Zasadnicze czynności w zarządzaniu projektem, fazy cyklu życia systemu informatycznego. Modele cyklu życia - część 1

Zofia Kruczkiewicz

Literatura

1. Roger S. Pressman, Praktyczne podejście do oprogramowania, WNT, 2004
2. Stephen H. Kan, Metryki i modele w inżynierii jakości oprogramowania, Mikom, 2006
3. Jacobson, Booch, Rumbaung, The Unified Software Development Process,Addison Wesley, 1999
4. Shalloway A.,Trott James R.,Projektowanie zorientowane obiektowo. Wzorce projektowe. Gliwice, Helion, 2005
5. Robert C. Martin, Micah Martin, Agile Programowanie zwinne. Zasady, wzorce i praktyki zwinnego wytwarzania oprogramowania w C#, Helion 2008

Wstęp - kierowanie projektami [1]

(wykład 1, slajdy 45-49, rozwinięcie treści slajdu 47)

Co obejmuje kierowanie projektem

Niezbędny element powstania systemów i produktów informatycznych obejmujący:

- Planowanie
- Monitorowanie
- Organizowanie i kierowanie działaniami wykonawców
- Organizowanie i kierowanie procesami i zdarzeniami występującymi od koncepcji do implementacji

Kto kieruje projektem

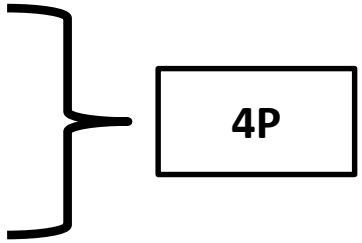
- **Wykonawcy**: analitycy, projektanci (w tym programiści) i testerzy
  - Planują, monitorują, kierują wykonaniem swoich codziennych czynności nad produktem
- **Kierownicy**
  - Planują, monitorują i kierują pracą wykonawców produktu
- **Dyrektorzy**
  - Koordynują działania kierowników z działaniami specjalistów w zakresie zastosowań produktu

Znaczenie kierowania projektem

- Skomplikowany proces tworzenia oprogramowania
- Długotrwała praca z ludźmi

Jak kieruje się projektem

1) Elementy kierowania:
   - **Ludzie** (***People***)
   - **Produkt** (***Product)***
   - **Proces** ***(Process)***
   - **Projekt** (***Project***)



2) Zorganizowanie działań pracowników
3) Komunikacja z klientem w zakresie ustalania wymagań
4) Wybór modelu procesu wytwórczego
5) Planowanie przebiegu prac, oceniając pracochłonność terminy wykonania etapów prac,
6) Określanie punktów kontroli jakości
7) Określenie metod kontrolowania postępu prac

Pierwszy wynik roboczy kierowania projektem

Plan realizacji projektu

- Model procesu wytwórczego
- Lista zadań do wykonania
- Przydział wykonawców do realizacji zadań
- Mechanizmy oceny ryzyka
- Zarządzanie zmianami
- Kontrola jakości

Skuteczne kierowanie projektem

- Kierownik pracuje dobrze, jeżeli:
  - podwładni stanowią zgrany zespół
  - skupiają się na jakości produktu
  - zaspakajają wymagania klienta
- Plan przedsięwzięcia jest dobry, jeśli dostarczy się klientowi produkt:
  - dobrej jakości
  - przygotowany na czas
  - w zgodzie z ustalonym budżetem

Przegląd zagadnień związanych z zarządzaniem projektami programistycznymi

Przegląd zagadnień związanych z zarządzaniem projektami programistycznymi

Ludzie (dalszy ciąg w jednym z kolejnych wykładów**)**

Wykonawcy

- **P-CMM – standardowy model dojrzałości zarządzania wykonawcami oprogramowania** (people capability maturity model), **uzupełniający model dojrzałości procesu:**
  - „przygotować firmy programistyczne do tworzenia coraz bardziej złożonych aplikacji, pomagając im przyciągać, wykształcić, motywować, używać i utrzymywać utalentowanych pracowników potrzebnych do usprawnienia działania firmy”
  - Rekrutacja i selekcja, ocena wyników pracy, szkolenie, wynagrodzenie, możliwość zrobienia kariery, organizacja pracy i kultura organizacyjna

Uczestnicy (wykład 1, slajd 7)

- Wykonawcy:
  - Dyrektorzy i prezesi
  - Kierownicy
  - Analitycy, projektanci (w tym programiści), testerzy
- Zamawiający
  - Klienci
  - Użytkownicy
  - Eksperci

Przegląd zagadnień związanych z zarządzaniem projektami programistycznymi [1]

Produkt –oprogramowanie (wykład 1, slajdy 8-24)

Produkt - koncepcja

- Należy ustalić cele i zadania, jakie ma realizować produkt
  - Spotkania programistów z klientami - cele i zakres działania produktu formułowane z punktu widzenia klienta
  - Rozważyć możliwe rozwiązania, ograniczenia organizacyjne i techniczne w celu:
    - Oszacowania kosztów
    - Przeprowadzenie analizy ryzyka
    - Wykonania harmonogramu prac

Produkt – zakres działania oprogramowania

- Cel
- Wymagania funkcjonalne: funkcje i efektywność wykonania funkcji
- Wymagania niefunkcjonalne: kontekst użycia, dane ilościowe, czasowe

Produkt – etapy dekompozycji oprogramowania

- Dekompozycja funkcji realizowanych przez oprogramowanie (diagram przypadków użycia)
- Dekompozycja metod realizacji tych funkcji

Symptomy złego produktu (oprogramowania) [5]

1) Sztywność

   Bardzo wiele zmienianych modułów w celu wprowadzenia nawet najdrobniejszych zmian

2) Wrażliwość

   Tendencja do ulegania uszkodzeniom lub usterkom w wielu miejscach wskutek wprowadzenia nawet najprostszych zmian. Często usterki pojawiają się w miejscach, gdzie wydają się być nie związane z dokonanymi zmianami.

3) Nieelastyczność

   Projekt zawiera elementy, które mogłyby być wykorzystane w innych systemach, jednak nie mogą być oderwane od oryginalnego systemu.

