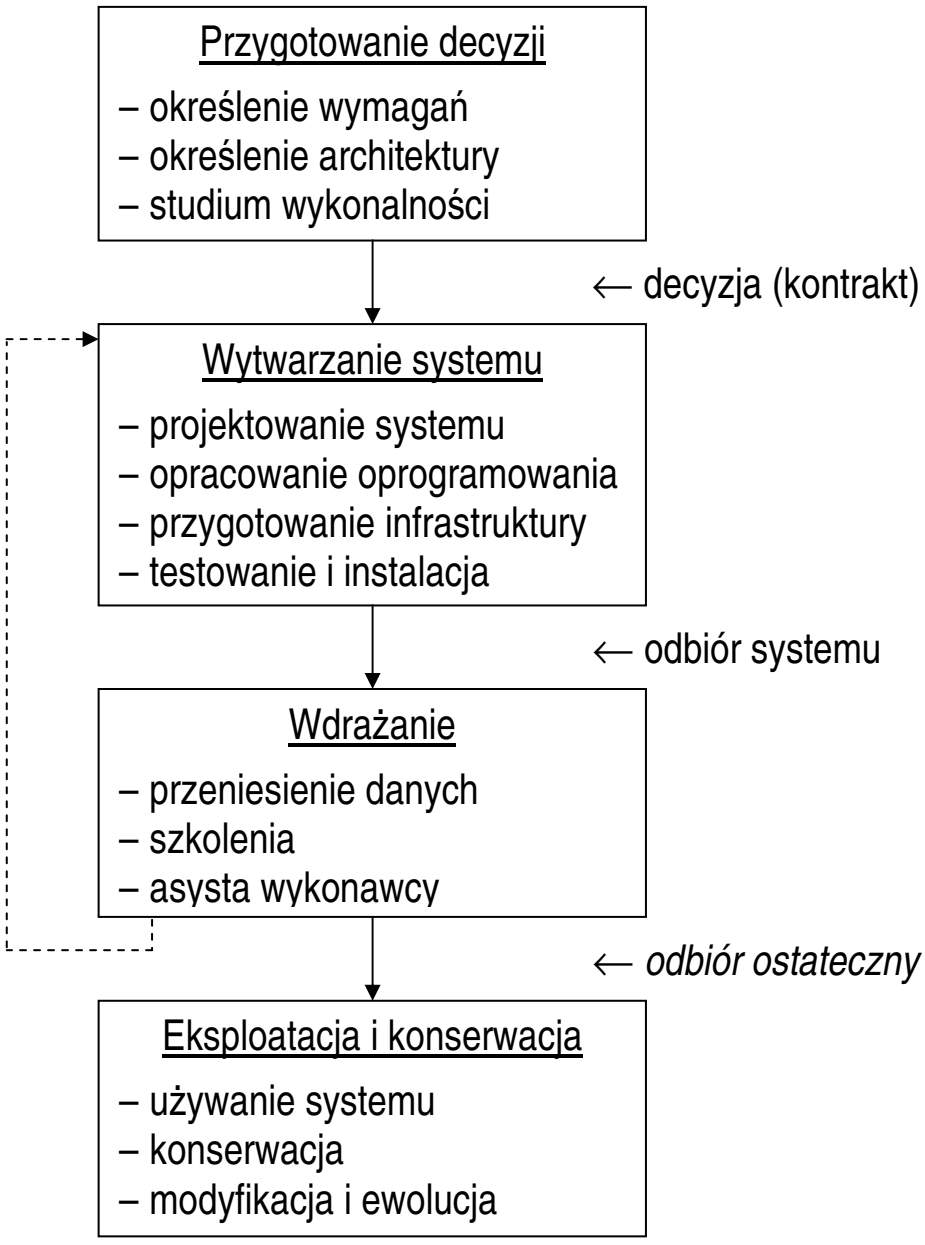
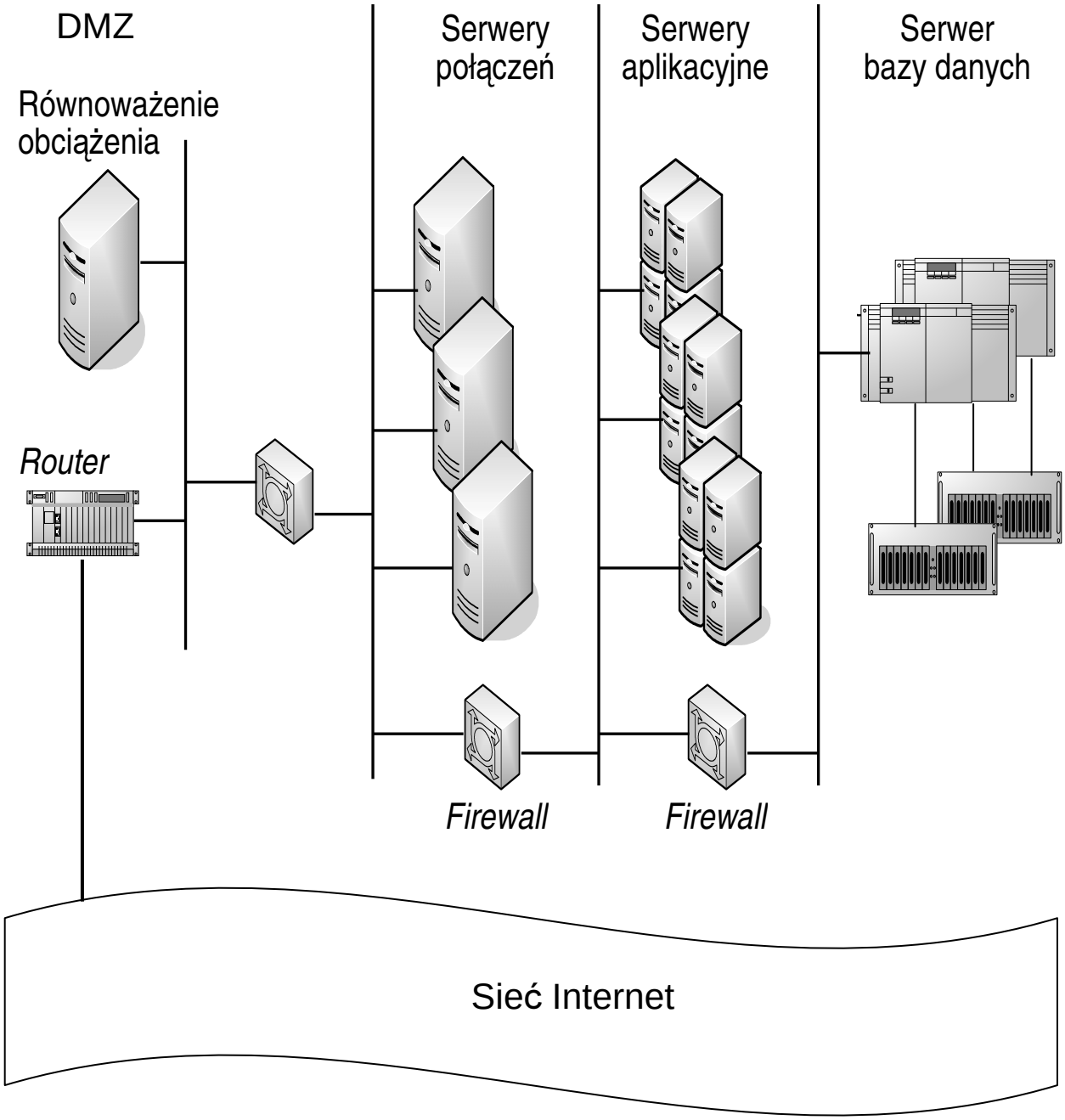
Modele procesów projektowych

