Prowadzenie projektu programistycznego. Modele tworzenia oprogramowania. Programowanie kaskadowe i zwinne. Wykład 9

Wykładowca

Dr inż. Zofia Kruczkiewicz

Literatura podstawowa (LPU) – UML

1. G. Booch, J. Rumbaugh, I. Jacobson, UML przewodnik użytkownika, Seria Inżynieria oprogramowania, WNT, 2001, 2002. .
2. M. Fowler, UML w kropelce, Wersja 2.0, LTP, Warszawa, 2005.

Literatura uzupełniająca (LPW) – Wzorce projektowe

1. E. Gamma, R. Helm, R. Johnson, J. Vlissides, Wzorce projektowe, Elementy oprogramowania obiektowego wielokrotnego użytku, WNT, Warszawa, 2005.
2. Shalloway A.,Trott James R., Projektowanie zorientowane obiektowo. Wzorce projektowe. Gliwice, Helion, 2005

Literatura uzupełniająca (LU) – Inżynieria oprogramowania

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. K. Frączkowski, Zarządzanie projektem informatycznym. Projekty w środowisku wirtualnym. Czynniki sukcesu i niepowodzeń projektów., Oficyna Wydawnicza Politechniki Wrocławskiej

Struktura wykładu

I. Prowadzenie projektu programistycznego
II. Modele tworzenia oprogramowania
III. Programowanie kaskadowe i zwinne

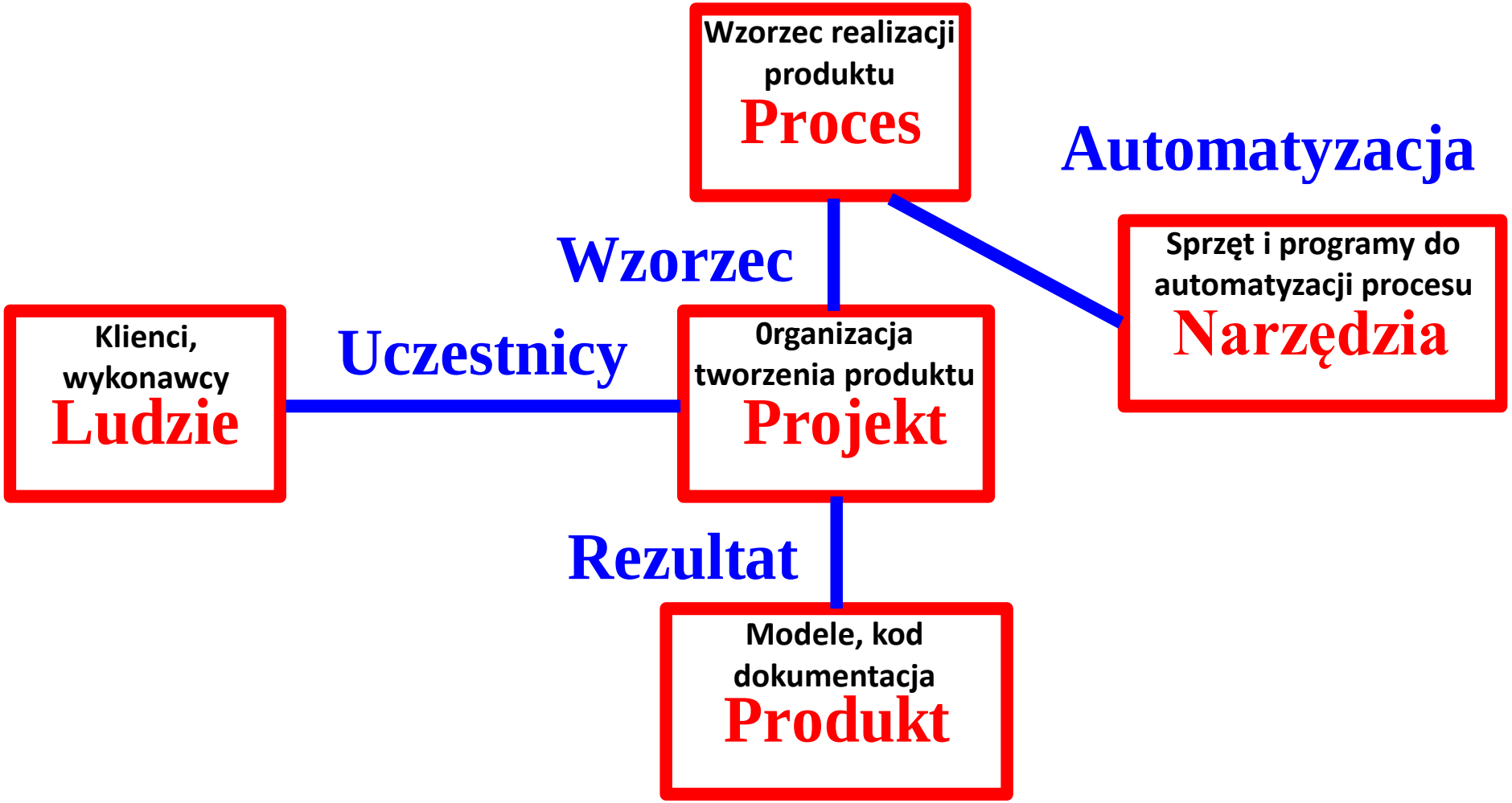
Zagadnienia

1. **Prowadzenie projektu programistycznego**
   (wykład 1, wykład 3)

Ogólne spojrzenie na inżynierię oprogramowania (wykład 1)

1) Jaki problem należy rozwiązać
2) Jakie cechy produktu umożliwiają rozwiązanie problemu
3) Jak ma wyglądać produkt (rozwiązanie problemu)
4) Jak skonstruować taki produkt
5) Jak wykrywać błędy w projekcie lub podczas konstrukcji produktu
6) Jak obsługiwać i pielęgnować gotowy produkt i jak uwzględniać uwagi, reklamacje i żądania jego użytkowników: **poprawianie, adaptowanie, rozszerzanie, zapobieganie**

Elementy tworzenia oprogramowania – struktura [3LU] (wykład 1)



Projekt - planowanie, organizowanie, monitorowanie i kontrolowanie [1LU] (wykład 1)

Główny, organizacyjny element powiązany z:

- Ludzie, Produkt, Proces

Pojęcia:

1. Wykonalność projektu
2. Zarządzanie ryzykiem
3. Struktura grup projektowych
4. Szeregowanie zadań projektowych
5. Zrozumiałość projektu
6. Sensowność działań w projekcie

Cechy projektu:

1. Sekwencja zmian w projekcie
2. Seria iteracji
3. Wzorzec organizacyjny

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

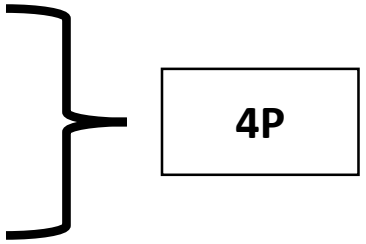
- **Wykonawcy**: analitycy, projektanci (w tym programiści) i testerzy
  - Planują, monitorują, kierują wykonaniem swoich codziennych czynności nad produktem
- **Kierownicy**
  - Planują, monitorują i kierują pracą wykonawców produktu
- **Dyrektorzy**
  - Koordynują działania kierowników z działaniami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