Symptomy złego produktu (oprogramowania) [5]

4) Niedostosowanie do rzeczywistości
   - Niedostosowanie oprogramowania do zmian (zmiany można realizować na wiele sposobów – sposoby prostsze są sprzeczne z projektem)
   - Niedostosowanie środowiska (długi czas kompilacji, zbyt długa kontrola wersji itd.)
5) Nadmierna złożoność
   - Program zawiera elementy, które są zbędne. Wynikają one z przedwczesnego przystosowania kodu do ewentualnych zmian, jednak skutek jest odwrotny – część udogodnień może być nigdy nie wykorzystana.
6) Niepotrzebne powtórzenia
   - Skutek zastosowania wycinania i wklejania fragmentów kodu przez różnych uczestników tworzenia programu
7) Nieprzejrzystość
   - Niezrozumiały, trudny do odczytania kod – programista nie wczuwa się w rolę innego programisty, który też powinien zrozumieć dany kod w celu jego rozwijania (brak również komentarzy)

Przegląd zagadnień związanych z zarządzaniem projektami programistycznymi

Proces (wykład 1, slajdy 27-44)

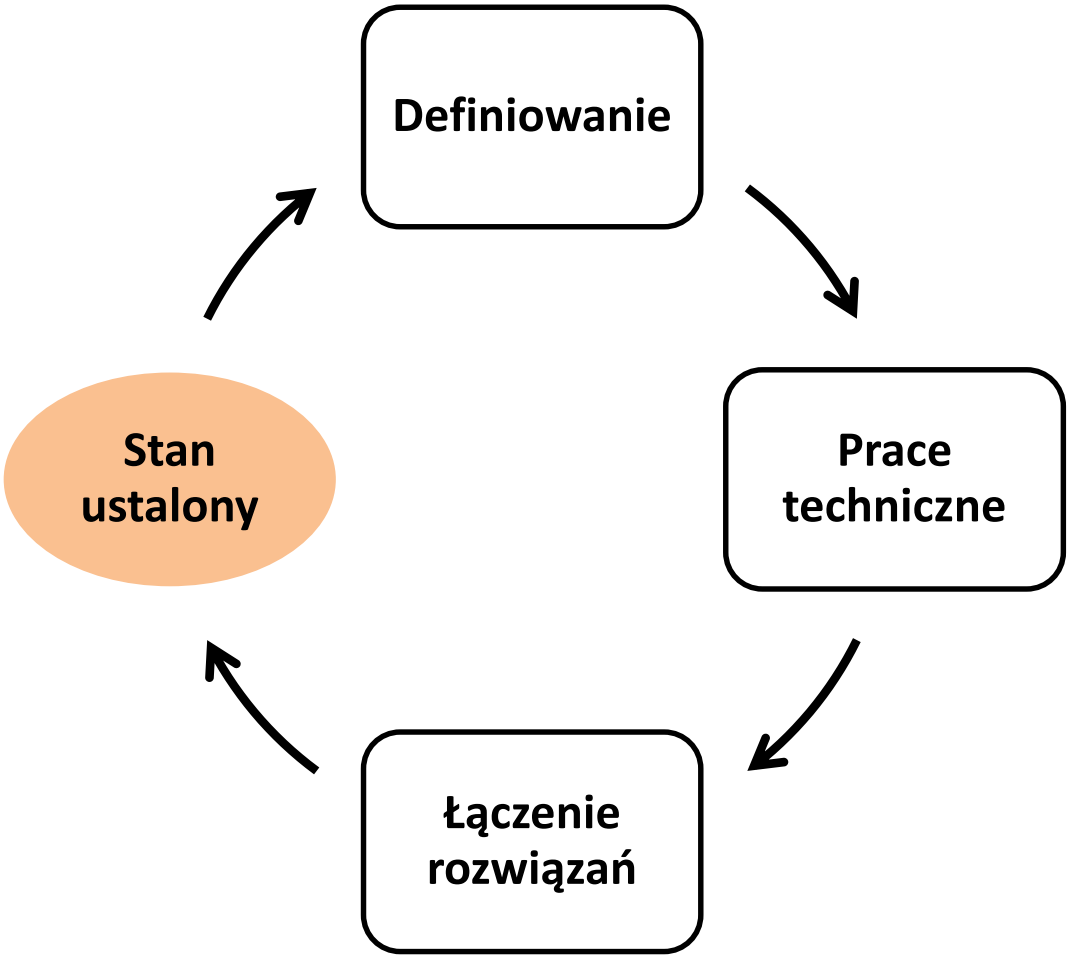
Proces: definiowanie, tworzenie i pielęgnowanie produktu

1) Wybór modelu procesu (potrzeby klientów i wykonawców, specyfika produktu, zwyczaje firmy)
2) Wstępny plan tworzenia produktu bazuje na uniwersalnym **schemacie procesu wytwórczego** (slajd 23)
3) Wybór konkretnego procesu wytwórczego
4) Dekompozycja procesu wytwórczego w celu wykonania kompletnego harmonogramu prac

Modele procesów tworzenia oprogramowania

- Koncepcja modeli

Uniwersalny schemat procesu wytwórczego (1) - model cyklu życia tworzenia oprogramowania [1]



Uniwersalny schemat procesu wytwórczego (2)- czyli model cyklu życia oprogramowania [3], [4]

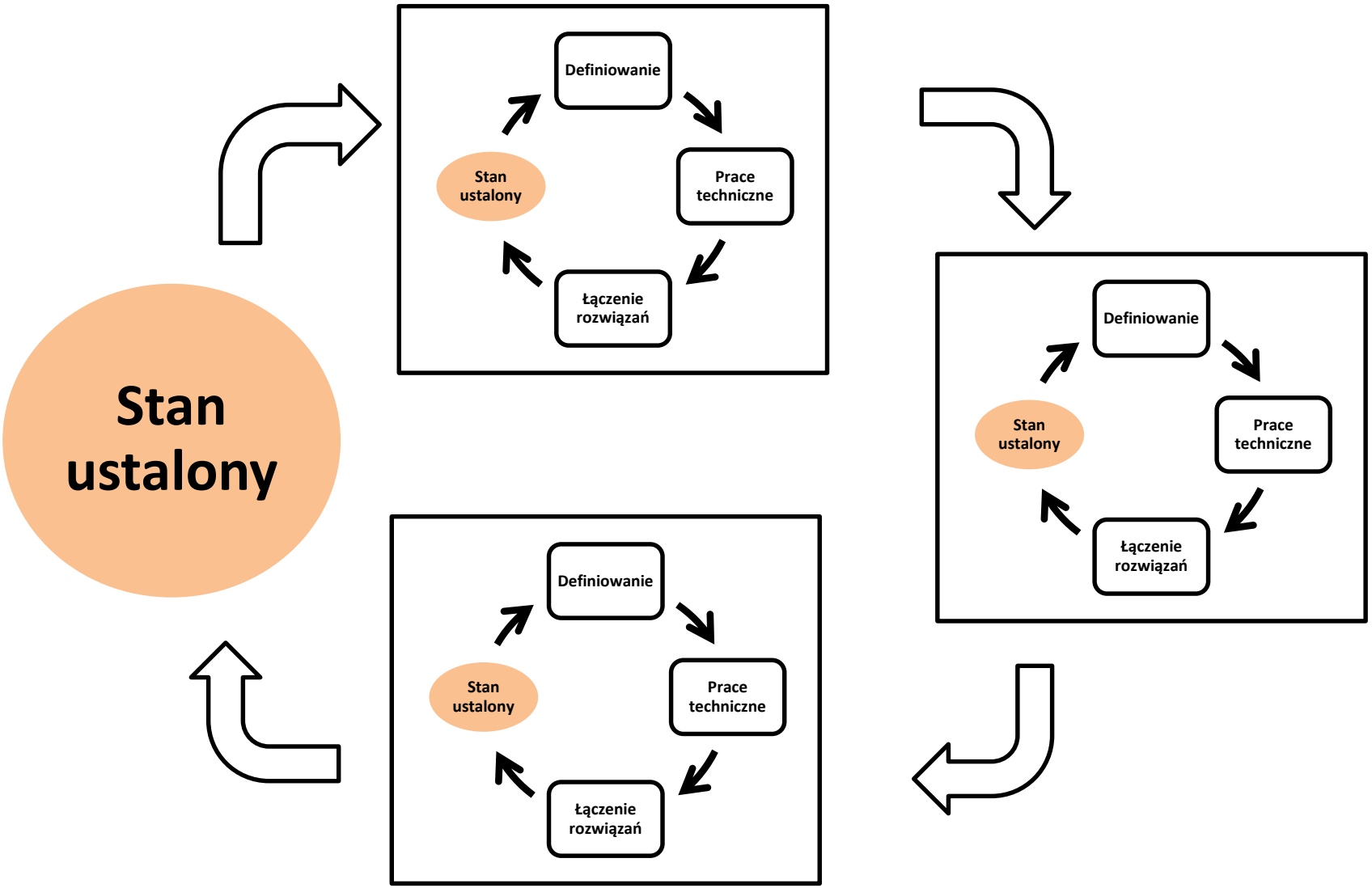
(powtórka - wykład 1, slajd 31)