Wytwarzanie systemu (*system lifecycle*)
Wytwarzanie oprogramowania (*software lifecycle*)

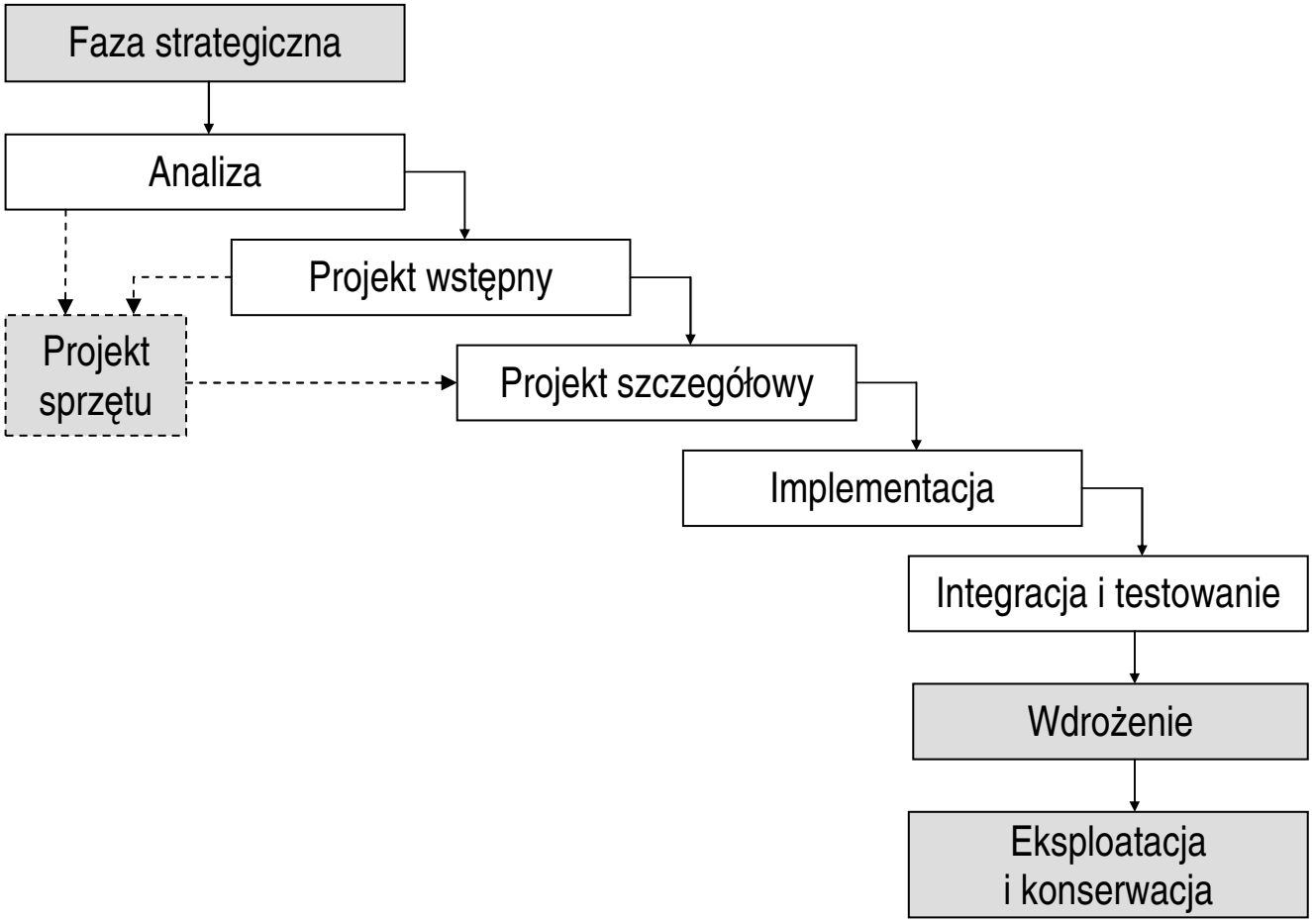
Cykl wytwarzania systemu informatycznego



Przykładowa infrastruktura IT



Kaskadowy model opracowania oprogramowania (waterfall model)



Punkty kontrolne (*milestones*)

- każda faza kończy się zestawieniem produktów i odbiorem,
- inne punkty warunkujące kontynuację prac.

Rozkład kosztów

- Opracowanie oprogramowania
  - Analiza — 15%
  - Projekt — 20%
  - Implementacja — 20%
  - Testowanie — 45%
  - Konserwacja — 70 ... 200%
- Konserwacja
  - Usuwanie błędów — 20%
  - Zmiana wymagań — 80%
- Usuwanie błędów
  - Błędy analizy i projektu — 80%
  - Błędy implementacji — 20%

Zalety

- dyscyplina i uporządkowanie działań,
- kontrola konfiguracji przez zestawienie produktów fazy,
- kontrola jakości przez weryfikację produktów każdej fazy.

Wady

- podejmowanie najważniejszych decyzji na początku projektu,
- późna ocena systemu,
- brak środków do kontroli zmian i ryzyka.

Ocena

- model często stosowany;
  działa dobrze przy stabilnych wymaganiach.

Szybkie makietowanie (rapid prototyping)

Opracowanie prototypowej wersji systemu
i przedstawienie użytkownikowi działającego programu.

Prototyp

- nie wykonuje wszystkich funkcji,
- nie osiąga żądanej wydajności.

