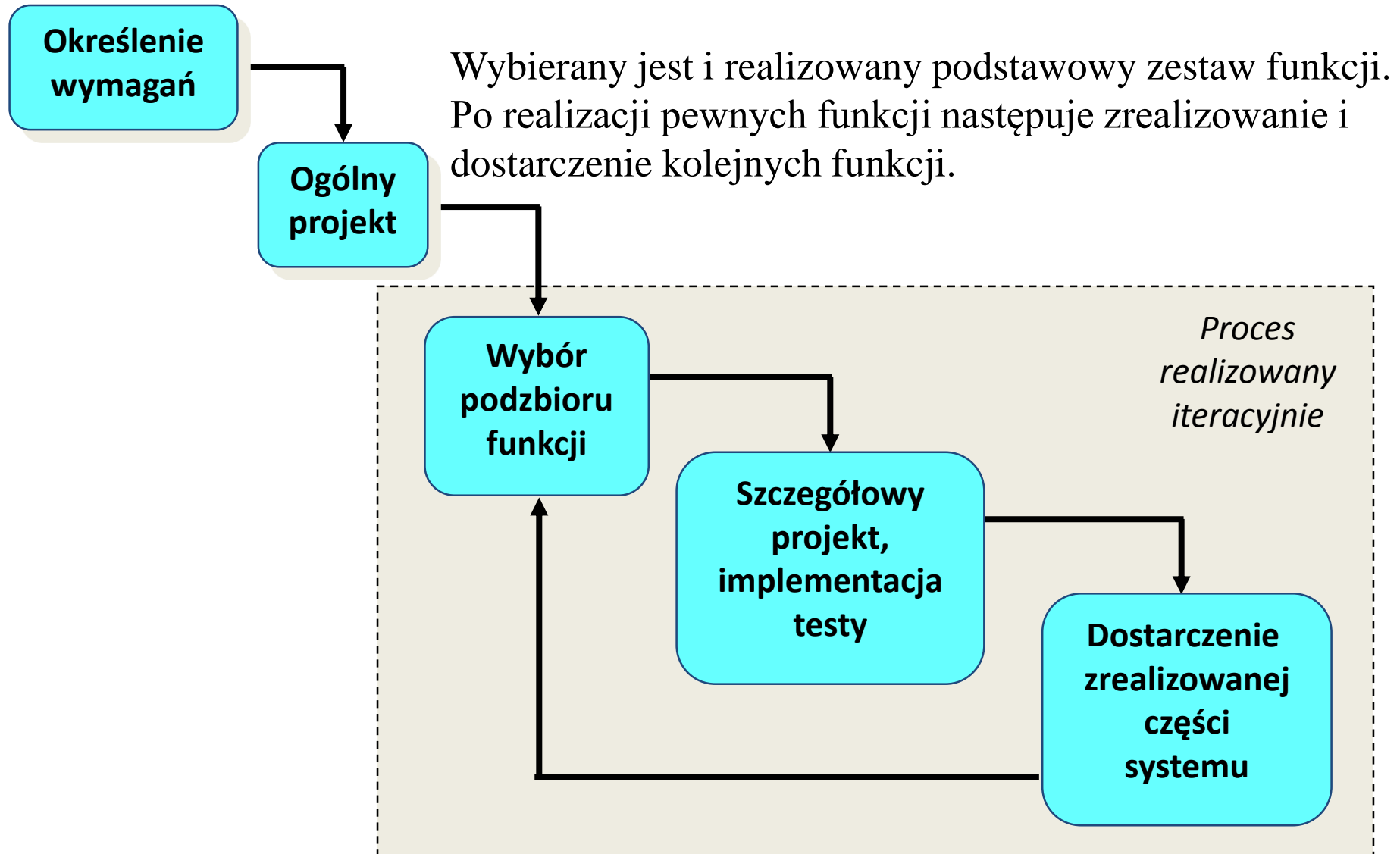
## Wady

- ✦ Duży nakład pracy na opracowanie dokumentów zgodnych ze standardem (DOD STD 2167) - ponad 50% całkowitych nakładów.
- ✦ Przerwy w realizacji niezbędne dla weryfikacji dokumentów przez klienta.

