Modelowanie i analiza obiektowa

Wykład 4

Analiza i projektowanie - przegląd

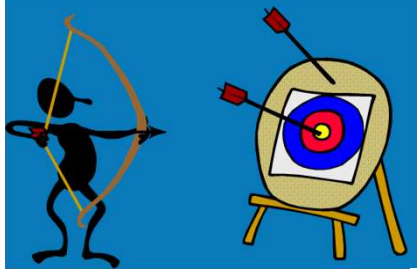
Plan wykładu

- Cele analizy i projektowania
- Proces analizy a proces projektowania
- Architektura oprogramowania
- Czynności wykonywane w fazie analizy i projektowania
- Realizacja przypadków użycia

Cele analizy i projektowania

Cele analizy i projektowania

- Transformacja wymagań w projekt systemu
- Wykreowanie trwałej architektury systemu
- Adaptacja projektu dla konkretnego środowiska implementacyjnego
- Zaprojektowanie wydajności



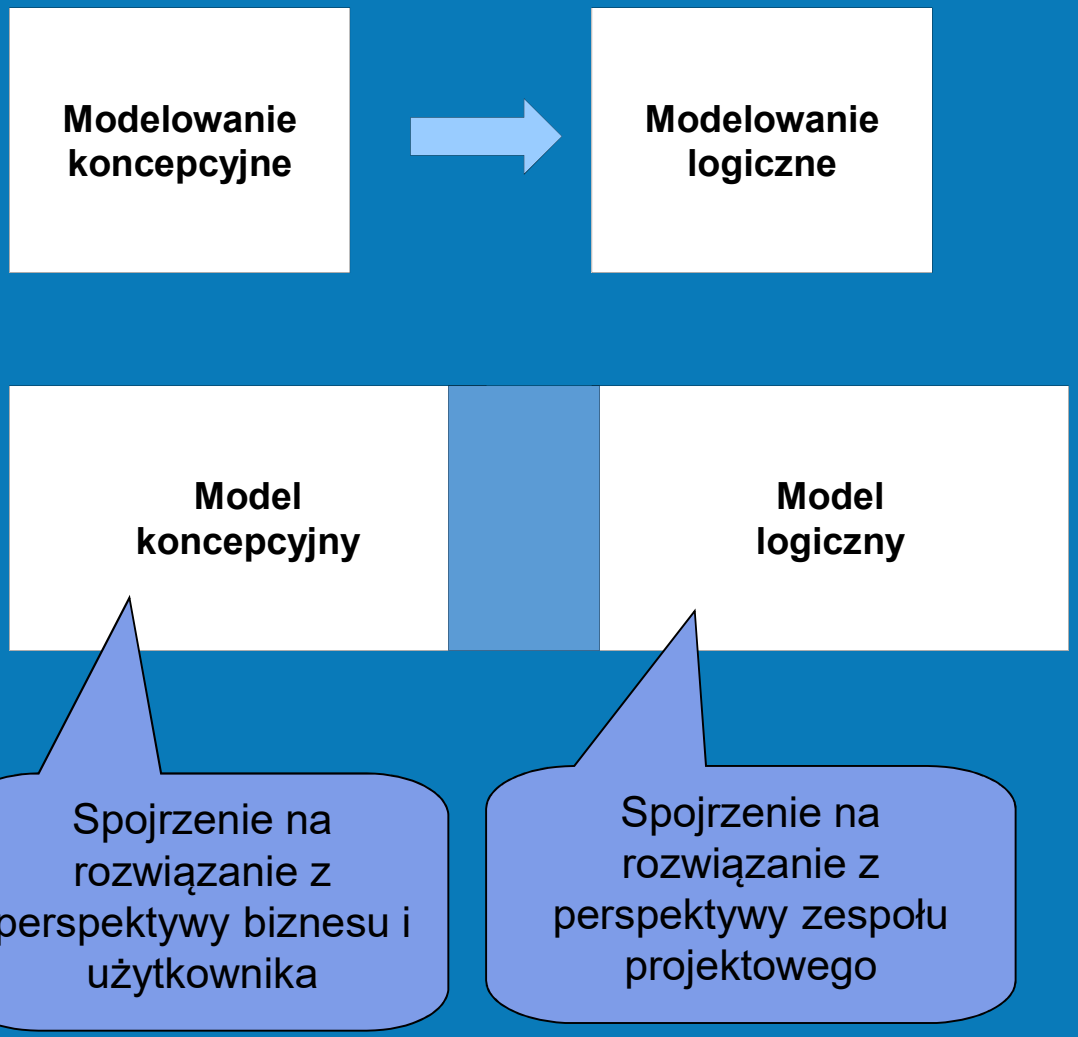
Rodzaje modeli

Modelowanie koncepcyjne

Model koncepcyjny

Spojrzenie na rozwiązanie z perspektywy biznesu i użytkownika

Rodzaje modeli



Rodzaje modeli

Analiza

Projektowanie

Modelowanie koncepcyjne → **Modelowanie logiczne** → **Modelowanie fizyczne**

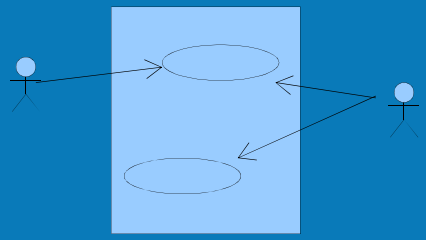
Model koncepcyjny | **Model logiczny** | **Model fizyczny**

Spojrzenie na rozwiązanie z perspektywy biznesu i użytkownika

Spojrzenie na rozwiązanie z perspektywy zespołu projektowego

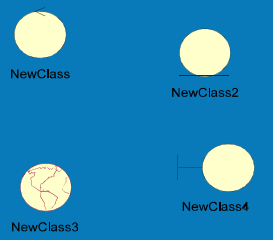
Spojrzenie na rozwiązanie z perspektywy programisty, wdrożeniowca

Transformacja modelowania punkt widzenia programisty

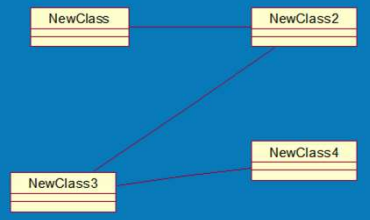


Model przypadków użycia

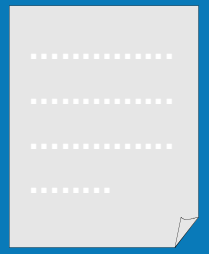