Zalety

- zwiększenie wiarygodności specyfikacji wymagań,
- poznanie dziedziny i wykrycie potencjalnych trudności,
- możliwość porównania różnych wariantów.

Wady

- wydłużenie i pewien wzrost kosztu fazy analizy.

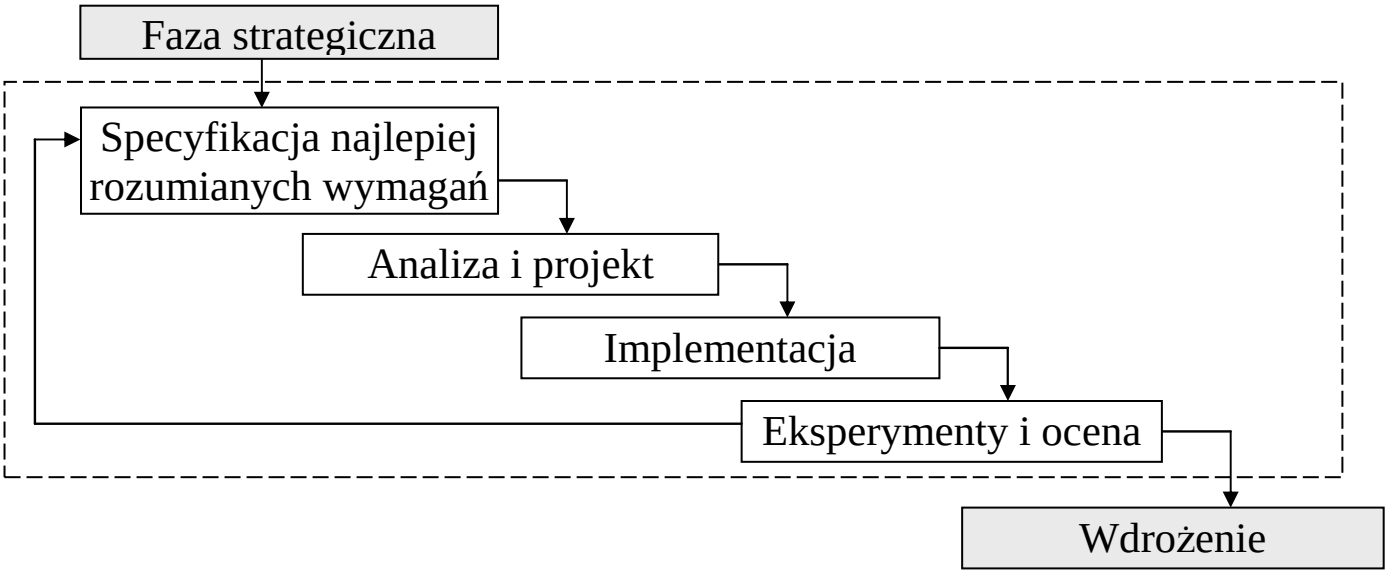
Ocena

- technika zalecana i szeroko stosowana.

Proces przyrostowy

Mniej systematyczne postępowanie, na przemian: w szerz i w głąb.

a) Programowanie odkrywcze (*exploratory programming*)



Zalety

- koncentracja prac na znanych fragmentach i przechodzenie dalej po zebraniu doświadczeń jest zgodne z psychologią człowieka,
- duży udział użytkownika,
- szybka dostawa działającej wersji (szkolenie, eksploatacja próbna).

Wady

- trudna kontrola postępów prac,
- marna struktura finalnego oprogramowania,
- konieczne narzędzia makietowania i automatycznej generacji kodu,
- niedopasowanie do procedury pozyskiwania.

Ocena:

- metoda działa dla systemów małych i nie przewidywanych do długotrwałej eksploatacji, do systemów dużych się nie nadaje.

b) Programowanie ekstremalne (*extreme programming*)

1. Określenie najważniejszych wymagań (horyzont ≤ miesiąc).
2. Uzgodnienie architektury na poziomie *metafory* systemu.
3. Pisanie testów → programowanie → integracja (cykl ≤ dzień)
   (kod wspólny, refaktoryzacja, brak nadgodzin, praca w parach)
4. Eksperymenty z wydaniem i ocena użytkownika

Opracowanie programów przez programowanie — metodyka lekka.

Zalety

- stały udział użytkownika,
- brak wydatków na dokładny projekt i dokumentację,
- jednoznaczna dokumentacja wymagań przez testy,
- szybka dostawa działającej wersji.