Tworzenie technicznego systemu informacyjnego jest powiązane z:

- budową oprogramowania: **co wykonać?, jak należy używać?, jak należy wykonać? i kiedy wykonać?**
- wdrażaniem oprogramowania

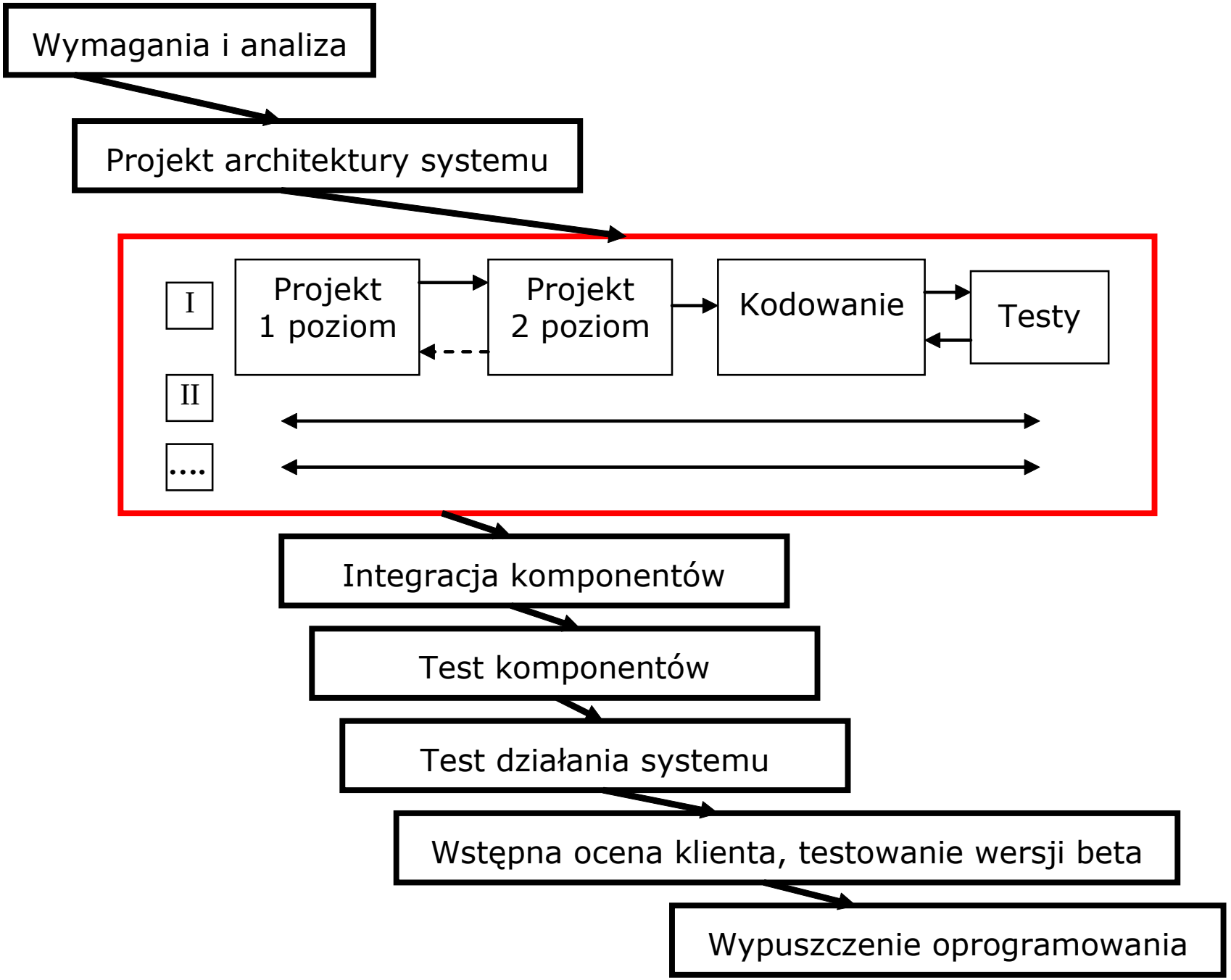
Modelowanie struktury i dynamiki systemu	**Implementacja systemu,**	**struktury i dynamiki generowanie kodu**
Perspektywa koncepcji ***co należy wykonać?***	Perspektywa specyfikacji ***jak należy używać?***	Perspektywa implementacji ***jak należy wykonać?***
• model problemu np. przedsiębiorstwa • wymagania • analiza (model konceptualny) • testy modelu	• projektowanie (model projektowy: architektura sprzętu i oprogramowania; dostęp użytkownika; przechowywanie danych) • testy projektu	• programowanie (specyfikacja programu : deklaracje, definicje; dodatkowe struktury danych: struktury „pojemnikowe”, pliki, bazy danych) • testy oprogramowania • wdrażanie • testy wdrażania

Model iteracyjnego cyklu życia procesu tworzenia oprogramowania [1]



Modele procesów tworzenia oprogramowania – przykłady [2]

- Proces - model kaskadowy
- Produkt – oprogramowanie obiektowe i nieobiektowe


Wymagania i analiza
Projekt architektury systemu
I
II
....
Projekt
1 poziom
Projekt
2 poziom
Kodowanie
Testy
Integracja komponentów
Test komponentów
Test działania systemu
Wstępna ocena klienta, testowanie wersji beta
Wypuszczenie oprogramowania

2 początkowe fazy kaskadowego cyklu życia tworzenie oprogramowania

- **Wymagania i analiza**
  - Analiza wymagań i potrzeb klienta
  - Wstępna analiza tworzonego systemu

- **Projekt architektury systemu**
  - Projekt struktury systemu w postaci komponentów

3-a faza kaskadowego cyklu życia tworzenia oprogramowania – zawiera współbieżnie realizowane fragmenty oprogramowania