# Model spiralny



# Realizacja przyrostowa (odmiana modelu spiralnego)



# Prototypowanie

**Sposób na uniknięcie zbyt wysokich kosztów błędów popełnionych w fazie określania wymagań.** Zalecany w przypadku, gdy określenie początkowych wymagań jest stosunkowo łatwe.

## Fazy

- ☐ ogólne określenie wymagań
- ☐ budowa prototypu
- ☐ weryfikacja prototypu przez klienta
- ☐ pełne określenie wymagań
- ☐ realizacja pełnego systemu zgodnie z modelem kaskadowym

## Cele

- ☐ wykrycie nieporozumień pomiędzy klientem a twórcami systemu
- ☐ wykrycie brakujących funkcji
- ☐ wykrycie trudnych usług
- ☐ wykrycie braków w specyfikacji wymagań

## Zalety

- ☐ możliwość demonstracji pracującej wersji systemu
- ☐ możliwość szkoleń zanim zbudowany zostanie pełny system

# Metody prototypowania

- ✦ **Niepełna realizacja:** objęcie tylko części funkcji
- ✦ **Języki wysokiego poziomu:** Smalltalk, Lisp, Prolog, 4GL, ...
- ✦ **Wykorzystanie gotowych komponentów**
- ✦ **Generatory interfejsu użytkownika:** wykonywany jest wyłącznie interfejs, wewnątrz systemu jest “podróbka”.
- ✦ **Szybkie programowanie** (*quick-and-dirty*): normalne programowanie, ale bez zwracania uwagi na niektóre jego elementy, np. zaniechanie testowania

Dość często następuje ewolucyjne przejście od prototypu do końcowego systemu. Należy starać się nie dopuścić do sytuacji, aby klient miał wrażenie, że prototyp jest prawie ukończonym produktem. Po fazie prototypowania najlepiej prototyp skierować do archiwum.

# Montaż z gotowych komponentów

Kładzie nacisk na możliwość redukcji nakładów poprzez wykorzystanie podobieństwa tworzonego oprogramowania do wcześniej tworzonych systemów oraz wykorzystanie gotowych komponentów dostępnych na rynku.

Temat jest określany jako **ponowne użycie** (*reuse*)

## Metody

- ☐ zakup elementów ponownego użycia od dostawców
- ☐ przygotowanie elementów poprzednich przedsięwzięć do ponownego użycia

## Zalety

- ☐ wysoka niezawodność
- ☐ zmniejszenie ryzyka
- ☐ efektywne wykorzystanie specjalistów
- ☐ narzucenie standardów

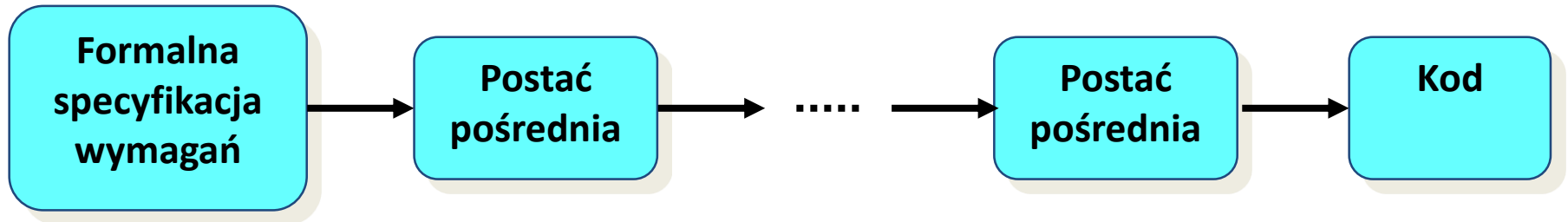
## Wady

- ☐ dodatkowy koszt przygotowania elementów ponownego użycia
- ☐ ryzyko uzależnienia się od dostawcy elementów
- ☐ niedostatki narzędzi wspomagających ten rodzaj pracy.

# Formalne transformacje

Jest on postulowany w ramach tzw. nurtu formalnego w inżynierii oprogramowania.

Wymagania na system są formułowane w pewnym formalnym języku, następnie poddawane są kolejnym transformacjom, aż do uzyskania działającego kodu.