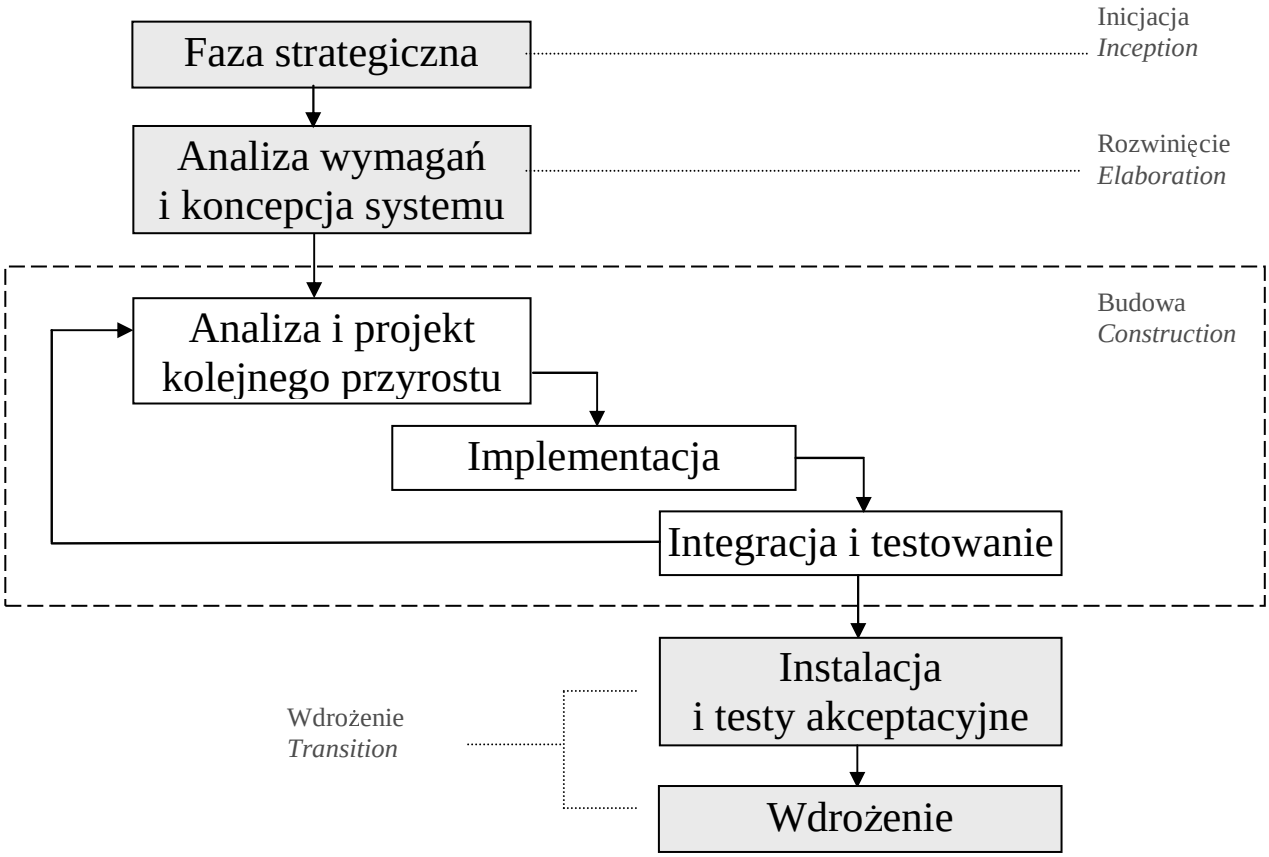
Wady

- trudna kontrola postępów prac,
- brak dokumentacji bardzo utrudniający konserwację,
- konieczne zaawansowane narzędzia integracji i testowania,
- niedopasowanie do procedury pozyskiwania.

Ocena

- metoda działa dla systemów małych i nie przewidywanych do długotrwałej eksploatacji, do systemów dużych się nie nadaje.

c) Model przyrostowy iteracyjny (*iterative model*)



Analiza całości. Koncepcja obejmuje określenie przyrostów.

Zalety

- duży udział użytkownika,
- szybka dostawa działającej wersji (szkolenie, eksploatacja).

Wady:

- nowe fragmenty mogą nie pasować do już istniejących,
- dodatkowy koszt integracji kolejno realizowanych fragmentów,
- trudniejsze zarządzanie niż w modelu kaskadowym.

Ocena:

- stosowana też w dużych projektach, zwłaszcza obiektowych,
- konieczne staranne zarządzanie i dobra koncepcja całości.

Formalne konstruowanie programów

Formalny (matematyczny) zapis specyfikacji.
Iteracyjne rozbijanie kolejnych formuł na coraz prostsze,
aż do formuł realizowanych przez instrukcji języka programowania,
matematyczne dowodzenie poprawności każdego kroku.

Zalety

- udowodniona poprawność programu,
- eliminacja procesu testowania.

Wady

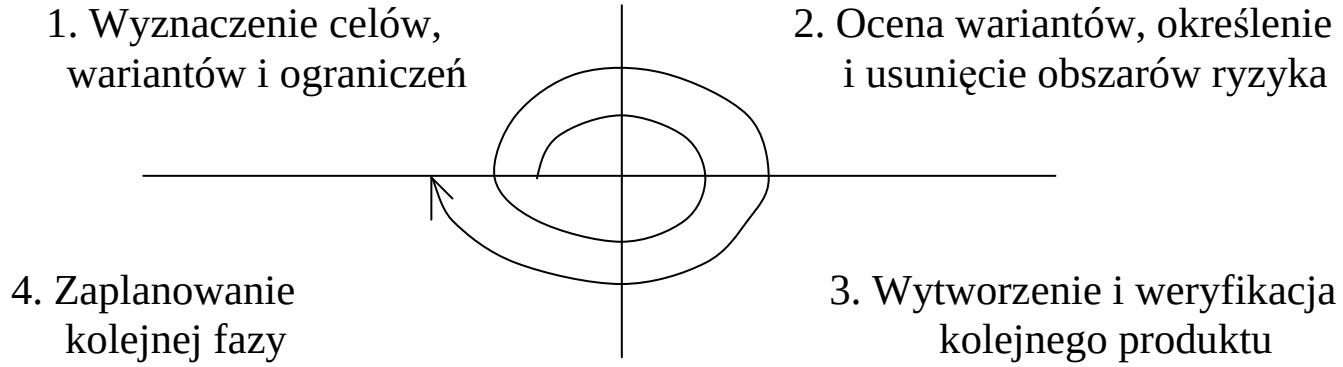
- nieczytelność specyfikacji i trudność oceny (makietowanie ?),
- trudność konstrukcji programu i dowodu poprawności
- trudność stosowania w dziedzinach bez naturalnego modelu matematycznego

Ocena

- na razie technika niemal nie stosowana w praktyce (metoda VDM),
- wykorzystywana czasami do badania elementów poprawności,
- może mieć rosnące znaczenie w przyszłości:
  → po automatyzacji dowodzenia,
  → do wielokrotnie używanych bloków programowych.

Proces spiralny

Proces opracowania posuwa się naprzód po linii spiralnej.
Każdy obrót reprezentuje planowanie i wykonanie pewnego produktu (np. specyfikacji lub modułu).



W ten wzorzec można wtłoczyć dowolny proces projektowy — obrót spirali może odpowiadać fazie procesu kaskadowego albo przyrostowi modelu przyrostowego. Wartością jest zaakcentowanie roli ryzyka.

Przykładowy przebieg projektu:

0 Wstępne wymagania użytkownika

1.1 Ocena wymagań, analiza problemów, ustalenie zakresu systemu, ograniczeń i uwarunkowań. Określenie wariantów

1.2 Analiza wariantów, określenie czynników ryzyka (np.: opóźnienie realizacji, błędy obsługi itp. oraz rozwiązań awaryjnych.

1.3 **Projekt koncepcyjny**.

1.4 Określenie planu opracowania specyfikacji wymagań.

2.1 Zbieranie danych o potrzebach użytkownika, porządkowanie danych, definiowanie celów projektu.

2.2 Analiza ryzyka: określenie obszarów zagrożenia i poszukiwanie strategii rozwiązania — modelowanie, makietowanie, analizę itp.

2.3 **Specyfikacja wymagań**.

2.4 Określenie planu rozwoju systemu

3.1 Analiza i ocena wariantów realizacji (np. projekt, użycie gotowego, zakup), analiza wymagań i ograniczeń.

3.2 Analiza ryzyka różnych wariantów, makietowanie.

3.3 **Projekt wstępny**.