- Projekt 1 poziomu- projektowanie zewnętrznych i wewnętrznych cech oprogramowania z perspektywy komponentów:
  - Wykonanie zewnętrznych funkcji i interfejsów
  - Projekt wewnętrznej struktury komponentu (interfejsy i struktury danych)
  - Sprawdzenie, czy wymagania funkcjonalne są spełnione
  - Sprawdzenie, czy prawidłowo dobrano komponenty
  - Sprawdzenie, czy projekty komponentów są dopracowane
  - Sprawdzenie, czy oczekiwane funkcje będą prawidłowo wykonywane

3-a faza kaskadowego cyklu życia tworzenia oprogramowania – zawiera współbieżnie realizowane fragmenty oprogramowania

- Projekt 2 poziomu- przekształcanie produktów I poziomu w bardziej szczegółową postać
  - Opracowanie projektów komponentów
  - Opracowanie planów testów komponentów
  - Weryfikacja produktu I poziomu projektowania
- Kodowanie
  - Kodowanie modułów, makr, bibliotek, itd
  - Kodowanie przypadków testowych
  - Weryfikacja zmian w projektach I i II poziomu
- Testy
  - Ocena poprawności różnych części kodu komponentów (testy jednostkowe) oraz związku tego kodu z projektami I i II poziomu

5, 6 i 7 fazy kaskadowego cyklu życia tworzenia oprogramowania

- Test komponentów
  - Test interfejsów: użytkownika, międzykomponentowych i wewnętrznych komponentów pod względem poprawności kodu (testy integracyjne) i spełniania wymagań klienta
- Test działania systemu
  - Test akceptacyjny systemu
  - Regresyjny test systemu (test po wprowadzenie zmian),
  - Test wydajności systemu
  - Test funkcjonalny systemu pod kątem łatwości obsługi
- Wstępna ocena klienta, testowanie wersji beta
  - Dostarczanie wstępnych programów klienta (ECP – early customer programs) w celu uzyskania oceny klienta w zakresie niezawodności, wydajności, instalacji i obsługi

Właściwości modelu kaskadowego – „dziel i rządź”: zalety

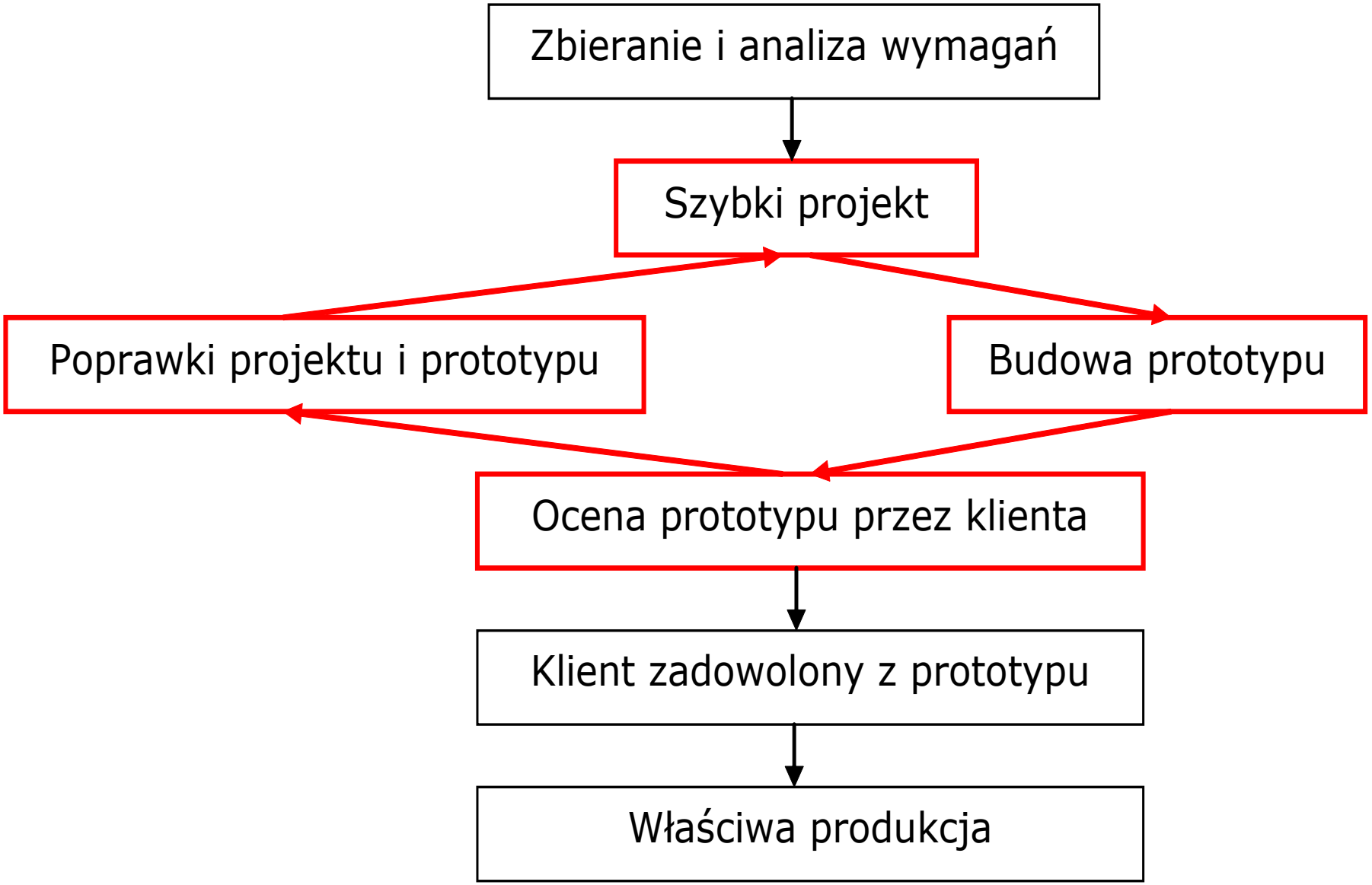
- Schemat działania: **Wejście –Zadanie – Sprawdzenie – Wyjście** (Entry-Task-Validation_Exit)
- Odpowiedni dla doświadczonych wykonawców i klientów poprawnie definiujących proces biznesowy
- Ułatwienie prowadzenia dużego, złożonego projektu
- Podstawowe podejście przy tworzeniu oprogramowania metodą strukturalną
- Umożliwia dotrzymanie terminu wykonania w ramach określonego budżetu
- Doświadczenia ostatnich dekad potwierdziły jego przydatność np. podczas tworzenia systemów operacyjnych

Właściwości modelu kaskadowego – „dziel i rządź”: wady

- Brak kontaktów z klientem
- Brak odporności na zmianę wymagań klienta
- Możliwość strat, ponieważ ostateczna ocena następuje pod koniec cyklu życia oprogramowania

Modele procesów tworzenia oprogramowania – przykłady [2]

- Proces - model prototypowy
- Produkt – oprogramowanie obiektowe i nieobiektowe


Zbieranie i analiza wymagań
Szybki projekt
Poprawki projektu i prototypu
Budowa prototypu
Ocena prototypu przez klienta
Klient zadowolony z prototypu
Właściwa produkcja