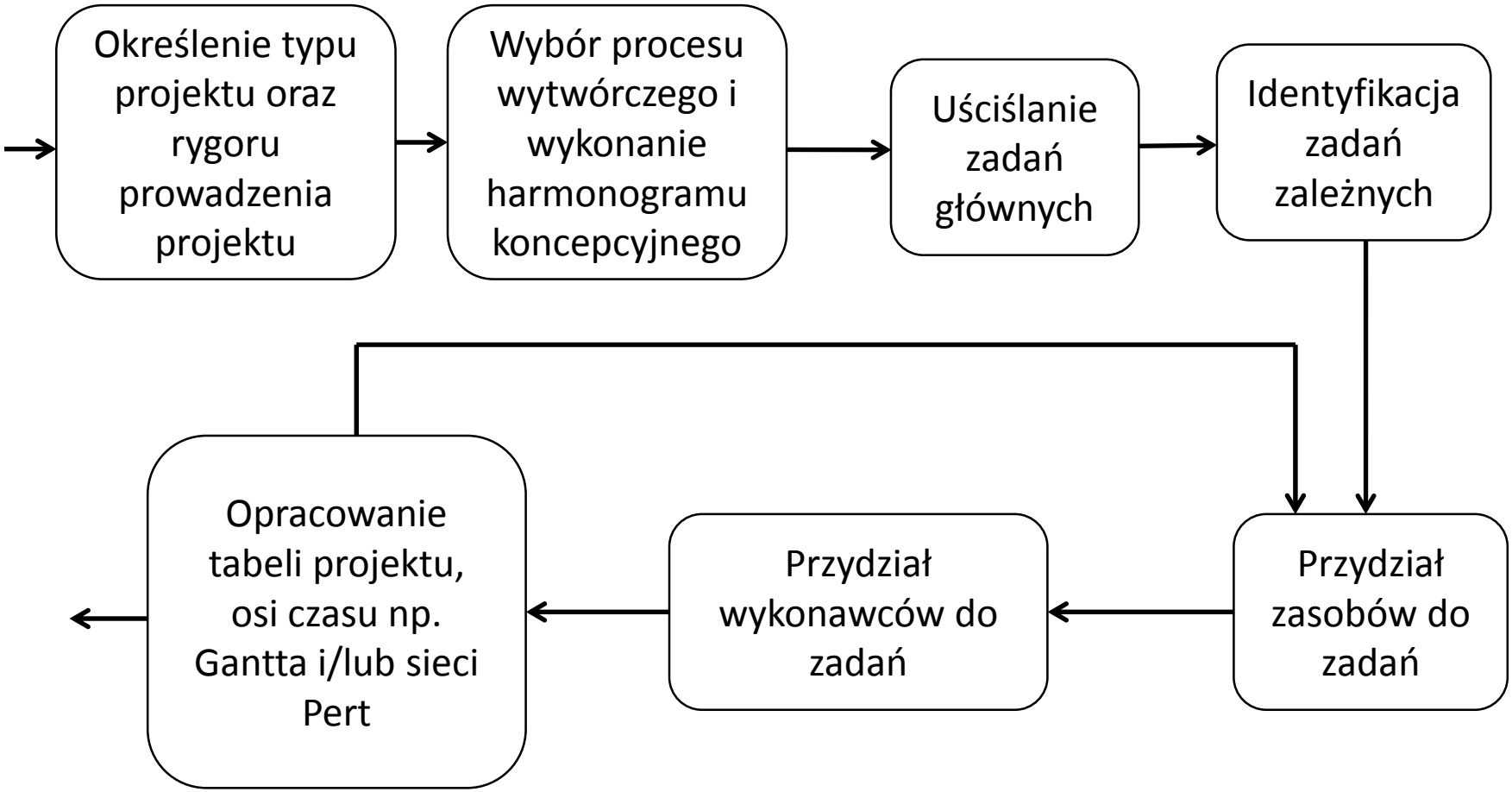
Skuteczne kierowanie projektem

- 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

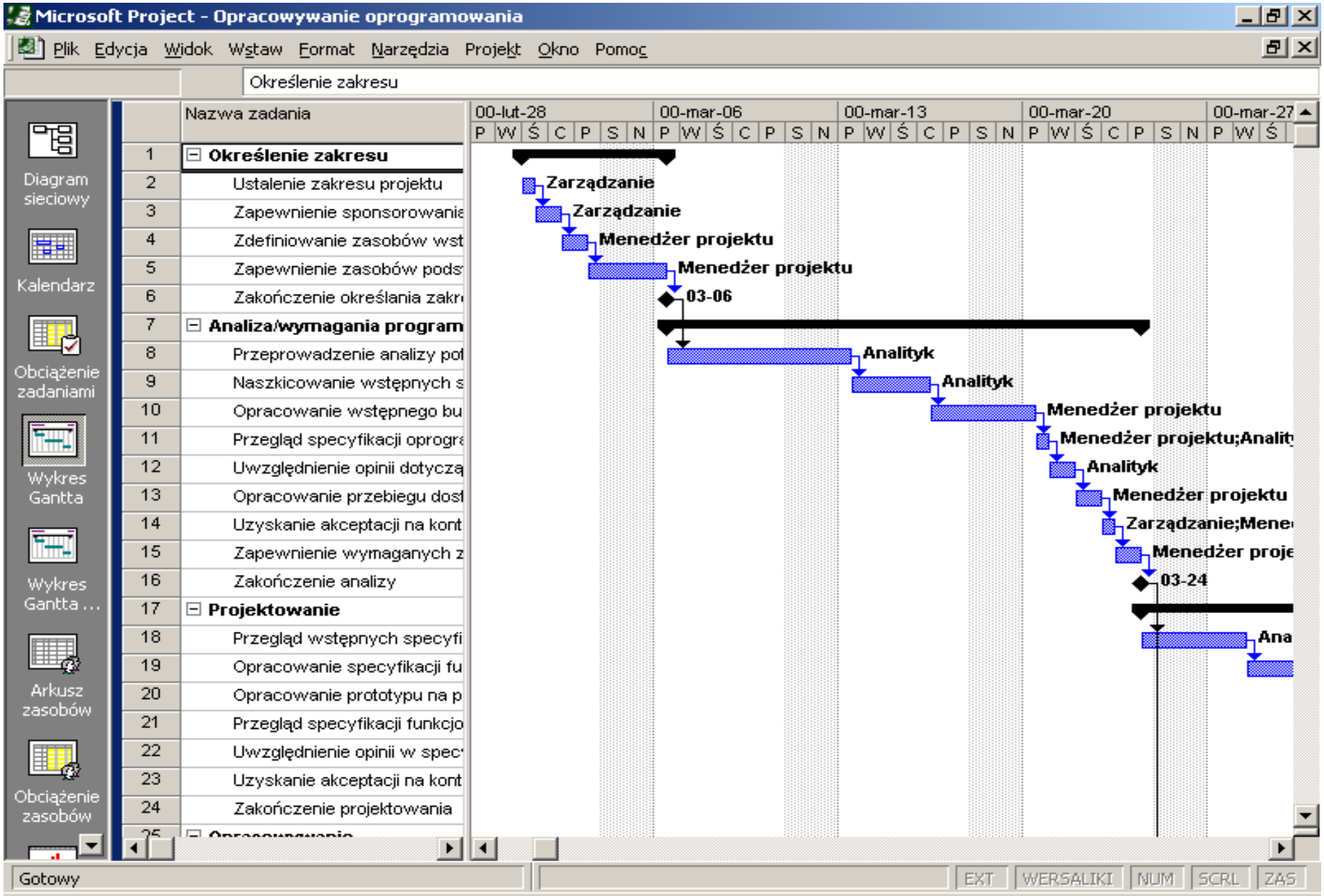
Wykonawcy

- **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

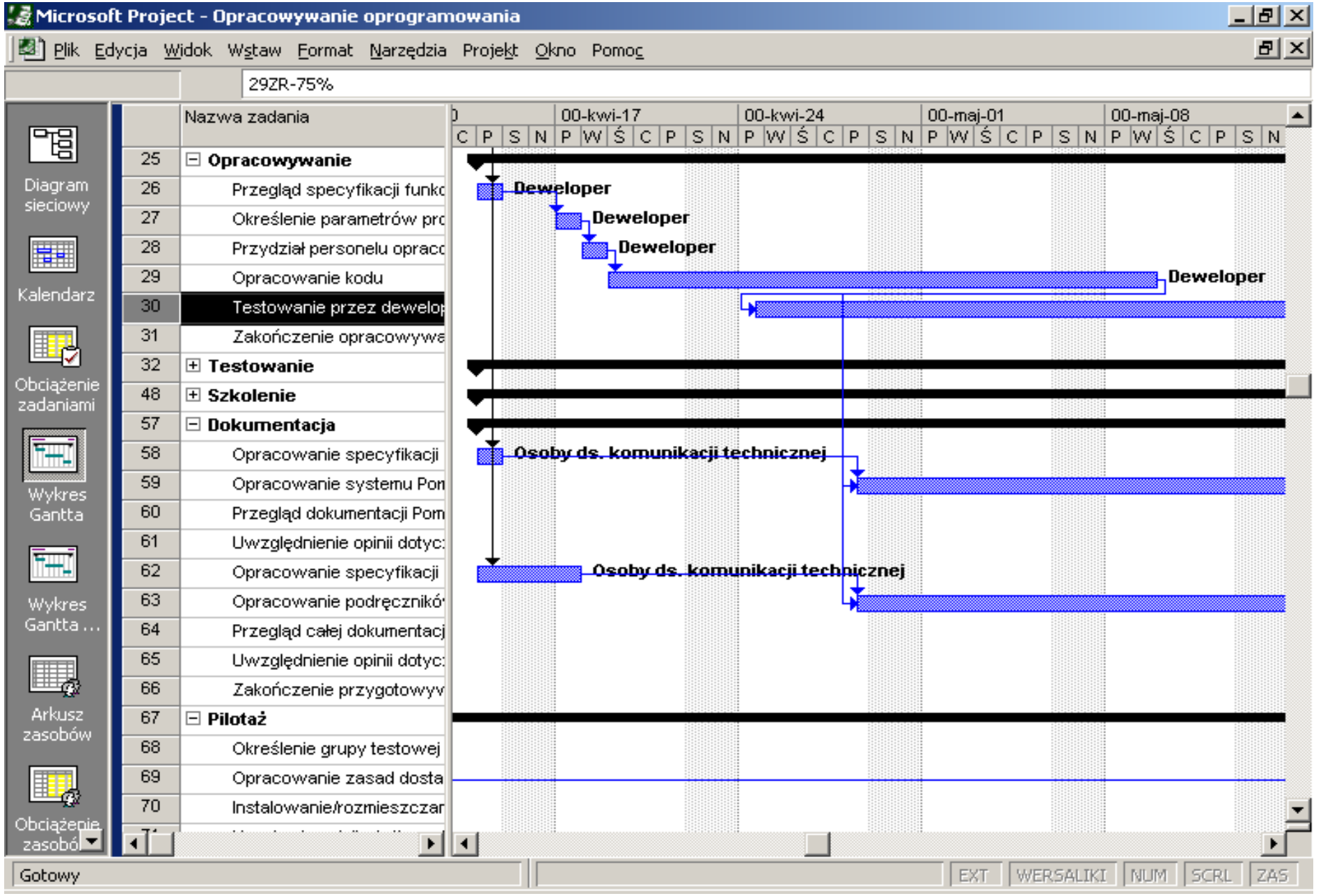
Metody tworzenia harmonogramów prac – schemat (wykład 3)[1LU, 4LU]