3.4 Określenie **planu integracji i testowania**.

4.1 Analiza wymagań i ograniczeń (wydajność, modyfikowalność, dopuszczalny koszt i termin, wymagane sprzęgi itp.).

4.2 Analiza ryzyka, budowa działającego prototypu.

4.3 **Produkcja programów**: projekt, implementacja, testowanie, opracowanie dokumentacji użytkowej.

4.4 Określenie planu wdrożenia systemu.

5.1 Analiza uwarunkowań wdrożeniowych.

5.2 Analiza ryzyka zakłóceń pracy przedsiębiorstwa, wypracowanie środków zaradczych i działań awaryjnych.

5.3 Instalacja w środowisku docelowym, **testowanie akceptacyjne**, szkolenie, przeniesienie danych, wdrożenie i wstępna akceptacja.

5.4 **Eksploatacja systemu**. Finalna akceptacja, planowanie rozwoju.

.....

Potrzeby rynku pracy (badanie Uniwersytetu Szczecińskiego, 2000r)

- **Braki wykształcenia absolwentów (wg. pracodawców)**

Ważność	Wiedza teoretyczna	Umiejętność analizy	Umiejętność projektowania	Umiejętność programowania
duża	13	60	44	11
mała	17	14	16	18
żadna	70	26	40	70

- **Rodzaj prac w firmach informatycznych**

Rodzaj	%firm	%czasu	%	%
Analiza	60	12	7	17
Projektowanie	74	13	10	
Programowanie	75	23	17	54
Wdrażanie	81	30	24	
Administracja	72	18	13	
Inne			29	

- **Czynniki uwzględniane przy rekrutacji pracowników**

Ważność	Umiejętności techniczne	Wykształcenie	Zagadnienia ekonomiczne	Doświadczenie	Inne
duża	94	82	44	86	67
mała	0	4	15	2	15
żadna	6	14	41	12	18

Inne = cechy osobowości + dyspozycyjność

Weryfikacja i ocena oprogramowania

Każdy program powinien być poprawny. Wymagania zależą jednak od charakteru zastosowania.

- Najwyższe, gdy może dojść do zagrożenia życia lub zdrowia ludzi
  → niedopuszczalne nawet pojedyncze błędy,
- Niższe jeśli błędy lub awarie mogą powodować straty materialne
  → często ważne przede wszystkim bezpieczeństwo danych.
- Najniższe gdy działanie programu jest weryfikowane przez człowieka
  → w razie błędu może być powtórzone.

Program musi działać poprawnie w stopniu akceptowalnym w danej dziedzinie zastosowania.

Działania kontrolne:

- **Weryfikacja** (*verification*). Sprawdzenie poprawności wyników etapu projektu, względem wyników poprzedniego etapu, np.:
  – czy projekt zapewnia spełnienie specyfikacji wymagań,
  – czy implementacja realizuje ustalenia projektu, itp.
- **Ocena** (*validation*). Sprawdzenie poprawność implementacji względem wymagań.

Metody weryfikacji i oceny

Metody weryfikacji zależą od postaci weryfikowanych wyników:

– dokumentację analityczną i projektową można analizować pod względem postaci formalnej i zawartości merytorycznej,

– wykonalny program można testować w działaniu.

- **Metody weryfikacji:**

1. Testowanie
   – stosowalne tylko do wykonalnej postaci programu (lub makiety),
   – nie gwarantuje poprawności,
   – kosztowne (zajmuje dużo czasu, angażuje ludzi),
   – spóźnione (kosztowne usuwanie błędów projektu i analizy).
2. Inspekcje (*inspection*, *walkthrough*) i przeglądy (*review*)
   – można stosować w każdym etapie projektu,
   – poważnie zredukuje liczbę błędów (30% – *IBM code inspection*).
3. Dowody poprawności
   – brak dojrzałych metod (metoda konstrukcji VDM),
   – wymaga szczególnych kwalifikacji wykonawców,
   – dowód jest zwykle długi i też może zawierać błędy.
   W praktyce: dowody częściowe lub przy super wymaganiach.
4. Odtwarzanie specyfikacji (*diverse back translation*)
   – tylko ocena finalna,
   – kosztowne i długotrwałe.
   Zweryfikowane eksperymentalnie, ale nie stosowane praktycznie.

- **Certyfikacja oprogramowania**

Urzędowe dopuszczenie systemu do eksploatacji po wykazaniu, że poziom niezawodności spełnia wymagania dziedziny aplikacji

– matematyczny dowód poprawności,
– odtworzenie specyfikacji,
– probabilistyczny model procesu testowania.

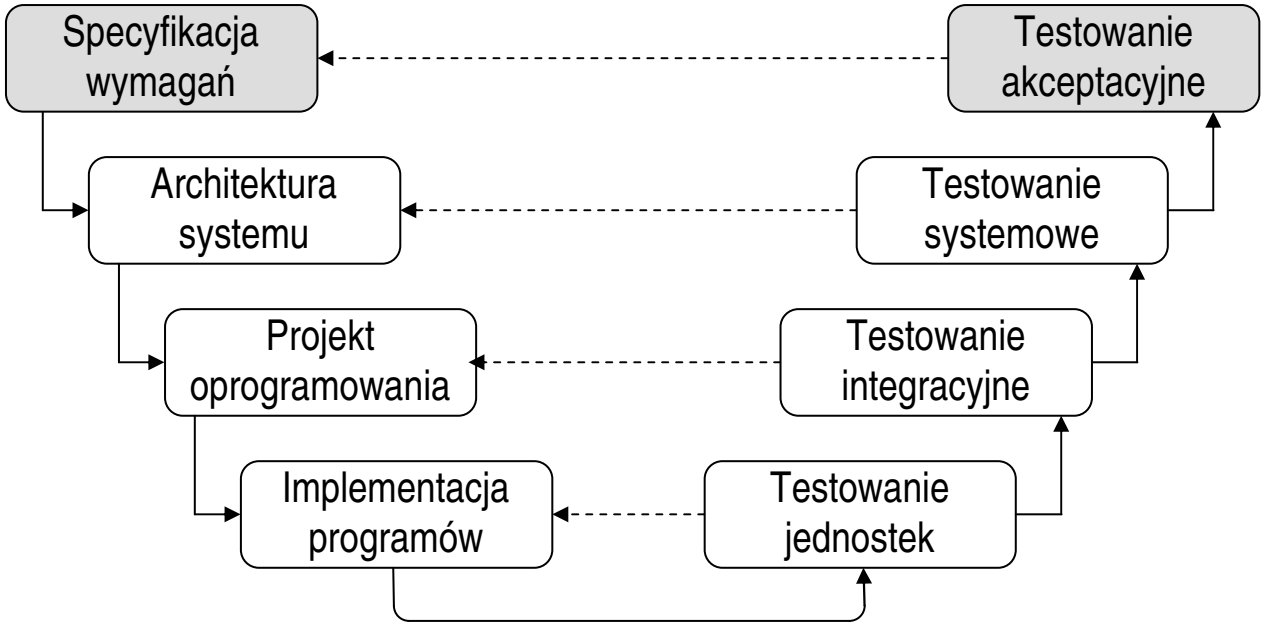
- **Projekty komercyjne**

Finalnym celem jest zadowolenie klienta (czasem specyfikacja wymagań nie do końca sprecyzowana)

– udział klienta we wszystkie etapy projektu (wczesna ocenę),
– testowanie akceptacyjne.