Transformacje są wykonywane bez udziału ludzi (czyli w istocie, język specyfikacji wymagań jest nowym “cudownym” językiem programowania).

**Tego rodzaju pomysły nie sprawdziły się w praktyce.** Nie są znane szersze (lub wręcz *jakiegolwiek*) ich zastosowania. Metody matematyczne nie są w stanie utworzyć pełnej metodyki projektowania, gdyż metodyki włączają wiele elementów (np. psychologicznych) nie podlegających formalnemu traktowaniu. Metody matematyczne mogą jednak wspomagać pewne szczegółowe tematy (tak jak w biologii, ekonomii i innych dziedzinach), np. obliczanie pewnych mierzalnych charakterystyk oprogramowania.

# Szacowanie kosztu oprogramowania

Szacowanie kosztów przeprowadza się dla każdego z alternatywnych rozwiązań.

**Na koszt oprogramowania składają się następujące główne czynniki:**

- ☐ koszt sprzętu będącego częścią tworzonego systemu
- ☐ koszt wyjazdów i szkoleń
- ☐ koszt zakupu narzędzi
- ☐ nakład pracy

Trzy pierwsze czynniki są dość łatwe do oszacowania.

Oszacowanie kosztów oprogramowania jest praktycznie utożsamiane z oszacowaniem nakładu pracy.



# Techniki oszacowania nakładów pracy

- ✦ **Modele algorytmiczne.** Wymagają opisu przedsięwzięcia przez wiele atrybutów liczbowych i/lub opisowych. Odpowiedni algorytm lub formuła matematyczna daje wynik.
- ✦ **Ocena przez eksperta.** Doświadczone osoby z dużą precyzją potrafią oszacować koszt realizacji nowego systemu.
- ✦ **Ocena przez analogię** (historyczna). Wymaga dostępu do informacji o poprzednio realizowanych przedsięwzięciach. Metoda podlega na wyszukaniu przedsięwzięcia o najbardziej zbliżonych charakterystykach do aktualnie rozważanego i znanym koszcie i następnie, oszacowanie ewentualnych różnic.
- ✦ **Wycena dla wygranej.** Koszt oprogramowania jest oszacowany na podstawie kosztu oczekiwanego przez klienta i na podstawie kosztów podawanych przez konkurencję.
- ✦ **Szacowanie wstępujące.** Przedsięwzięcie dzieli się na mniejsze zadania, następnie sumuje się koszt poszczególnych zadań.

# Algorytmiczne modele szacowania kosztów

Historycznie, podstawą oszacowania jest rozmiar systemu liczony w liniach kodu źródłowego. Metody takie są niedokładne, zawodne, sprzyjające patologiom, np. sztuczemu pomnażaniu ilości linii, ignorowaniu komentarzy, itp.

Obecnie stosuje się wiele miar o lepszych charakterystykach (z których będą omówione punkty funkcyjne). Miary te, jakkolwiek niedokładne i oparte na szacunkach, są jednak konieczne. Niemożliwe jest jakiekolwiek planowania bez oszacowania kosztów. Miary dotyczą także innych cech projektu i oprogramowania, np. czasu wykonania, jakości, niezawodności, itd.

Jest bardzo istotne uwolnienie się od religijnego stosunku do miar, tj. traktowanie ich jako obiektywnych wartości “policzonych przez komputer”. Podstawą wszystkich miar są szacunki, które mogą być obarczone znacznym błędem, nierzadko o rząd wielkości. Miary należy traktować jako latarnię morską we mgle - może ona nas naprowadzić na dobry kierunek, może ostrzec przed niebezpieczeństwem. Obowiązuje zasada patrzenia na ten sam problem z wielu punktów widzenia (wiele różnych miar) i zdrowy rozsądek.

# Metoda szacowania kosztów COCOMO

COCOMO jest oparte na kilku formułach pozwalających oszacować całkowity koszt przedsięwzięcia na podstawie oszacowanej liczby linii kodu. Jest to główna słabość tej metody, gdyż:

- ✦ liczba ta staje się przewidywalna dopiero wtedy, gdy kończy się faza projektowania architektury systemu; jest to za późno;
- ✦ pojęcie “linii kodu” zależy od języka programowania i przyjętych konwencji;
- ✦ pojęcie “linii kodu” nie ma zastosowania do nowoczesnych technik programistycznych, np. programowania wizyjnego.