Wykres Gantta – ścieżka krytyczna (wykład 3)



Wykres Gantta - zadania zależne (wykład 3)



Zagadnienia

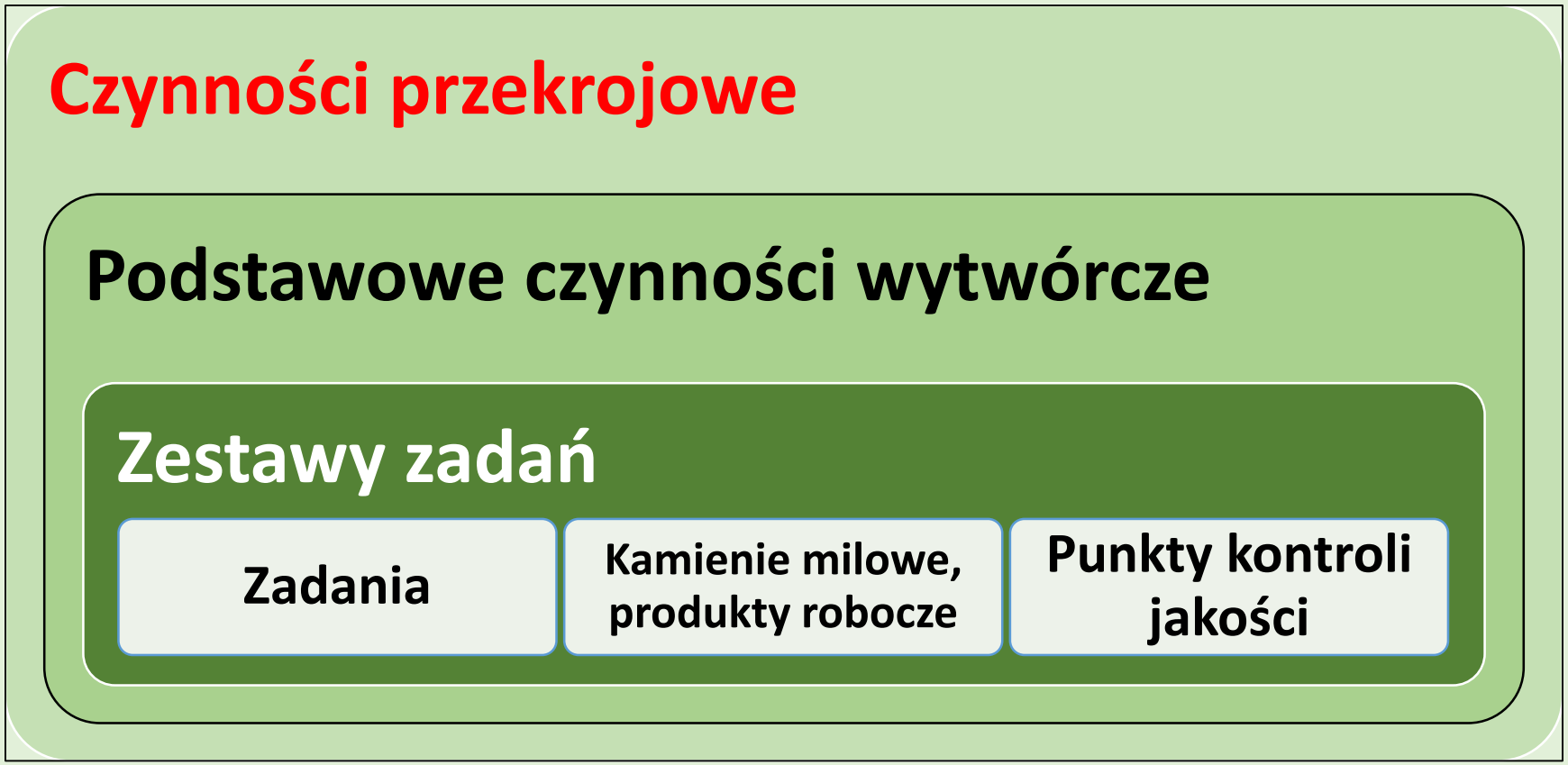
1. **Prowadzenie projektu programistycznego**
2. **Modele tworzenia oprogramowania**

Proces tworzenia oprogramowania [3LU] (wykład 1)

1) **Proces tworzenia oprogramowania** jest definicją kompletnego zbioru aktywności potrzebnych do **odwzorowania wymagań użytkownika w oprogramowanie**
2) **Obejmuje czynniki:**
   - Czynniki organizacyjne
   - Czynniki dziedzinowe
   - Czynniki **cyklu życia oprogramowania**
   - Czynniki techniczne

Proces wytwórczy [1LU]

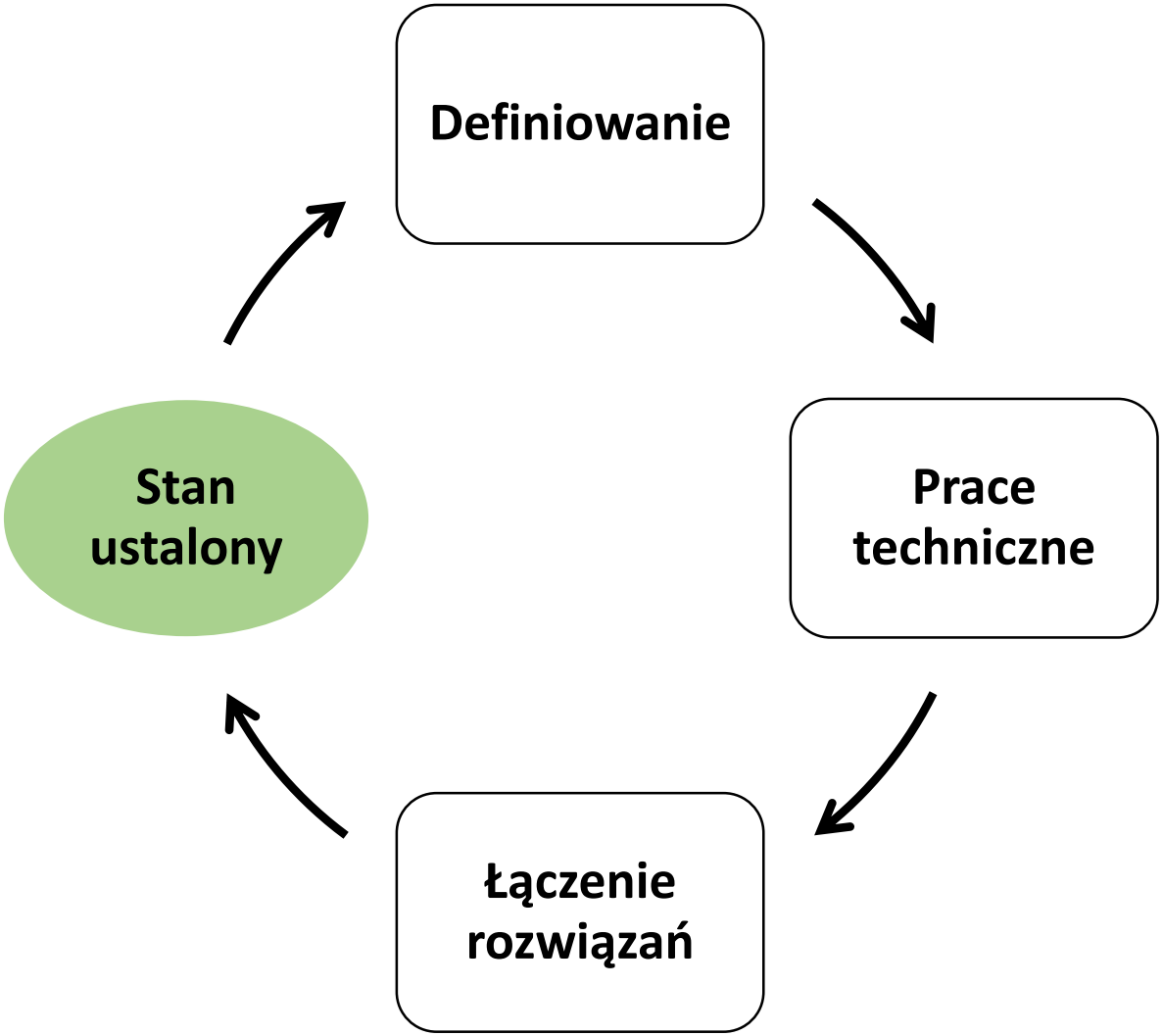
Uniwersalny schemat procesu wytwórczego



Uzupełnienie faz procesu wytwórczego - zestaw czynności przekrojowych, wykonywanych przez cały czas działania projektu