→ *Podstawową techniką weryfikacji i oceny jest obecnie testowanie.*

Proces testowania



unit testing
module testing
subsystem testing
interface testing
acceptance testing

- **Plan testowania (*test plan*)**
  - odpowiedzialność
  - zasoby (czas, ludzie, sprzęt)
  - harmonogram
  - kryteria akceptacji
  - scenariusze testowania
  - przypadki testowe
  - dane testowe (oczekiwane wyniki)

Ocena wyników testowania (czy kompletne? wiarygodne?)

- Miary pokrycia
  - bloków (sekwencji instrukcji)
  - rozejść decyzyjnych (warunkowych) w programie
  - różnych dróg wykonania
  - użycia zmiennych (*computational variables use*)
  - użycia zmiennych w wyrażeniach decyzyjnych
  - dziedziny danych wejściowych

  Wymagają znajomości budowy programu (*white box testing*).
  Można wykorzystać do oceny postępów testowania.

- Bezpośredni pomiar metodą zasiewów (mutacje programów)

  N_s: liczba wprowadzonych błędów
  n_s: liczba wykrytych błędów wprowadzonych
  N_p: liczba błędów prawdziwych w programie
  n_p: liczba wykrytych błędów prawdziwych

  $$N_p = n_p \times N_s / n_s$$

- Odwołanie do specyfikacji
  - pokazanie wszystkich komunikatów o błędach (i braku błędów),
  - przetestowania wszystkich wyodrębnionych wymagań.

  Nie wymaga znajomości budowy programu (*black box testing*).

Testowanie probabilistyczne

Liczba wykrywanych błędów maleje z czasem testowania

→ buduje się statystyczny model tego procesu i wyznacza rozkład zmiennej losowej: *czas do wykrycia następnego błędu*,

Wartość oczekiwana tego czasu:

– określa spodziewaną długość procesu testowania
– można interpretować jako „średni czas międzyawaryjny”

Testowanie regresyjne

– maskowanie błędów,
– nowe błędy po zmianach.

Środowiska wspomagające testowanie

- Generacja danych testowych, rejestracja wyników.
- Wykorzystanie danych projektowych (diagramy stanów obiektów, diagramy sekwencji) do obserwacji zachowania programu:
  – metoda sekwencji: weryfikacja sekwencji komunikatów odbieranych i wysyłanych przez obiekty,
  – metoda wektora stanów: obserwacja i weryfikacja stanu obiektów podczas wykonania.

Plan testów akceptacyjnych

- procedura testowania
- scenariusze testowania

- **Scenariusz testowania** (*test scenario*)
  - opisuje testowanie wydzielonej funkcjonalności
  - składa się z kroków wykonania przypadków testowych
- **Przypadek testowy** (*test case*)
  - opisuje testowanie konkretnej funkcji
  - określa dane wejściowe, spodziewane rezultaty i kryteria oceny
  - składa się z kroków odpowiadających prostym czynnościom
- **Zestaw danych testowych**
  - komplet danych dla wykonania przypadku testowego
- **Krok przypadku testowego**
  - opisuje konkretną czynność

Przykład — system IACS

• Procedura testowa

	Nazwa	Naliczanie płatności obszarowych
		Procedura rozpoczyna się w stanie .....
1	S-01	Wprowadzanie, kontrola i zatwierdzanie danych w sprawie obsługi wniosków obszarowych
2	S-02	Weryfikacja wniosków obszarowych po kontroli na miejscu
3	S-03	Naliczanie płatności obszarowych

• Scenariusz testowania

	Identyfikator	S-01	
	Nazwa	Wprowadzanie, kontrola i zatwierdzanie danych w sprawie obsługi wniosków obszarowych	
1	P-01	Utworzenie sprawy z przyjęciem dokumentu	Z-01, Z-02
2	P-02	Wprowadzenie danych wniosku przez Operatora	Z-01, Z-02
3	P-03	Zatwierdzenie danych wniosku przez Kontrolera	
4	P-04	Przeprowadzenie kontroli administracyjnej	

• Przypadek testowy

	Identyfikator	P-01
	Nazwa	Utworzenie sprawy z przyjęciem dokumentu
1	Wybranie funkcji *Przekazywanie dokumentów – przyjęcie*	
2	Zaznaczenie pola *Wniosek o płatności obszarowe*	
3	Uruchomienie funkcji *Sprawy – Utworzenie sprawy*	
4	Wybranie z listy dokumentu inicjującego sprawę	
5	Wprowadzenie danych strony sprawy (dane nadawcy)	
6	Wybranie opcji – *Zatwierdzenie danych*	

Przygotowanie zbiorów danych testowych

- Zalecenia:
  - liczba zestawów danych określa wiarygodność testu,
  - należy zapewnić pokrycie całego zakresu danych,
  - należy wprowadzać dane poprawne i niepoprawne.
- Generacja danych testowych:
  - metody deterministyczne (pokrycie równomierne lub wg. profilu operacyjnego),
  - metody losowe (generator danych testowych).

Ocena kosztów oprogramowania

Ocena kosztów opracowania sprowadza się do oceny:

– czasu realizacji,
– rozmiarów zatrudnienia (osobomiesiąc).

Jest trudnym i nie do końca rozwiązanym problemem.

- Podejścia heurystyczne:
  – ocena przez analogię,
  – ocena przez eksperta.
- Podejście analityczne:
  – budowa modelu złożoności przetwarzania (FPA)
  – obliczenie pracochłonności i czasu realizacji (COCOMO).