Analiza i projektowanie



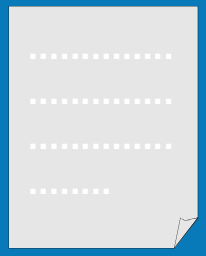
Model analizy



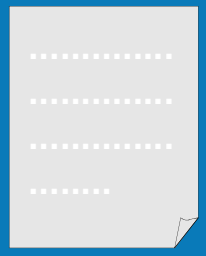
Model projektowy



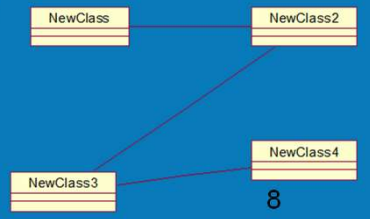
Dokument architektury



Słownik



Wymagania uzupełniające



Model danych

Analiza

- Proces analizy nazywany jest również modelowaniem logicznym.
- Obrazujemy pewien wycinek rzeczywistości w terminach modelu (obiektowego).
- Faza analizy nie zajmuje się szczegółową specyfikacją atrybutów i operacji, widzi klasę jako pewną abstrakcję zachowania.
- W zasadzie analiza nie powinna stawiać nacisku na zmianę rzeczywistości poprzez wprowadzenie tam nowych elementów np. w postaci systemu komputerowego.
- Jej celem jest rozpoznanie wszystkich tych aspektów dziedziny problemowej, które mogłyby być poddane procesowi informatyzacji.

Projektowanie

- Celem fazy projektowania (modelowanie fizyczne) jest opracowanie szczegółowego planu implementacji systemu.
- W odróżnieniu od analizy, w projektowaniu dużą rolę odgrywa środowisko implementacji. Projektanci muszą posiadać dobrą znajomość języków, bibliotek i narzędzi stosowanych w trakcie implementacji.
- Dąży się, by struktura projektu zachowywała strukturę modelu stworzonego w poprzedniej fazie (analizie) - podejście bezszwowe (*seamless*). Niewielkie zmiany w dziedzinie problemowej powinny implikować niewielkie zmiany w projekcie.