1) Zarządzanie przedsięwzięciem programistycznym i śledzenie jego przebiegu
2) Formalne przeglądy techniczne
3) Zapewniane jakości
4) Zarządzanie konfiguracją
5) Przygotowanie i drukowanie dokumentacji
6) Zarządzanie elementami oprogramowania nadającymi się do wielokrotnego użycia
7) Pomiary
8) Panowanie nad ryzykiem

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



Proces - *model* uniwersalny procesu wytwarzania oprogramowania - czyli *model* cyklu życia oprogramowania [3LU, 2LPW]

<table>
<tr><th>Definiowanie
Modelowanie struktury i dynamiki systemu</th><th colspan="2">Prace techniczne, łączenie rozwiązań
Implementacja systemu, struktury i dynamiki generowanie kodu</th></tr>
<tr><td>Perspektywa koncepcji
co należy wykonać?</td><td>Perspektywa specyfikacji
jak należy używać?</td><td>Perspektywa implementacji
jak należy wykonać?</td></tr>
<tr><td>• model problemu np. przedsiębiorstwa
• wymagania
• analiza (model konceptualny)
• testy modelu</td><td>• projektowanie (model projektowy: architektura sprzętu i oprogramowania; dostęp użytkownika; przechowywanie danych)
• testy projektu</td><td>• programowanie (specyfikacja programu : deklaracje, definicje; dodatkowe struktury danych: struktury „pojemnikowe”, pliki, bazy danych)
• testy oprogramowania
• wdrażanie
• testy wdrażania</td></tr>
</table>

Proces - rozłożenie pracy w czasie (1LU)

- Zasada **40-20-40 (przybliżona)**:
  - Początkowe analizowanie (**10-25%**) wymagań (co zrobić?), projektowanie (**20-25%**) wymagań (jak robić ?)
  - Pisanie kodu (**15-20%**) (jak robić ?)
  - Testowanie i usuwanie błędów (**30-40%**) (poprawa)

Proces - *modele* procesów wytwórczych (1LU)

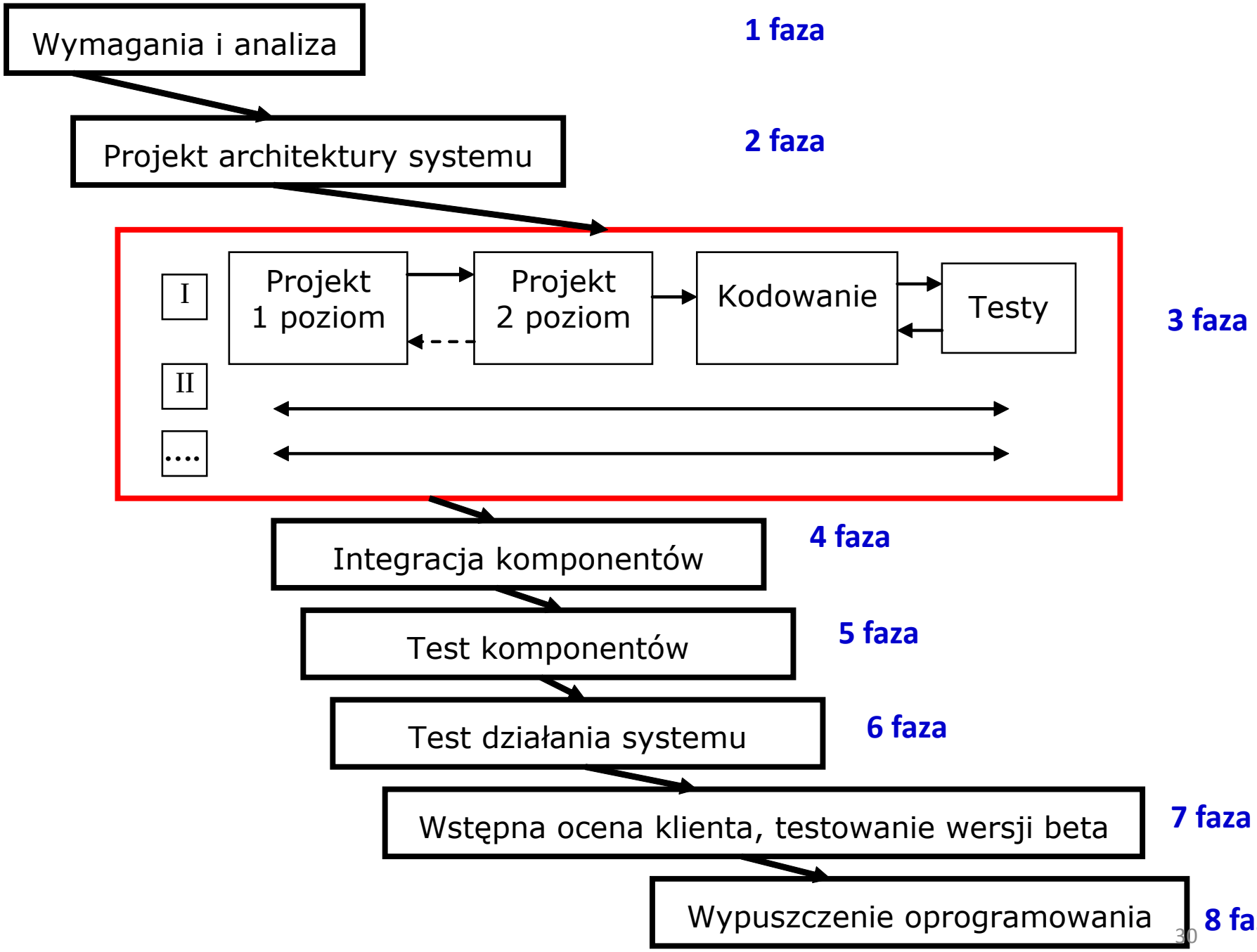
1. Sekwencyjny model liniowy, model kaskadowy
2. Model oparty na prototypowaniu
3. Model szybkiej rozbudowy aplikacji
4. Modele ewolucyjne
   - Model przyrostowy
   - Model spiralny
   - Model spiralny WINWIN
   - Model równoległy
5. Model oparty na metodach formalnych
6. Model oparty na komponentach
7. Techniki czwartej generacji

Zagadnienia

1. **Prowadzenie projektu programistycznego**
2. **Modele tworzenia oprogramowania**
3. **Programowanie kaskadowe i zwinne**

1. Modele procesów tworzenia oprogramowania – proces kaskadowy [2LU]

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


Wymagania i analiza
1 faza
Projekt architektury systemu
2 faza
I
II
....
Projekt
1 poziom
Projekt
2 poziom
Kodowanie
Testy
3 faza
Integracja komponentów
4 faza
Test komponentów
5 faza
Test działania systemu
6 faza
Wstępna ocena klienta, testowanie wersji beta
7 faza
Wypuszczenie oprogramowania
8 faza

2 początkowe fazy kaskadowego cyklu życia tworzenia oprogramowania

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

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

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

Opis realizacji jednego fragmentu

- **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
- **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

5. **Test komponentów**
   - Test interfejsów: użytkownika, międzykomponentowych i wewnętrznych komponentów pod względem poprawności kodu (testy integracyjne) i spełniania wymagań klienta
6. **Test działania systemu**
   - Test akceptacyjny systemu
   - Regresyjny test systemu (test po wprowadzenie zmian),
   - Test wydajności systemu
   - Test funkcjonalny systemu pod kątem łatwości obsługi
7. **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