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

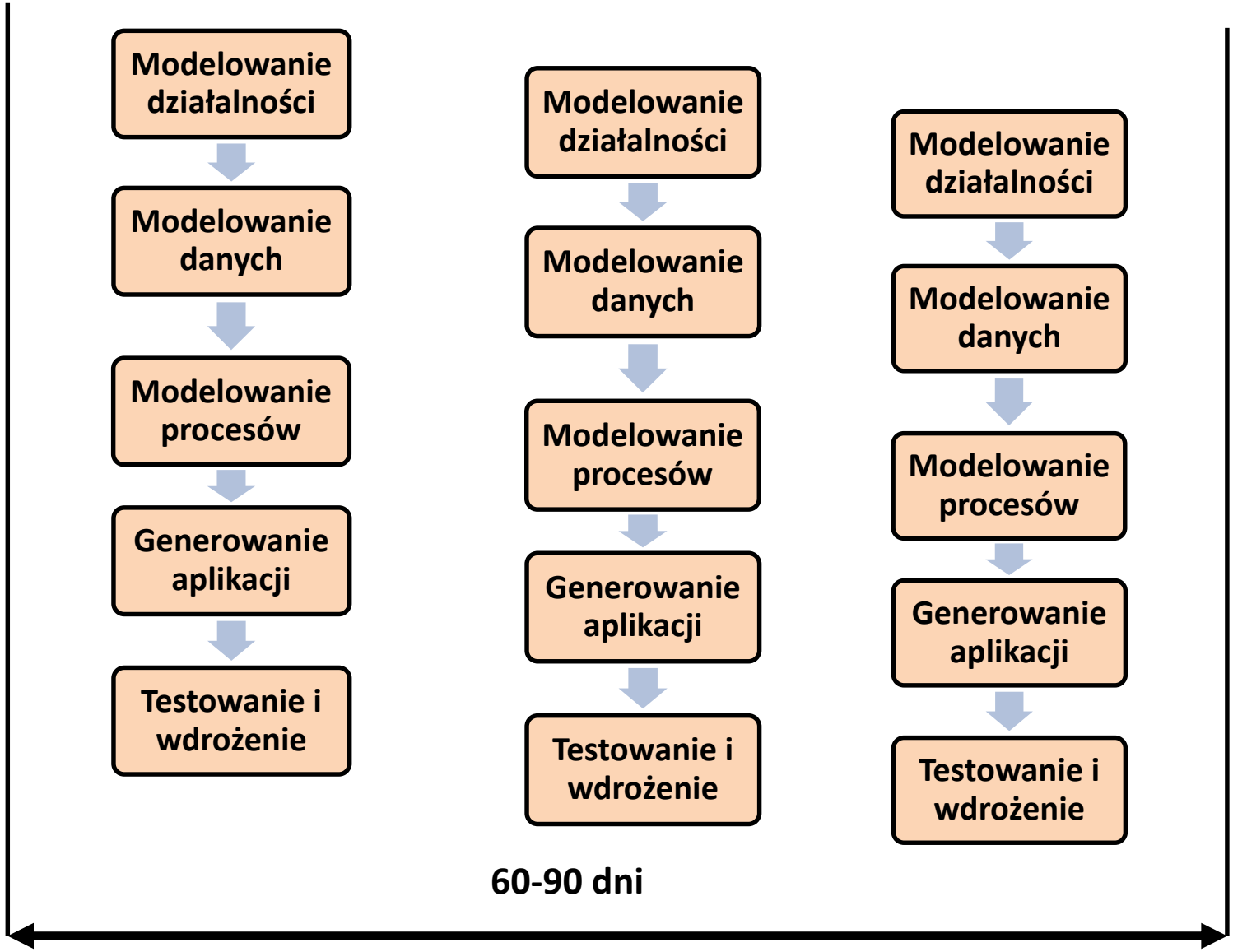
Właściwości tworzenia prototypów

- 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

Modele procesów tworzenia oprogramowania – przykłady [1]

- Proces - model szybkiej rozbudowy aplikacji RAD (Rapid application development)
- Produkt – oprogramowanie obiektowe i nieobiektowe