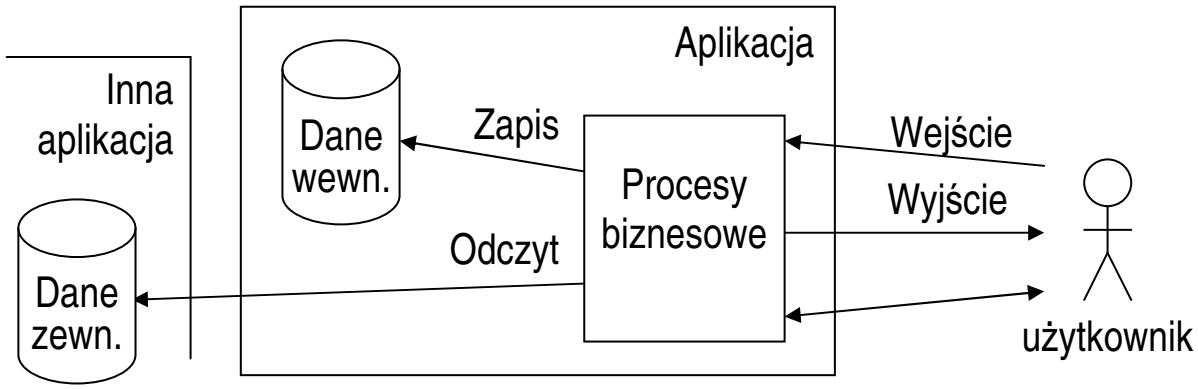
Metoda punktów funkcyjnych

(*Function Point Analysis* — FPA)

- Konsorcjum IFPUG (*International Function Point Users Group*)
- Ocena złożoności struktur danych i przetwarzania
- Wynikiem jest liczba (punktów funkcyjnych)

Możliwe przeliczenie: złożoność → rozmiaru → czas realizacji

Elementy oceny:



- zbiory wewnętrzne (*Internal Logical File — ILF*)
- zbiory zewnętrzne (*External Interface File — EIF*)
- wejścia zewnętrzne (*External Input — EI*)
- wyjścia zewnętrzne (*External Output — EO*)
- zapytania (*External Inquiry — EQ*)

1. Identyfikacja i ocena złożoności zbiorów i transakcji

- Ocena ILF i EIF

  RET — liczba różnych formatów rekordów (formularzy),
  DET — liczba pól rekordu,

Ocena ILF, EIF	***1...19 DET***	***20...50 DET***	***51... DET***
1 RET	*mała*	*mała*	*średnia*
2...5 RET	*mała*	*średnia*	*wysoka*
6... RET	*średnia*	*wysoka*	*wysoka*

- Ocena EI i EO

  DET — liczba pól rekordu,
  FTR — liczba zbiorów ILF i EIF używanych w transakcji

Ocena EI	***1...4 DET***	***5...15 DET***	***16... DET***
0...1 FTR	*mała*	*mała*	*średnia*
2 FTR	*mała*	*średnia*	*wysoka*
3... FTR	*średnia*	*wysoka*	*wysoka*
Ocena EO	***1...5 DET***	***6...19 DET***	***20... DET***
0...1 FTR	*mała*	*mała*	*średnia*
2...3 FTR	*mała*	*średnia*	*wysoka*
4... FTR	*średnia*	*wysoka*	*wysoka*

- Ocena EQ

  max [złożoność wejścia, złożoność wyjścia]

2. Obliczenie złożoności aplikacji

Złożoność	***mała***	***średnia***	***wysoka***	
zbiory wewnętrzne (ILF)	$x * 7 =$	$x * 10 =$	$x * 15 =$	Σ *wiersza*
zbiory zewnętrzne (EIF)	$x * 5 =$	$x * 7 =$	$x * 10 =$	Σ *wiersza*
wejścia zewnętrzne (EI)	$x * 3 =$	$x * 4 =$	$x * 6 =$	Σ *wiersza*
wyjścia zewnętrzne (EO)	$x * 4 =$	$x * 5 =$	$x * 7 =$	Σ *wiersza*
zapytania (EQ)	$x * 3 =$	$x * 4 =$	$x * 6 =$	Σ *wiersza*
nieskorygowane punkty funkcyjne UFP=				$\boldsymbol{\Sigma}$

3. Korygowanie oceny

Czynniki wpływające na złożoność przetwarzania, np:

- rozproszenie aplikacji,
- wymagana wydajność,
- złożoność algorytmów przetwarzania,
- zespół projektowy rozproszony w wielu miejscach,
- ....

Ocena roli czynnika (0..5):

żadna, nieznaczna, umiarkowana, średnia, znacząca, duża

$\rightarrow$ TDI (*Total Degree of Influence*).

Współczynnik korekcyjny:

$$VAF = 0.65 + (0.01 * TDI)$$

Złożoność skorygowana:

$$AFP = UFP * VAF$$

4. Oszacowanie rozmiaru programu

AFP jest miarą wysiłku potrzebnego do wytworzenia programu.

- Można na tym poprzestać i używać punktów do porównywania.
- Można obliczyć przewidywaną liczbę linii programu, korzystając ze współczynnika obrazującego wydajność wybranego języka (*Backfire Method*):

$$SLOC = AFP * LM$$

gdzie: *SLOC — Source Lines of Code*
LM — Language Multiplier

Język	*LM*
Asembler	*320*
Makroasembler	*213*
Pascal	*91*
Basic (kompilowany)	*91*
C	*128*
C++	*53*
Visual Basic v5	*29*
SQL	*13*
AI Shell	*49*

Model kosztów COCOMO

(*Constructive Cost Model*)

- Rozwijany przez konsorcjum *COCOMO Research Group*
- Podstawą oceny jest spodziewany rozmiar programu.
- Wynikiem analizy jest liczbowa ocena pracochłonności:

$$Effort = A * (KSLOC)^B * \Pi(F_i)$$

oraz optymalny czas realizacji:

$$Time = C * (Effort)^D$$

gdzie:

A, B, C, D — stałe zależne od rodzaju systemu:

	organic	***semidetached***	***embedded***
A	3.2	3.0	2.8
B	1.05	1.12	1.20
C	2.5	2.5	2.5
D	0.38	0.35	0.32

F_i — czynniki wpływające na koszt projektu, np.:

- wymagana niezawodność (0.75 .. 1.40),
- rozmiar bazy danych (0.94 .. 1.18),
- złożoność (0.70 .. 1.65)
- ograniczenia wydajnościowe (1.00 .. 1.66)
- ograniczenia pamięciowe (1.00 .. 1.56)
- biegłość analityków (0.71 .. 1.46)
- kwalifikacje projektantów (0.70 .. 1.42)
- biegłość programistów (0.95 .. 1.14)
- wykorzystanie metod IO (0.82 .. 1.24)
- wykorzystanie narzędzi CASE (0.83 .. 1.24)
- .....