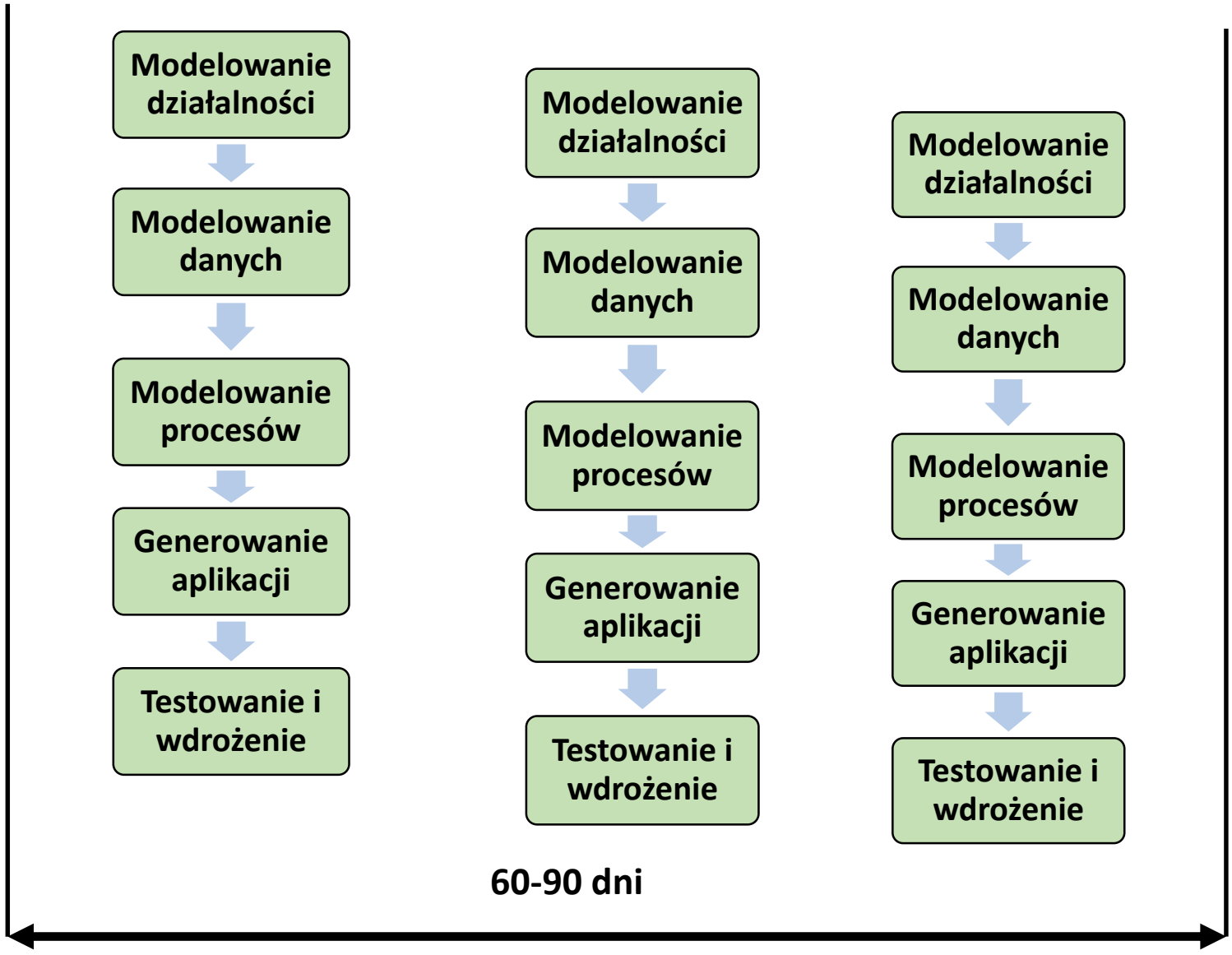
1. Schemat działania: **Wejście –Zadanie – Sprawdzenie – Wyjście** (**Entry-Task-Validation_Exit**)
2. Odpowiedni dla doświadczonych wykonawców i klientów poprawnie definiujących proces biznesowy
3. Ułatwienie prowadzenia dużego, złożonego projektu
4. Podstawowe podejście przy tworzeniu oprogramowania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