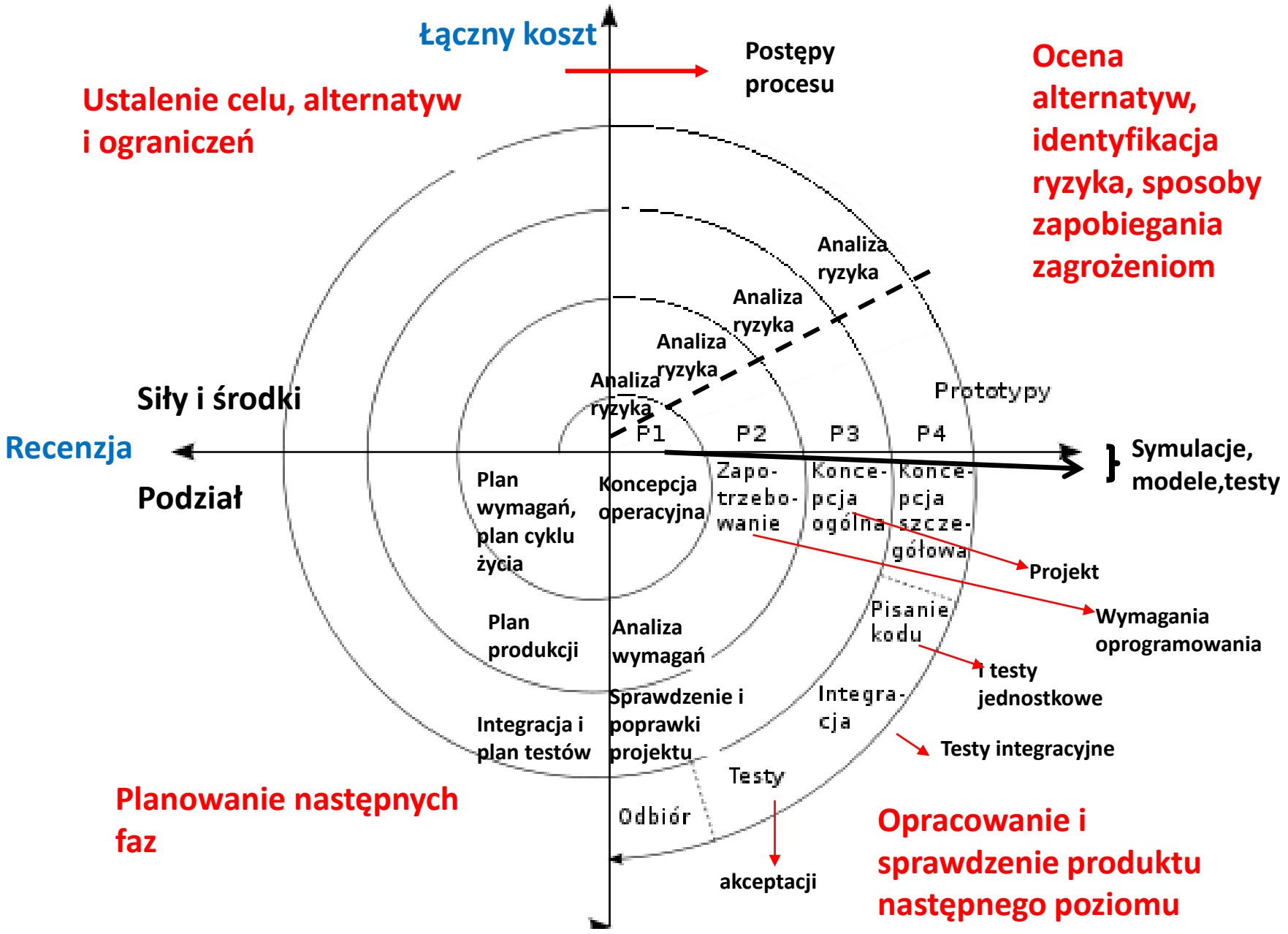

Modelowanie działalności
Modelowanie danych
Modelowanie procesów
Generowanie aplikacji
Testowanie i wdrożenie
Modelowanie działalności
Modelowanie danych
Modelowanie procesów
Generowanie aplikacji
Testowanie i wdrożenie
Modelowanie działalności
Modelowanie danych
Modelowanie procesów
Generowanie aplikacji
Testowanie i wdrożenie
60-90 dni

Cechy RAD

- Duża liczba wykonawców dla dużych projektów
- Klienci i wykonawcy z dużą motywacją do szybkiego ukończenia projektu
- Odpowiedni tylko dla zagadnień i systemów modularnych
- Nieodpowiedni dla systemów korzystających z nowych technologii
- Nieodpowiedni dla systemów współpracującymi intensywnie z innymi istniejącymi programami

Modele procesów tworzenia oprogramowania – przykłady [2]

- Modele ewolucyjne
  - **Proces - model spiralny**
  - **Produkt – oprogramowanie nieobiektowe i obiektowe**


Łączny koszt
Postępy procesu
Ustalenie celu, alternatyw i ograniczeń
Ocena alternatyw, identyfikacja ryzyka, sposoby zapobiegania zagrożeniom
Analiza ryzyka
Analiza ryzyka
Analiza ryzyka
Analiza ryzyka
Siły i środki
Prototypy
Recenzja
P1
P2
P3
P4
Symulacje, modele,testy
Podział
Plan wymagań, plan cyklu życia
Koncepcja operacyjna
Zapo-trzebo-wanie
Konce-pcja ogólna
Konce-pcja szcze-gółowa
Projekt
Wymagania oprogramowania
Pisanie kodu
Plan produkcji
Analiza wymagań
i testy jednostkowe
Sprawdzenie i poprawki projektu
Integra-cja
Integracja i plan testów
Testy integracyjne
Testy
Odbiór
akceptacji
Planowanie następnych faz
Opracowanie i sprawdzenie produktu następnego poziomu

Właściwości tworzenia modelu spiralnego

- Posiada zalety modeli kaskadowego i prototypu, natomiast analiza ryzyka pozwala unikać wad tych modeli
- Stosowana do dużych projektów
- Wielożywalność 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** (kolejne spirale: studium produktu, stworzenie nowego produktu, rozszerzenie produktu, pielęgnacja produktu)
- Solidny fundament do integrowania oprogramowania ze sprzętem

Właściwości tworzenia modelu spiralnego

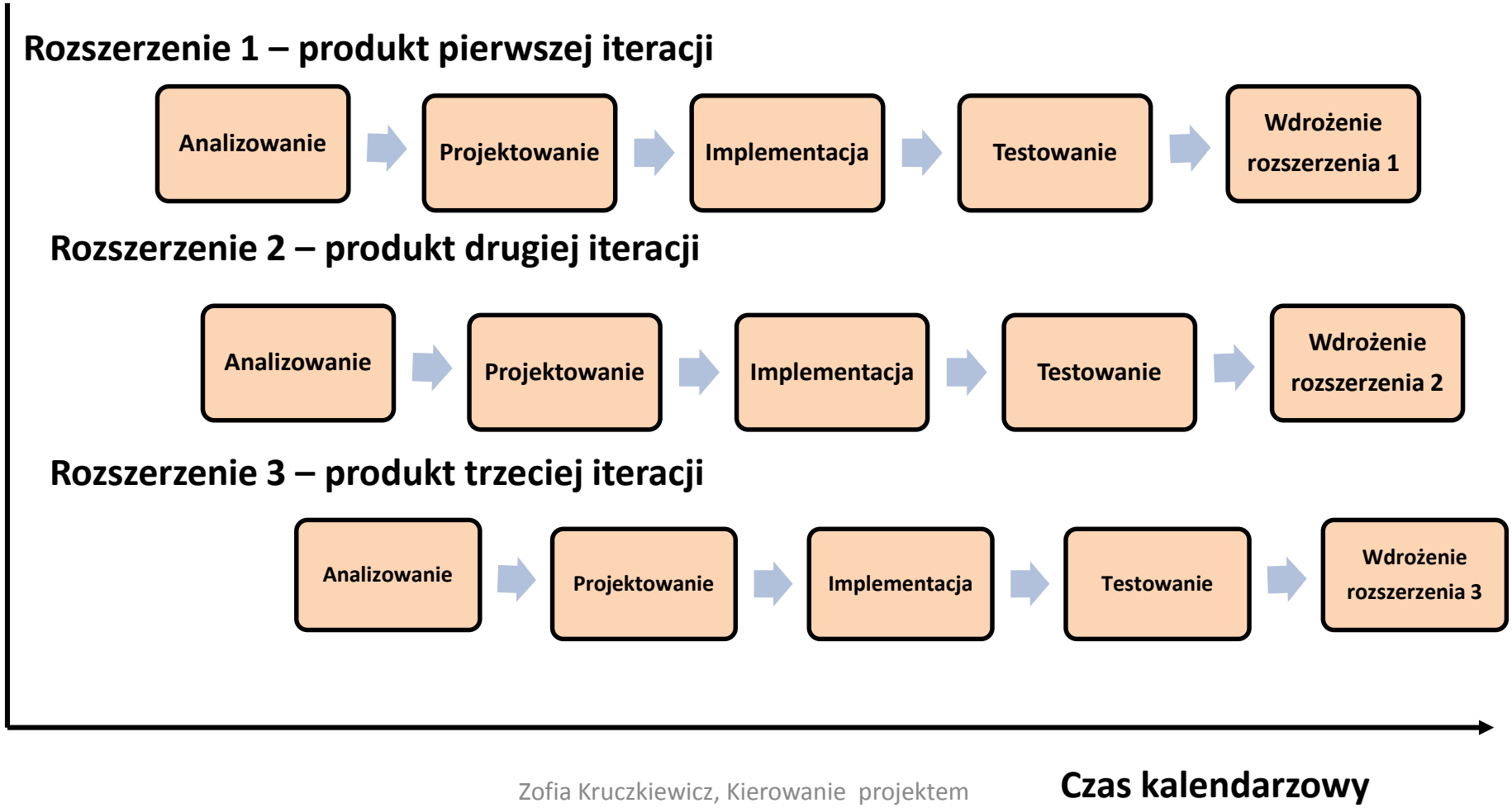
- Trudność z wywiązania się z warunków kontraktu, lepsza w przypadku budowania oprogramowania do sprzedaży
- Duży wpływ analizy ryzyka na przebieg projektowania - błędy w analizie mogą negatywnie wpłynąć na wynik produkcji oprogramowania
- Potrzeba opracowania kolejnych kroków przy zachowaniu spójności projektu - może ją stosować jedynie zespół profesjonalistów