COCOMO oferuje kilka metod określanych jako podstawowa, pośrednia i detaliczna.

- ✦ **Metoda podstawowa:** prosta formuła dla oceny osobo-miesięcy oraz czasu potrzebnego na całość projektu.
- ✦ **Metoda pośrednia:** modyfikuje wyniki osiągnięte przez metodę podstawową poprzez odpowiednie czynniki, które zależą od aspektów złożoności.
- ✦ **Metoda detaliczna:** bardziej skomplikowana, ale jak się okazało, nie dostarcza lepszych wyników niż metoda pośrednia.

# Metoda punktów funkcyjnych

**Metoda punktów funkcyjnych oszacowuje koszt projektu na podstawie funkcji użytkowych, które system ma realizować.** Stąd wynika, że metoda ta może być stosowana dopiero wtedy, gdy funkcje te są z grubsza znane.

Metoda jest oparta na zliczaniu ilości wejść i wyjść systemu, miejsc przechowywania danych i innych kryteriów. Te dane są następnie mnożone przez zadane z góry wagi i sumowane. Rezultatem jest liczba „punktów funkcyjnych”.

Punkty funkcyjne mogą być następnie modyfikowane zależnie od dodatkowych czynników złożoności oprogramowania.

Istnieją przeliczniki punktów funkcyjnych na liczbę linii kodu, co może być podstawą dla metody COCOMO.

Metoda jest szeroko stosowana i posiada stosunkowo mało wad. Niemniej, istnieje wiele innych, mniej popularnych metod, posiadających swoich zwolenników.

# Metoda Delphi i inne metody

✧ **Metoda Delphi** zakłada użycie kilku niezależnych ekspertów, którzy nie mogą się ze sobą w tej sprawie komunikować i naradzać. Każdy z nich szacuje koszty i nakłady na podstawie własnych doświadczeń i metod. Eksperti są anonimowi. Każdy z nich uzasadnia przedstawione wyniki.

Koordynator metody zbiera wyniki od ekspertów. Jeżeli znacznie się różnią, wówczas tworzy pewne sumaryczne zestawienie (np. średnią) i wysyła do ekspertów dla ponownego oszacowania. Cykl jest powtarzany aż do uzyskania pewnej zgody pomiędzy ekspertami.

✧ **Metoda analizy podziału aktywności** (*activity distribution analysis*): Projekt dzieli się na aktywności, które są znane z poprzednich projektów. Następnie dla każdej z planowanych aktywności ustala się, na ile będzie ona bardziej (lub mniej) pracochłonna od aktywności już wykonanej, której koszt/nakład jest znany. Daje to szacunek dla każdej planowanej aktywności. Szacunki sumuje się dla uzyskania całościowego oszacowania.

✧ **Metody oszacowania pracochłonności testowania systemu**

✧ **Metody oszacowania pracochłonności dokumentacji**

✧ **Metody oszacowania obciążenia sieci**

....

# Podsumowanie: kluczowe czynniki sukcesu



**Szybkość pracy.** Szczególnie w przypadku firm realizujących oprogramowanie na zamówienie, opóźnienia w przeprowadzeniu fazy strategicznej mogą zaprzepaścić szansę na wygranę przetargu lub na następne zamówienie. Faza ta wymaga więc stosunkowo niedużej liczby osób, które potrafią wykonać pracę w krótkim czasie.



**Zaangażowanie kluczowych osób ze strony klienta.** Brak akceptacji dla sposobu realizacji przedsięwzięcia ze strony kluczowych osób po stronie klienta może uniemożliwić jego przyszły sukces.



**Uchwycenie (ogólne) całości systemu.** Podstawowym błędem popełnianym w fazie strategicznej jest zbytne przywiązanie i koncentracja na pewnych fragmentach systemu. Niemożliwe jest w tej sytuacji oszacowanie kosztów wykonania całości. Łatwo jest też przeoczyć szczególnie trudne fragmenty systemu.