Właściwości modelu kaskadowego – „dziel i rządź”: wady

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

2. Modele procesów tworzenia oprogramowania – szybka rozbudowa aplikacji RAD [1LU]

- **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

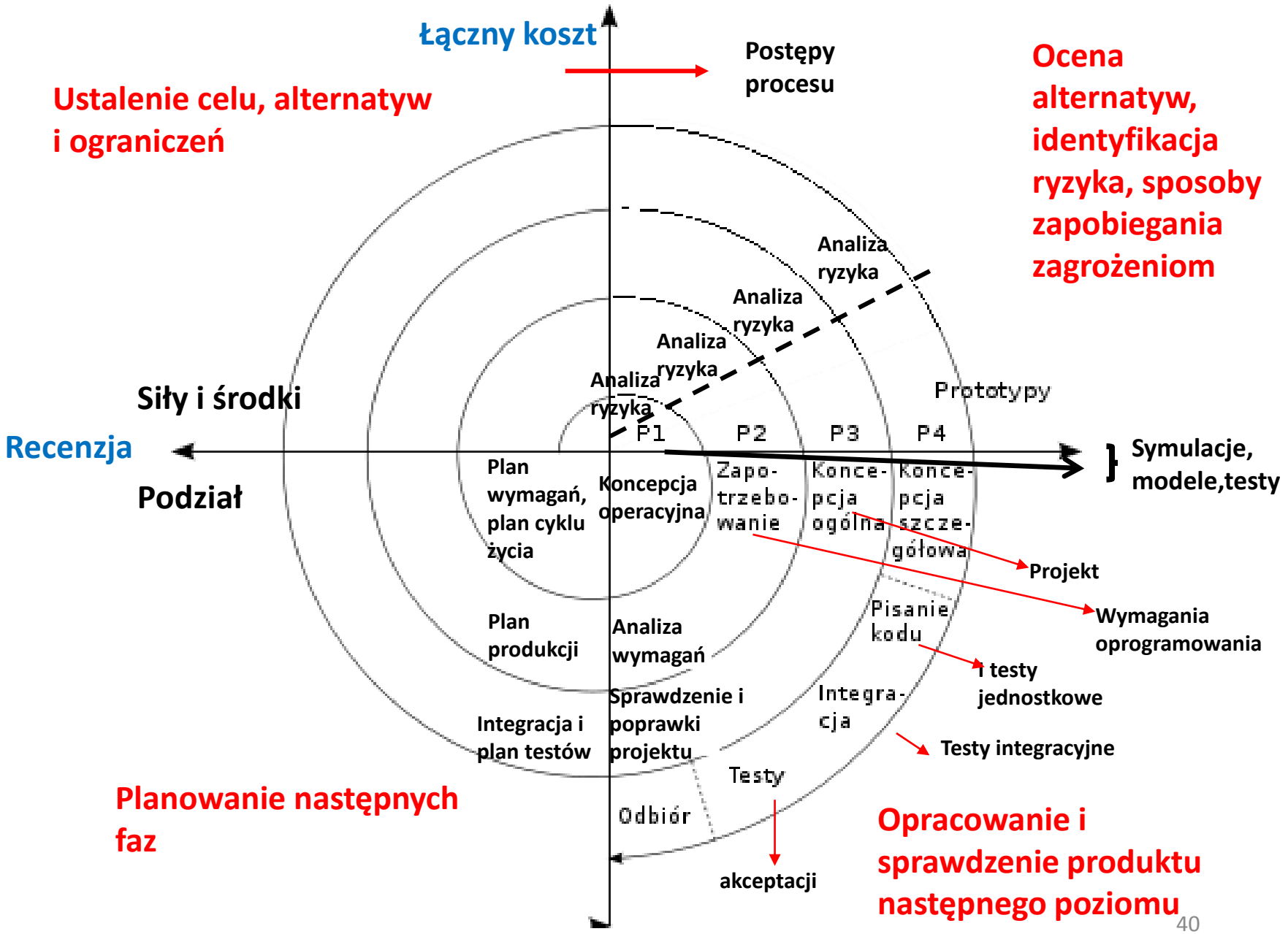
Cechy RAD

- 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

3. Modele procesów tworzenia oprogramowania – proces spiralny [2LU]

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

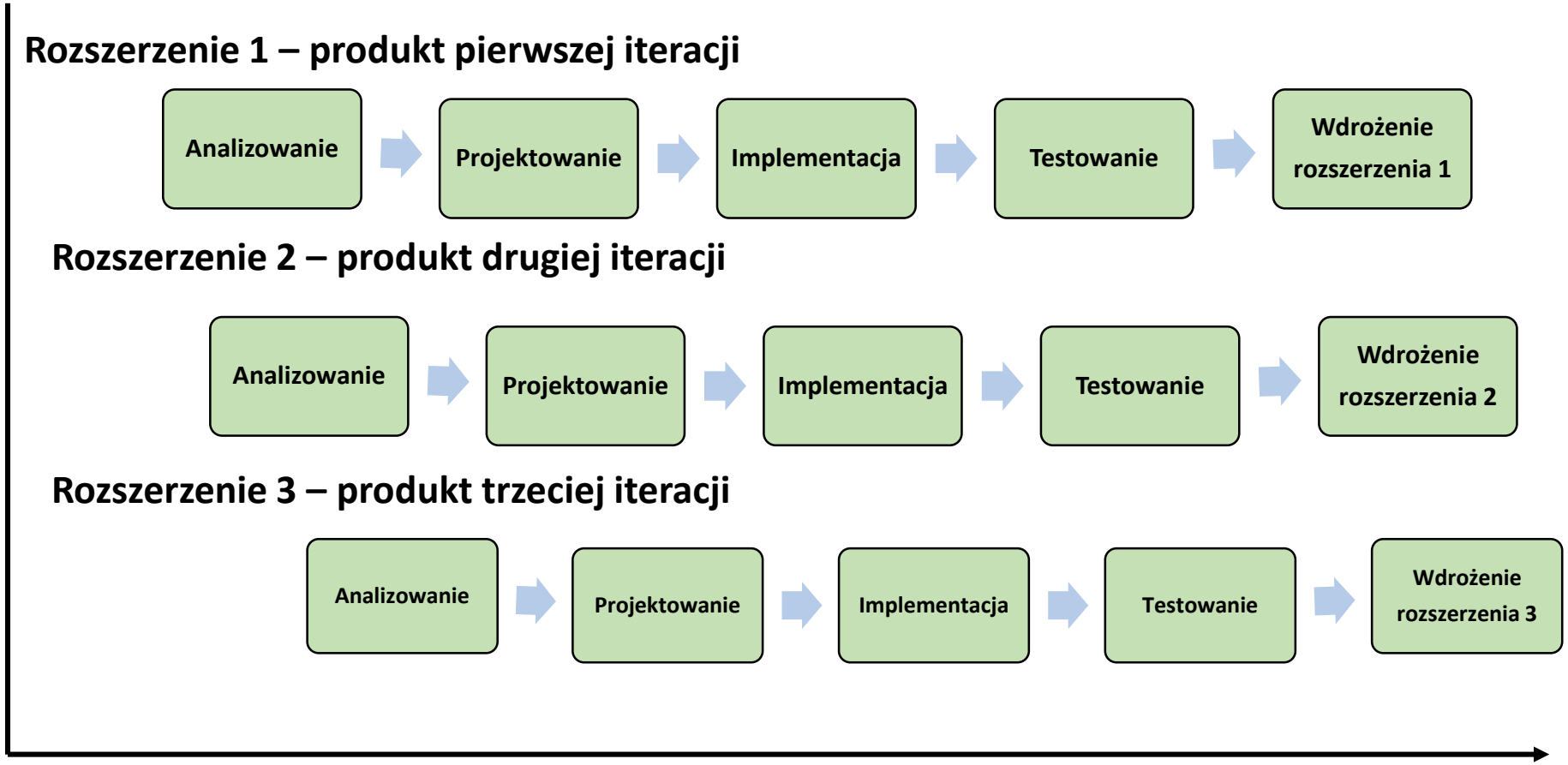
- Trudność z wywiązania się z warunków kontraktu, lepsza w przypadku budowania oprogramowania do sprzedaży
- Duży wpływ analizy ryzyka na przebieg projektowania - błędy w analizie mogą negatywnie wpłynąć na wynik produkcji oprogramowania
- Potrzeba opracowania kolejnych kroków przy zachowaniu spójności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

4. Modele procesów tworzenia oprogramowania – proces iteracyjno-rozwojowy [1LU, 3LU, 2LPW]

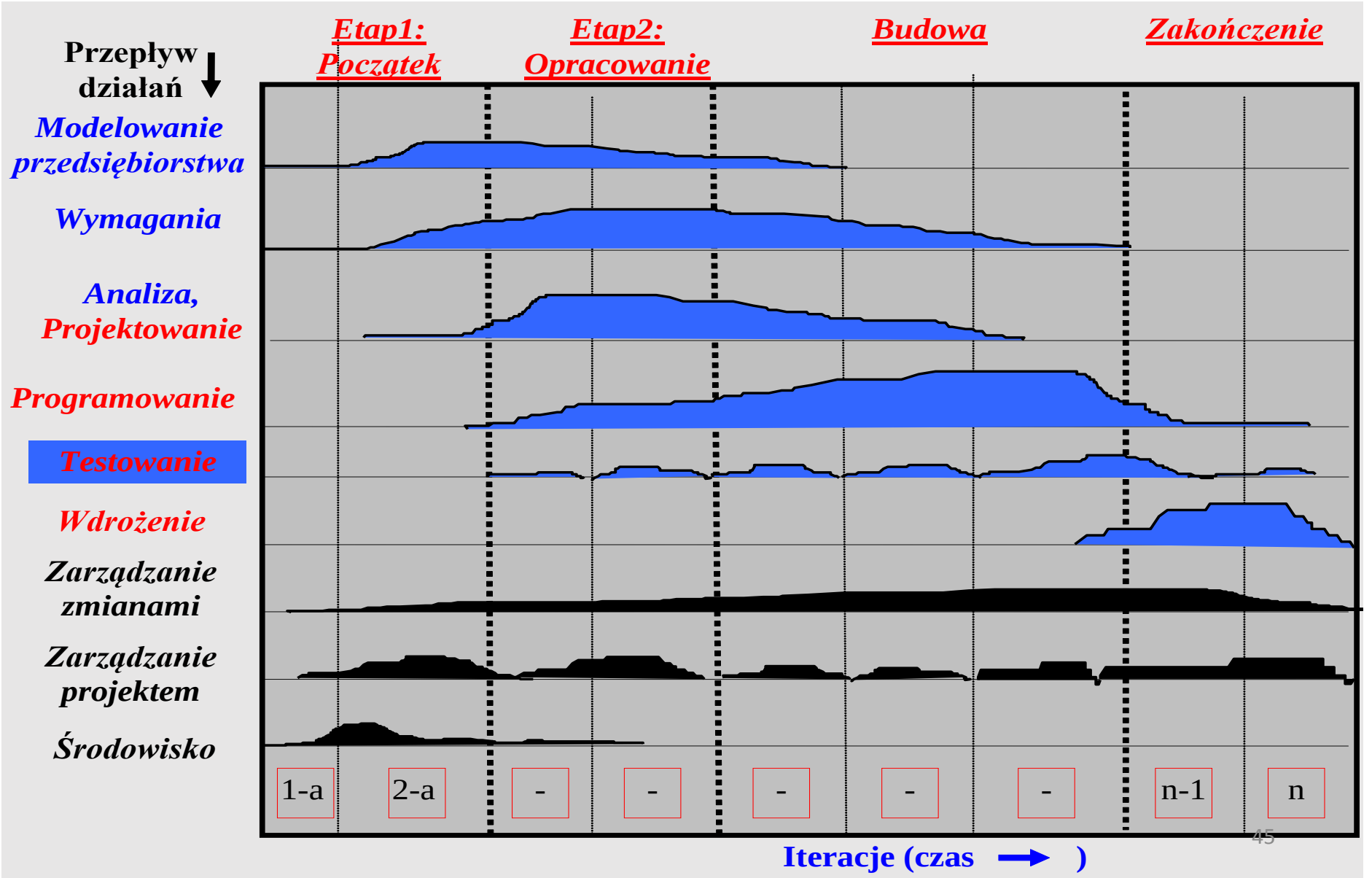
Modele ewolucyjne

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

Ogólna zasada [1LU]
powiązany z czynnościami przekrojowymi (wykład 1)



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



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

Perspektywy rozumienia obiektów – identyfikacji obiektów [2LPW]

- **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 [2LPW]

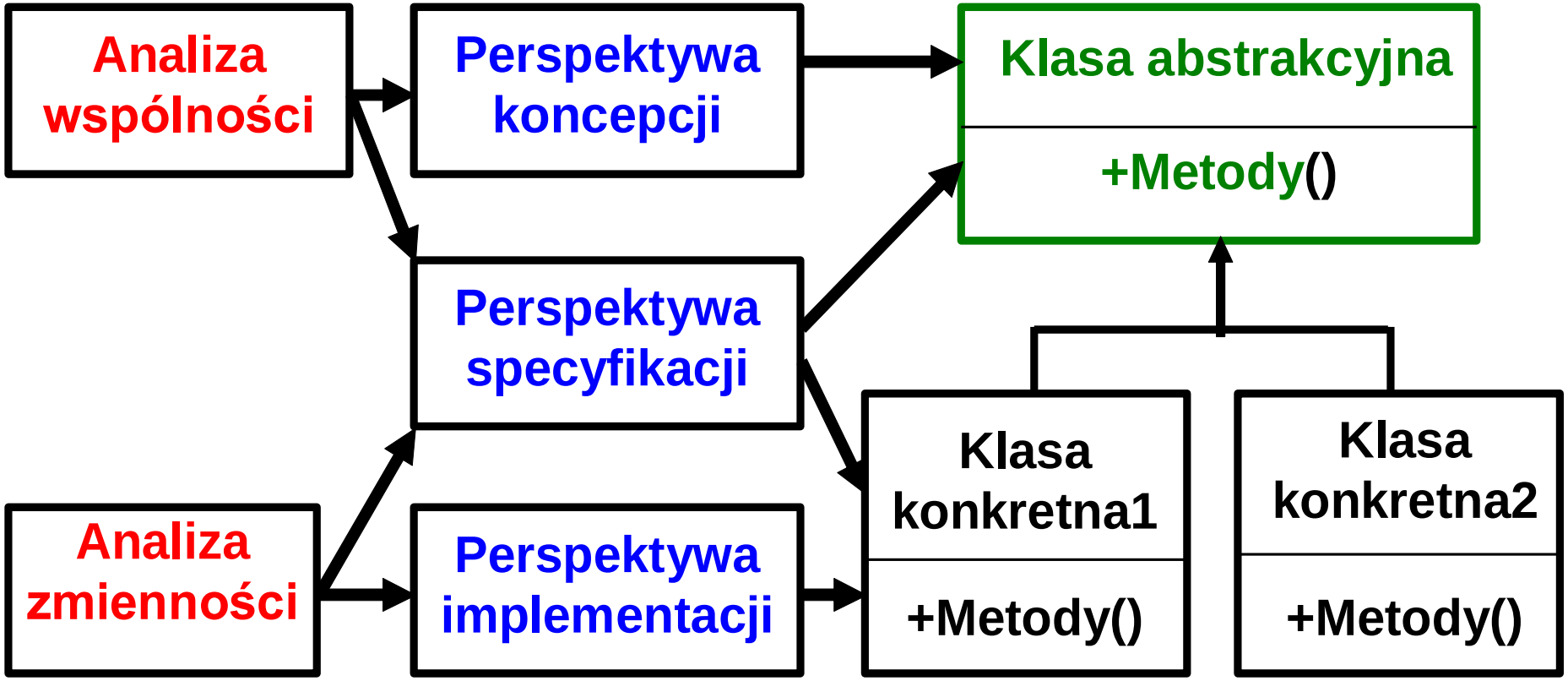
- **Perspektywa tworzenia i zarządzania obiektami**