Modele procesów tworzenia oprogramowania – przykłady

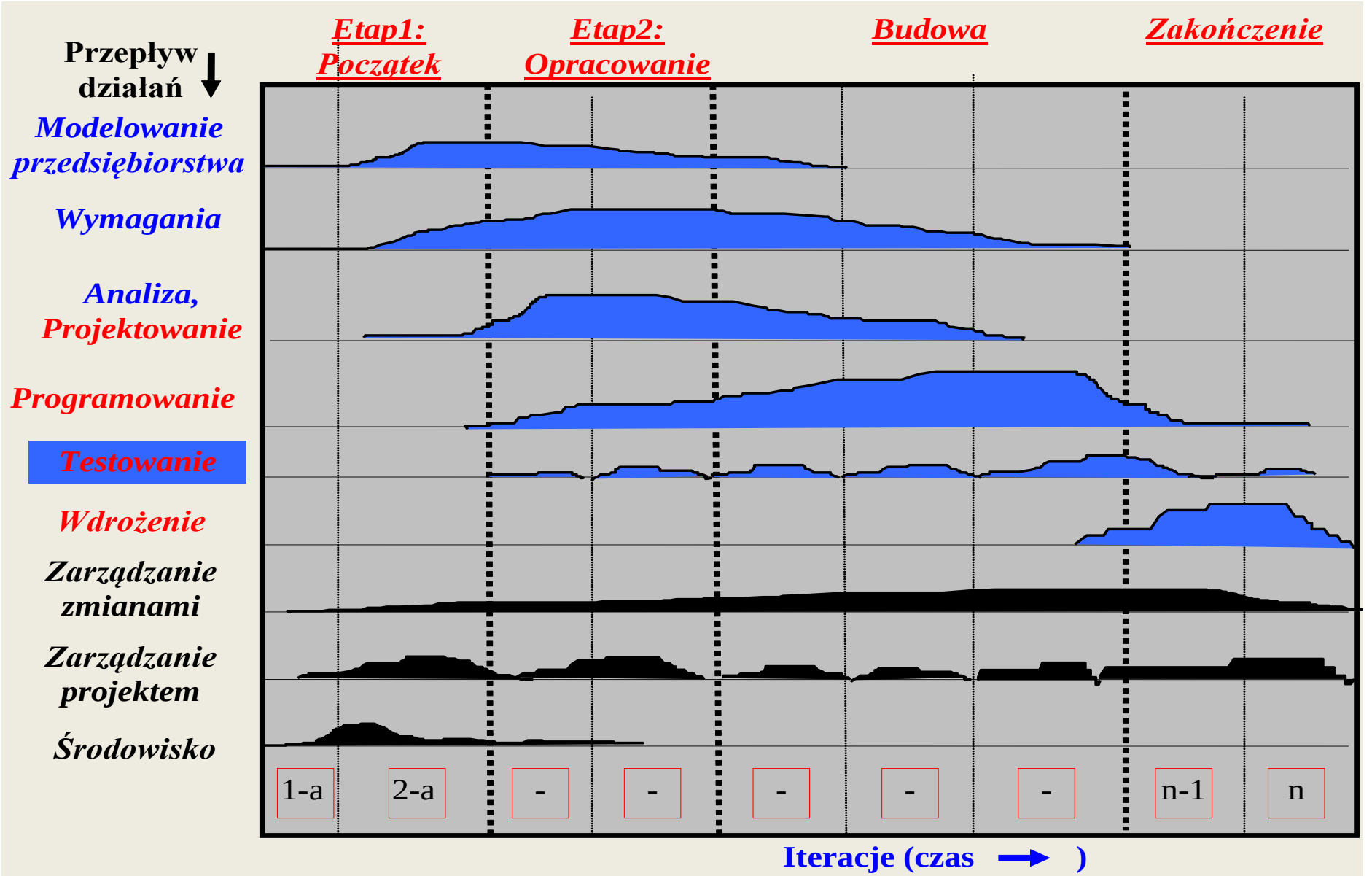
- Modele ewolucyjne
  - **Proces - model iteracyjno-rozwojowy**
  - **Produkt – oprogramowanie obiektowe**

Ogólna zasada [1]

powiązany z czynnościami przekrojowymi (wykład 1, slajdy 34-35)



Zunifikowany iteracyjno- przyrostowy proces tworzenia oprogramowania 1997 [3], (wykład 1, slajdy 51-57)



Przepływy czynności

- Modelowanie przedsiębiorstwa – opis dynamiki i struktury przedsiębiorstwa
- Wymagania – zapisanie wymagań metodą opartą na przypadkach użycia
- Analiza i projektowanie – zapisanie różnych perspektyw architektonicznych
- Implementacja – tworzenie oprogramowania, testowanie modułów, scalanie systemu
- Testowanie – opisanie danych testowych, procedur i metryk poprawności
- Wdrożenie – ustalenie konfiguracji gotowego systemu
- Zarządzanie zmianami – panowanie nad zmianami i dbanie o spójność elementów systemu
- Zarządzanie projektem - opisane różnych strategii prowadzenia procesu iteracyjnego
- Określenie środowiska – opisanie struktury niezbędnej do opracowania systemu

Co i jak wykonać?

Perspektywy projektowania obiektowych systemów informacyjnych [4]

- **koncepcji** (model analizy)

  (co obiekty powinny robić?)

- **specyfikacji interfejsów** (model projektowy)

  (jak używać obiektów?)

- **implementacji** (implementacja)

  (w jaki sposób zaimplementować interfejs ?)

- **tworzenia i zarządzania obiektami** (implementacja)

  (*obiekt A w roli fabryki obiektów tworzy obiekt B i/lub zarządza obiektem)*

- **używania obiektów** (implementacja)

  (*obiekt A tylko używa obiektu B – nie może go jednocześnie tworzyć; opiera się na hermetyzacji i polimorfizmie obiektu B)*

Perspektywy rozumienia obiektów – identyfikacji obiektów [4]

- **Perspektywa koncepcji** (modelu konceptualnego)

  - obiekt jest zbiorem różnego rodzaju odpowiedzialności

- **Perspektywa specyfikacji** (modelu projektowego)

  - obiekt jest zbiorem metod (zachowań), które mogą być wywoływane przez metody tego obiektu lub innych obiektów

- **Perspektywa implementacji** (kodu źródłowego)

  - obiekt składa się z kodu metod i danych oraz interakcji między nimi

Perspektywy skalowania systemu – tworzenia, zarządzania i używania obiektów [4]