Model COCOMO II

Zestaw 3 modeli stosowanych do różnych typów i faz projektu:

– kompozycyjny,
(aplikacje generowanych za pomocą *GUI-builder tools*),

– wczesny,
(budowany na początku, przed projektem architektury),

– po-architektoniczny,
(budowany po określeniu architektury projektu),

- Szacowanie rozmiaru programu: $SLOC = \underline{UFP} * LM$
- Podstawowy wzór b.z.:

  $Effort = A * (KSLOC)^B * \Pi(F_i)$

- Dokładniejsza ocena wykładnika B:

  $B = 1.01 + 0.01 * \Sigma(S_i)$

  gdzie S_i (*scale factors*) oceniane: *very high* (0) ... *very low* (5) odzwierciedlają różne uwarunkowania projektu:

  PREC — powtarzalność,
  FLEX — elastyczność wymagań,
  RESL — jakość analizy ryzyka,
  TEAM — spójność celów,
  PMAT — dojrzałość procesu (wg. *Capability Maturity Model*).

  // $B \in [\ 1.01\ ,\ 1.26]$
  // $5 * nominal = 5 * 0.3 \quad \Rightarrow \quad B=1.16$

- Model wczesny

  $Effort = A * (KSLOC)^B * \Pi_{i=1..7}(F_i)$

  7 mnożników kosztu (ocena w 7 punktowej skali symetrycznej):

  RCPX — złożoność i niezawodność systemu,
  RUSE — przewidywane ponowne wykorzystanie komponentów,
  PDIF — ograniczenia sprzętowe,
  PERS — kwalifikacje zespołu projektowego,
  PREX — znajomość platformy projektowej,
  FCIL — użycie narzędzi wspomagających i rozproszenie zespołu,
  SCED — napięty harmonogram.

- Model po-architektoniczny

  $Effort = A * (KSLOC)^B * \Pi_{i=1..17}(F_i)$

  17 mnożników kosztu.

- Czas realizacji:

  $Time = [\ 3.67 * (\ \underline{Effort}\ )^{(0.28+0.2*(B-1.01))}\ ] * SCED$

  gdzie *Effort* oznacza pracochłonność **bez** uwzględnienia SCED.

Metody oceny rozwiązań

Może być wiele koncepcji rozwiązania problemu, np.:

(A) opracowanie nowego systemu,
(B) adaptacja podobnego, istniejącego systemu,
(C) adaptacja oprogramowania SAP do potrzeb firmy,
(D) rozbudowa istniejącego i pozostawienie części działań ręcznych.

Problem: co wybrać? — kryteriów oceny jest wiele, np.:

- koszt opracowania i wdrożenia,
- czas realizacji,
- niezawodność rozwiązania systemu,
- wydajność.

- Problem optymalizacji wielokryterialnej

Kryterium	**A**	**B**	**C**	**D**
Koszt [tys. zł]	150	80	170	80
Czas realizacji [miesiące]	12	18	10	18
Niezawodność [aw./mies.]	1	3	2	3
Wydajność [trans./min.]	5	4	3	1

1. Usuniecie rozwiązań zdominowanych (tu D przez B)
2. Wypunktowanie rozwiązań:
   a) przypisanie wag (np.: bardzo ważne – 3 ... nieważne – 0)
   b) normalizacja ocen (najgorsze=0 ... najlepsze=1):
   ocena = (wartość – najgorsze) / (najlepsze – najgorsze)
   c) obliczenie sumy ważonej.

Wagi		**A**	**B**	**C**		**A**	**B**	**C**
3		0,22	1	0		0,66	3	0
2		0,75	0	1		1,5	0	2
3		1	0	0,5		3	0	1,5
1		1	0,5	0		1	0,5	0
Suma						**6,16**	3,5	3,5

- Uwzględnienie niepewności

  Oszacowanie daje zwykle wynik w pewnym zakresie, np.:

  koszt A: od 120 (optymistycznie) do 180 (pesymistycznie)
  koszt B: od 50 (optymistycznie) do 200 (pesymistycznie)

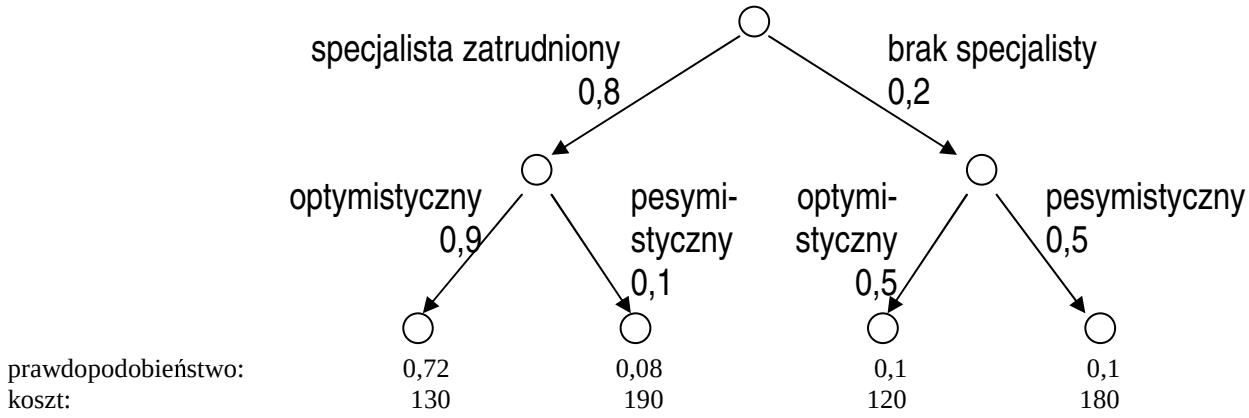
  jeśli można oszacować prawdopodobieństwo, np.:

  A: 0,5 (wariant optymistyczny) i 0,5 (wariant pesymistyczny)
  B: 0,8 (wariant optymistyczny) i 0,2 (wariant pesymistyczny)

  to można też obliczyć oczekiwaną wartość kosztu:

  A: $koszt = 120 * 0{,}5 + 180 * 0{,}5 = 60 + 90 = 150$
  B: $koszt = 50 * 0{,}8 + 200 * 0{,}2 = 40 + 40 = 80$

  Rachunek jest bardziej złożony, jeśli czynników losowych jest kilka. Można się tu posłużyć drzewem ryzyka:



  i obliczyć wartość oczekiwaną (uwzględniając gażę specjalisty):

  $koszt = 130*0{,}72 + 190*0{,}08 + 120*0{,}1 + 180*0{,}1$

  $= 93{,}6 + 15{,}2 + 12 + 18 = 138{,}8$