  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 [2LPW]



Związek między perspektywą specyfikacji, koncepcji i implementacji

Zależności między analizą, projektowaniem i implementacją [2LPW]

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 [2LU]

Fazy: modelowanie przedsiębiorstwa, wymagania, analiza

1. **Modelownie podstawowego procesu**
   - Tworzenie punktu widzenia klienta
   - Modelowanie podstawowych funkcji (przypadki użycia)
   - **Czynność przekrojowa**: opiniowanie wymagań
   - Zdefiniowanie danych potrzebnych do zakończenia projektu
   - **Czynność przekrojowa**: 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 [2LU]

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: sprawdzenie analizy klas
6. Definiowanie interfejsów – perspektywa specyfikacji

   Czynność przekrojowa: przegląd zewnętrznej charakterystyki klas po kroku szóstym

8 kroków procesu iteracyjno-rozwojowego podsumowanie [2LU], [2LPW]

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: sprawdzenie kodu po zakończeniu wszystkich ośmiu kroków2

Model iteracyjno-przyrostowy – cechy [2LU]

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

Model iteracyjno-przyrostowy – wady [2LU]

1. Dodatkowy koszt związany z niezależną realizacją fragmentów systemu
2. Potencjalne trudności z wycinaniem podzbioru funkcji w pełni niezależnych
3. Konieczność implementacji szkieletów (interfejs zgodny z docelowym systemem) – dodatkowy nakład pracy (koszt), ryzyko niewykrycia błędów w fazie testowania

5. Modele procesów tworzenia oprogramowania - programowanie zwinne (*Agile software development*)

Modele ewolucyjne zwinne

- **Proces - programowanie zwinne** (*Agile software development*)
- **Produkt – oprogramowanie obiektowe**

Manifest Zwinnego Tworzenia Oprogramowania z 2001 (http://agilemanifesto.org)

Wytwarzając oprogramowanie i pomagając innym w tym zakresie, odkrywamy lepsze sposoby wykonywania tej pracy.
W wyniku tych doświadczeń przedkładamy:

Ludzi i interakcje ponad procesy i narzędzia.

Działające oprogramowanie ponad obszerną dokumentację.

Współpracę z klientem ponad formalne ustalenia.

Reagowanie na zmiany ponad podążanie za planem.

Doceniamy to, co wymieniono po prawej stronie, jednak bardziej cenimy to, co po lewej.

Kent Beck Mike Beedle Arie van Bennekum Alistair Cockburn	Ward Cunningham Martin Fowler James Grenning Jim Highsmith	Andrew Hunt Ron Jeffries Jon Kern Brian Marick

Zasady kryjące się za Manifestem Zwinnego Wytwarzania Oprogramowania

http://agilemanifesto.org/iso/pl/principles.html

Kierujemy się następującymi zasadami:

1. *Najważniejsze dla nas jest zadowolenie Klienta wynikające z wcześnie rozpoczętego i ciągłego dostarczania wartościowego oprogramowania.*
2. *Bądź otwarty na zmieniające się wymagania nawet na zaawansowanym etapie projektu. Zwinne procesy wykorzystują zmiany dla uzyskania przewagi konkurencyjnej Klienta.*
3. *Często dostarczaj działające oprogramowanie od kilku tygodni do paru miesięcy, im krócej tym lepiej z preferencją krótszych terminów.*
4. *Współpraca między ludźmi biznesu i programistami musi odbywać się codziennie w trakcie trwania projektu.*
5. *Twórz projekty wokół zmotywowanych osób. Daj im środowisko i wsparcie, którego potrzebują i ufaj im, ze wykonają swoją pracę.*
6. *Najwydajniejszym i najskuteczniejszym sposobem przekazywania informacji do i w ramach zespołu jest rozmowa twarzą w twarz .*

7. *Podstawową i najważniejszą miarą postępu jest działające oprogramowanie.*
8. *Zwinne procesy tworzą środowisko do równomiernego rozwijania oprogramowania. Równomierne tempo powinno być nieustannie utrzymywane poprzez sponsorów, programistów oraz użytkowników.*
9. *Poprzez ciągłe skupienie na technicznej doskonałości i dobremu zaprojektowaniu oprogramowania zwiększa zwinność.*
10. *Prostota – sztuka maksymalizacji pracy niewykonanej – jest zasadnicza.*
11. *Najlepsze architektury, wymagania i projekty powstają w samoorganizujących się zespołach.*
12. *W regularnych odstępach czasu zespół zastanawia się jak poprawić swoją efektywność, dostosowuje lub zmienia swoje zachowanie.*

Metodyka typu Agile - Scrum

Podstawowe fazy, etapy

- **sprint** (iteracja – (2-4 tygodnie),
- ***scrum meeting*** (spotkanie codziennie 15 min),
- ***sprint review*** *(podsumowanie sprintu)*)

Podstawowe zadanie metodyki:

- dostarczaniu kolejnych, coraz bardziej dopracowanych produktów,
- włączaniu się przyszłych użytkowników w proces wytwórczy,
- samoorganizacji zespołu wykonawców.

Role główne

- **Zespół** - 5-9 osób, zaangażowana w tworzenie oprogramowania
- **Właściciel produktu** (**Product owner**) - osoba reprezentująca klienta, która może należeć do zespołu
- **Kierujący zespołem** **(Scrum master)** – osoba ułatwiająca działania zespołu

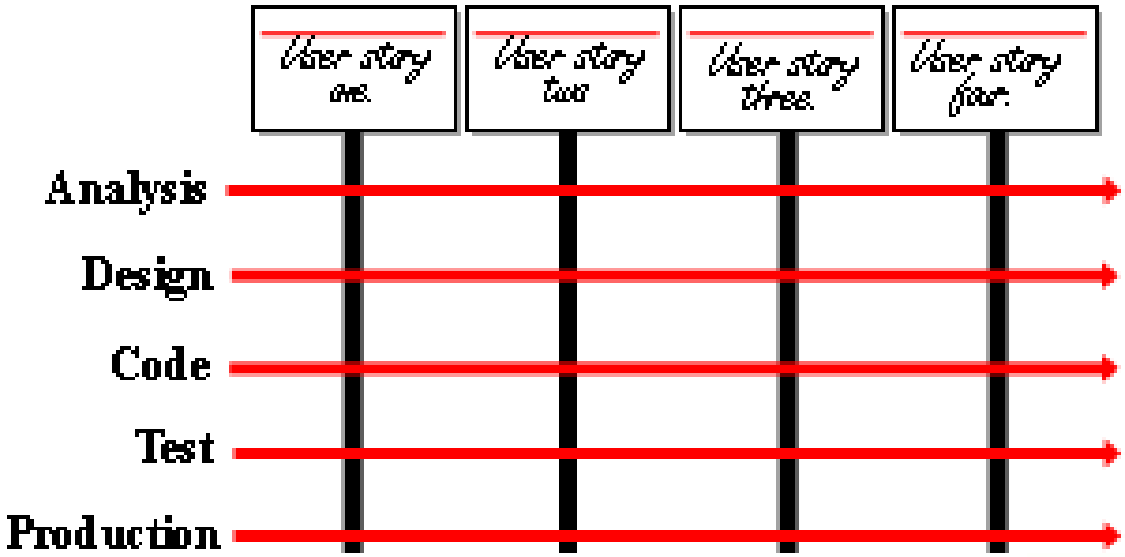
6. Modele procesów tworzenia oprogramowania - XP(Extreme Programming)