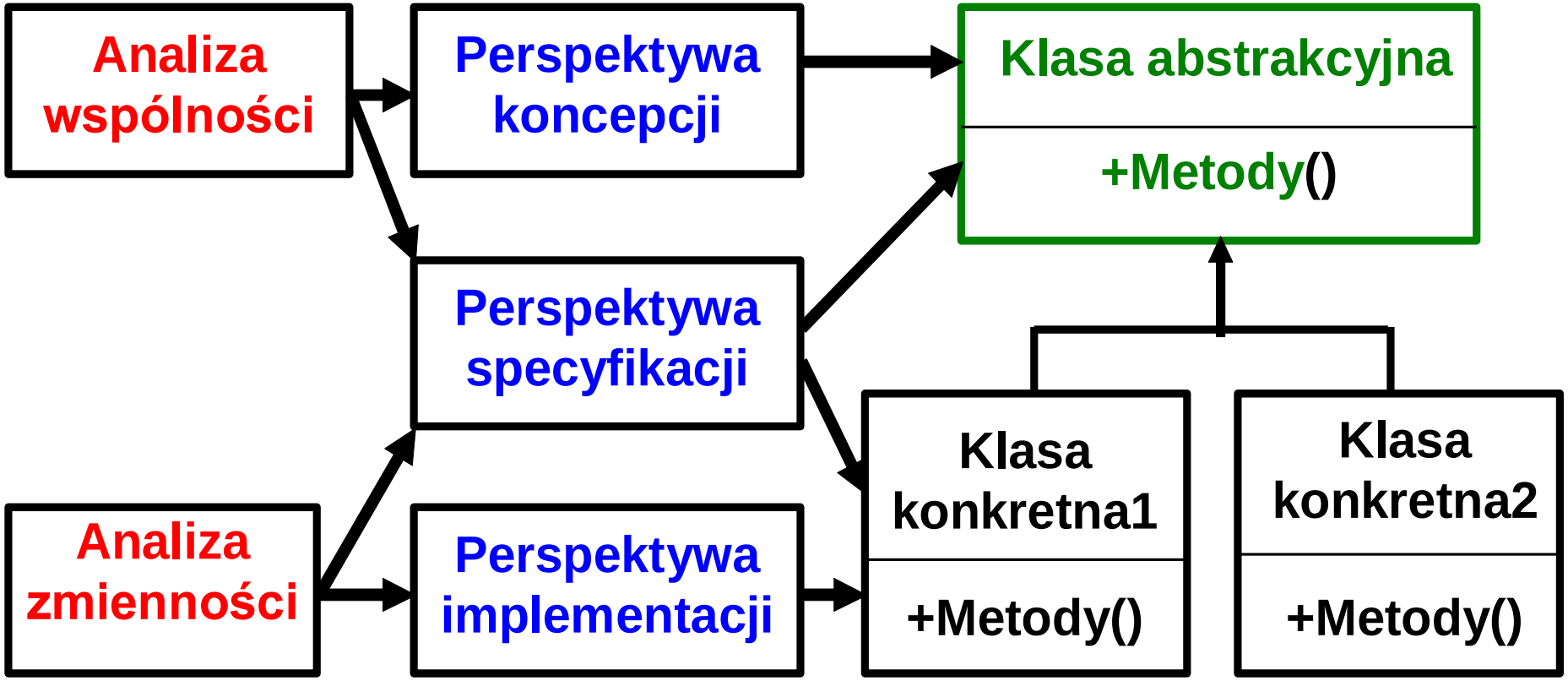
- **Perspektywa tworzenia i zarządzania obiektami**

  Zmiany w implementacji obiektów dotyczą obiektów czyli fabryk obiektów (tworzących te obiekty i zarządzających tworzeniem tych obiektów)

- **Używanie obiektów**

  Zmiana implementacji obiektów nie zmienia implementacji obiektów, które używają zmieniane obiekty

Metoda identyfikacji obiektów i klas [4]



Związek między perspektywą specyfikacji, koncepcji i implementacji

Zależności między analizą, projektowaniem i implementacją [4]

Związek między perspektywą koncepcji i specyfikacji

- Perspektywa specyfikacji określa interfejs potrzebny do obsługi wszystkich przypadków danego problemu (czyli **część wspólną** określoną przez perspektywę koncepcji)

Związek pomiędzy perspektywą specyfikacji i implementacji

- Biorąc pod uwagę określoną specyfikację ustala się, w jaki sposób należy zaimplementować poszczególne przypadki (czyli **część zmienną**)

8 kroków procesu iteracyjno-rozwojowego podsumowanie [2]

Fazy: modelowanie przedsiębiorstwa, wymagania, analiza

1. Modelownie podstawowego procesu
   - Tworzenie punktu widzenia klienta
   - Modelowanie podstawowych funkcji (przypadki użycia)
   - **Czynność przekrojowa** (wykład 1, slajdy 35-36): opiniowanie wymagań
   - Zdefiniowanie danych potrzebnych do zakończenia projektu
   - **Czynność przekrojowa** (wykład 1, slajdy 35-36): opiniowanie zewnętrznej struktury i poprawionego modelu
2. Uzyskanie proponowanych klas: perspektywa koncepcji podczas analizy wspólności (na podstawie scenariuszy przypadków użycia, słowników danych, zewnętrznych źródeł związanych z dziedziną wykonywanego oprogramowania)

8 kroków procesu iteracyjno-rozwojowego podsumowanie [2]

Faza: projekt

3. Ograniczenie podstawowego modelu, dostosowanie do możliwości wykonania – perspektywa specyfikacji podczas analizy wspólności
4. Opracowanie dodatkowych klas – perspektywa specyfikacji podczas analizy zmienności
5. Synteza klas – cd analizy wspólności i zmienności, korzystanie z wzorców projektowych

   Czynność przekrojowa (wykład 1, slajdy 35-36): sprawdzenie analizy klas
6. Definiowanie interfejsów – perspektywa specyfikacji

   Czynność przekrojowa (wykład 1, slajdy 35-36): przegląd zewnętrznej charakterystyki klas po kroku szóstym

8 kroków procesu iteracyjno-rozwojowego podsumowanie [2], [4]

Fazy: programowanie, testowanie, wdrożenie

7. Dokończenie projektu
   - perspektywa implementacji
   - logika programowania:
     - perspektywa tworzenia i zarządzania obiektami
     - perspektywa używania obiektów
8. Kodowanie klas i testy jednostkowe, tworzenie prototypu

Czynność przekrojowa (wykład 1, slajdy 35-36)**:** sprawdzenie kodu po zakończeniu wszystkich ośmiu kroków2

Model iteracyjno-przyrostowy – cechy [2]

- **Pierwsze rozszerzenie** jest rdzeniem systemu
- Potokowe wykonanie kolejnego rozszerzenia
- Przekazywanie kolejnych rozszerzeń może zapobiec opóźnieniom = częste kontakty z klientem
- Możliwy niepełny zbiór wymagań
- Przekazywanie coraz bardziej zaawansowanych wersji programu pozwala na wprowadzanie zmian wymagań = **ewolucyjna natura oprogramowania**
- Utrzymanie terminu – możliwość elastycznego reagowania na opóźnienia realizacji jednej części i przyspieszenie prac nad inną/innymi częściami
- Rozwijanie dokumentacji
- Ograniczona liczba wykonawców

Model iteracyjno-przyrostowy – wady [2]

- 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