Modele ewolucyjne zwinne

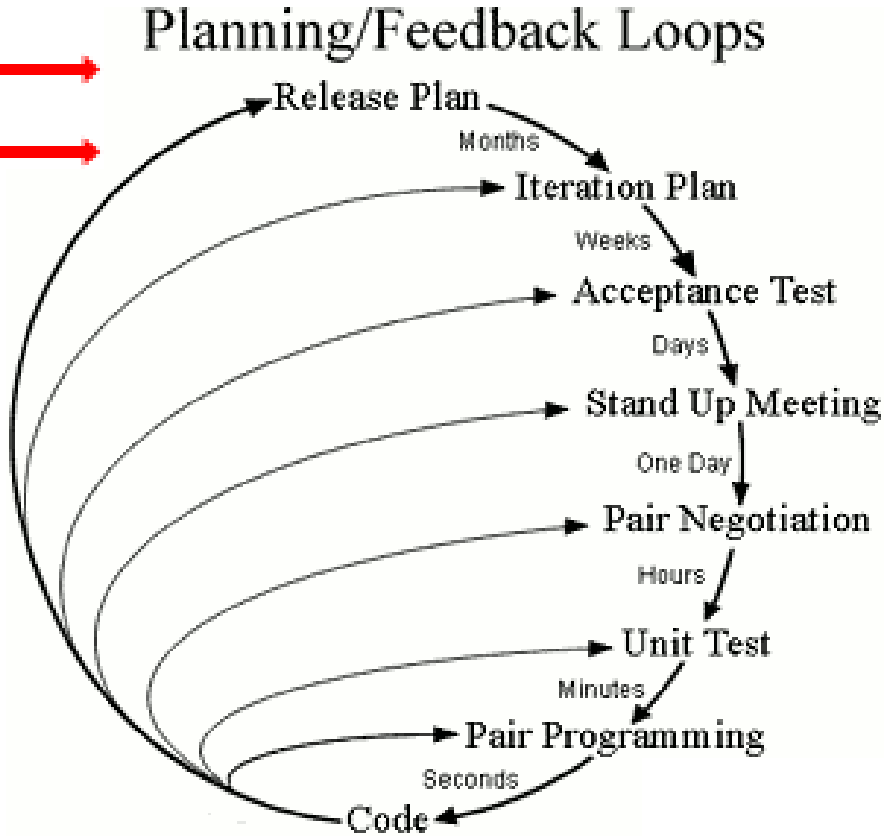
- **Proces** - programowanie ekstremalne XP(Extreme Programming) - praktyki XP
- **Produkt** - oprogramowanie obiektowe

XP podejście do iteracyjno-rozwojowego tworzenia oprogramowania

http://www.agile-process.org

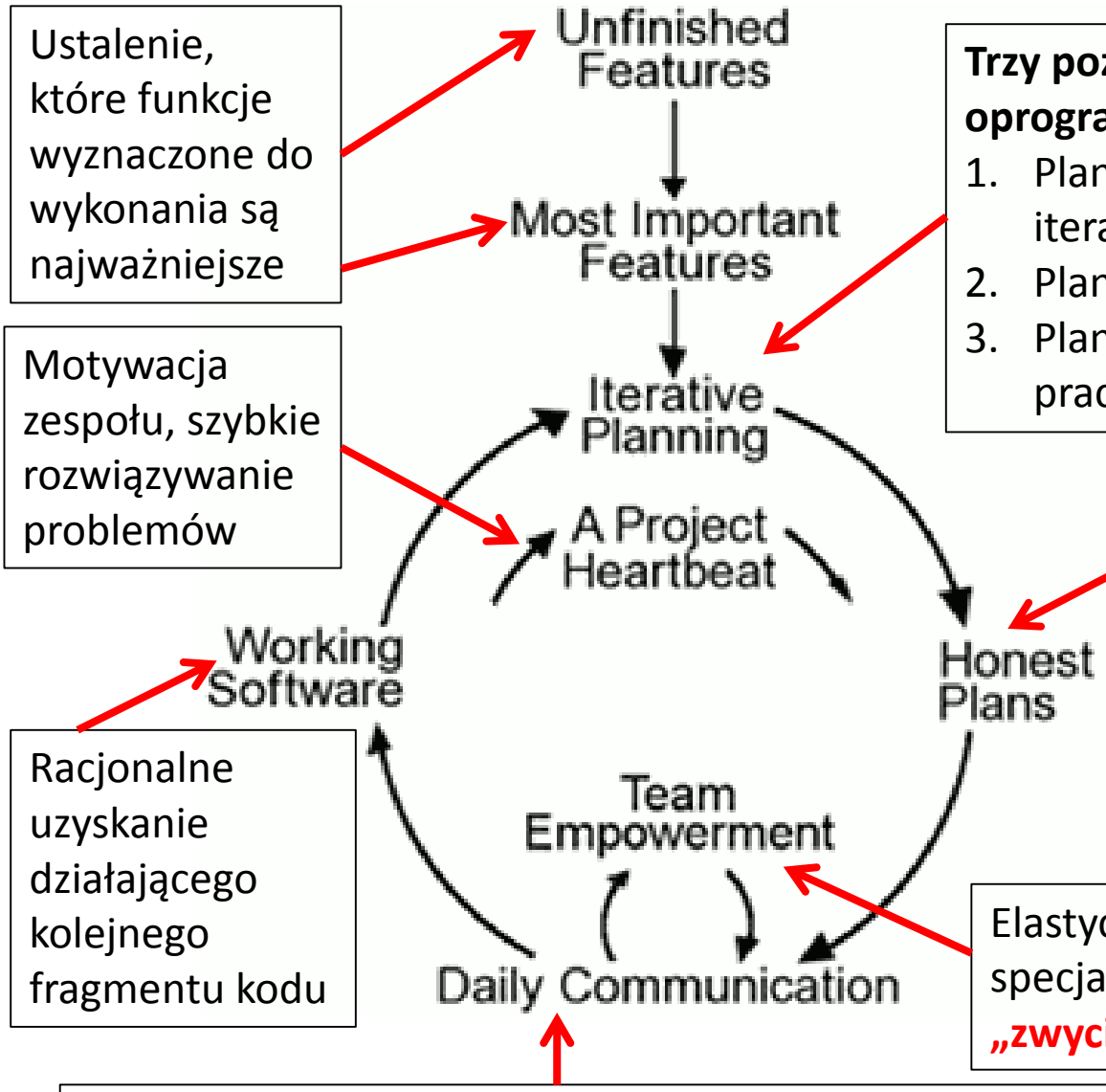


Struktura spiralnej komunikacji w zespole – ważny jest wybór właściwej drogi wynikającej z zaawansowania projektu



Cykl życia oprogramowania XP - http://www.agile-process.org

Ustalenie, które funkcje wyznaczone do wykonania są najważniejsze



Trzy poziomy planowania (rozwoju oprogramowania):

1. Planowanie wydania obejmujące kilka iteracji
2. Planowanie pojedynczej iteracji (sprint)
3. Planowanie obejmujące 1-dniowy zakres pracy (fragment iteracji)

Motywacja zespołu, szybkie rozwiązywanie problemów

Rzetelny plan obejmujący zakres prac w ciągu 1 tygodnia (SCRUM – 30 dni). Plan obejmuje elastyczne wykorzystanie modelowania, projektowania i implementacji oraz możliwości wykonawców

Racjonalne uzyskanie działającego kolejnego fragmentu kodu

Elastyczne zarządzanie zespołem (sieciowe, specjalistyczne, nieegoistyczne) – **„zwycięstwo publiczne”**

Wynika z postępu prac określonych w planach jednodniowych – komunikacja między uczestnikami projektu

Praktyki XP [2LU]

1. Planowanie:
   1.1. zespół planuje czas i budżet,
   1.2. zespół planuje ryzyko
   1.3. zespół planuje kolejność opowiadań (opis działania systemu wykonany przez klienta lub wykonawcę)
   1.4. klient definiuje:
      - Zakres działań
      - Terminy ukończenia wersji
      - priorytety
2. Metafora systemu – sposób działania systemu
3. Prosty projekt – do testowania zakresu działań
4. Programowanie parami – przy jednej stacji roboczej (cały zespół około 10 osób)
5. Testy jednostkowe i akceptacji – przed i po wykonaniu właściwego kodu

Praktyki XP [2LU]

6. Refaktoryzacja – w ciągu tworzenia i rozwijania kodu
7. Wspólny kod - przez parokrotne przypisywanie zespołom różnych zadań związanych z innymi zadaniami, każdy wykonawca zna obraz całego kodu
8. Ciągła integracja kodu
9. Przedstawiciel klienta jako członek wykonawców (testy akceptacji, ekspert domen)
10. 40-godzinny tydzień pracy – ciągła aktywność zespołu wykonawców
11. „Małe wersje”: tworzone są wersje niewielkie z przydatnymi funkcjami
12. Standardy kodowania – najpierw definiowane, a potem stosowane

XP - cechy

1. Poszczególne elementy wzajemnie się wspierają
2. Zachowanie kontroli nad procesem
3. Ogranicza wstępne zbieranie danych o wymaganiach
4. Ogranicza analizę i modelowanie projektu
5. Ogranicza planowanie na rzecz późniejszej elastyczności – ogranicza liczbę klas i dokumentacji
6. Wydajne tworzenie małych i średnich "projektów wysokiego ryzyka” oparte na synergii stosowania rozmaitych praktyk zapewniające eliminację wad i wykorzystania zalet tych praktyk.

XP - kwestie kontrowersyjne

1. Brak dokładnej specyfikacji.
2. Stałe angażowanie strony klienta.
3. Zbyt swobodne zmiany kodu