Projektowanie

Zadania fazy projektowania:

- Uszczegółowienie wyników analizy. Projekt musi być wystarczająco szczegółowy aby mógł być podstawą implementacji.
- Stopień szczegółowości zależy od poziomu zaawansowania programistów.
- Dostosowanie projektu do ograniczeń i możliwości środowiska implementacji.
- Określenie fizycznej struktury systemu.

Analiza a projektowanie

Analiza (modelowanie logiczne)

- Skupia się na zrozumieniu problemu
- Projekt wyidealizowany
- Opisuje zachowanie
- Określa wymagania funkcjonalne
- W wyniku powstaje „mały model”

Projektowanie (modelowanie fizyczne)

- Skupia się na zrozumieniu rozwiązania
- Określa operacje i atrybuty
- Stawia nacisk na wydajność
- Określa cykl życia obiektu
- Określa wymagania niefunkcjonalne
- W wyniku powstaje „duży model”

Projektowanie jako proces uszczegóławiania analizy w procesie projektowania

Projektowanie – uszczegóławianie wyników analizy

Przekształcenie reguł odwzorowania notacji dla danych (notacji stosowanej w modelu pojęciowym - wyniku fazy analizy) w struktury języka programowania, który będzie wykorzystany do implementacji systemu.

Projektowanie – uszczegóławianie wyników analizy

- Uszczegóławianie metod:
- Podanie pełnych sygnatur metod (wyspecyfikowanie typu argumentów oraz wartości zwracanej dla każdej metody).
- Zastąpienie niektórych prostych metod publicznym dostępem do atrybutów, np. metod *PobierzNazwisko*, *UstawNazwisko*, etc. (!!!)
- Zastąpienie niektórych atrybutów redundantnych (pochodnych) przez odpowiednie metody, np.
  - *Wiek = BieżącaData - DataUrodzenia;*
  - *KwotaDochodu = KwotaPrzychodu - KwotaKosztów;*

Projektowanie – uszczegóławianie wyników analizy

Określenie sposobów implementacji asocjacji:

- Asocjacje można zaimplementować na wiele sposobów, z reguły poprzez wprowadzenie dodatkowych atrybutów. Mogą one być następujące:
  - obiekt (lub kolekcja obiektów) powiązanej klasy,
  - wskaźniki (referencje) do obiektów powiązanej klasy,
  - identyfikatory obiektów powiązanej klasy,
  - klucze kandydujące obiektów powiązanej klasy.
- W zależności od przyjętego sposobu oraz od liczności związków (1:1, 1:n, m:n) możliwe są bardzo różne deklaracje w przyjętym języku programowania (np. tablica obiektów, lista wskaźników, tablica wskaźników)

Projektowanie – uszczegóławianie wyników analizy

Dane z analizy

Realizacja w java

Adres
- Ulica
- Numer domu
- Numer mieszkania
- Miasto
- Kod

```
class Adres {
private String ulica;
       private int numerDomu;
private int numerMieszkania;
private String miasto;
private String kod;
//…
}
```

Dane osobowe
- Imię
- Nazwisko
- Adres

```
class DaneOsobowe {
private String imie;
private String nazwisko;
private Adres adres;
//…
}
```

Architektura oprogramowania

Architektura oprogramowania - *powtórzenie*

- Zbiór istotnych decyzji związanych z organizacją oprogramowania.
- Wybór elementów strukturalnych i ich interfejsów
- Współpraca pomiędzy tymi elementami dająca w wyniku zachowanie systemu
- Grupowanie elementów strukturalnych w podsystemy





Reguły odkrywania architektury oprogramowania

- Architektura podobnych systemów powinna być również podobna.
- Stworzenie z elementów pewnej spójnej formy architektury może odbywać się przy pomocy stworzonych wcześniej wzorców.





Reguły odkrywania architektury oprogramowania

- Nie ma wyraźnej granicy pomiędzy architekturą a projektem.
- Ogólny (najwyższy) poziom projektu

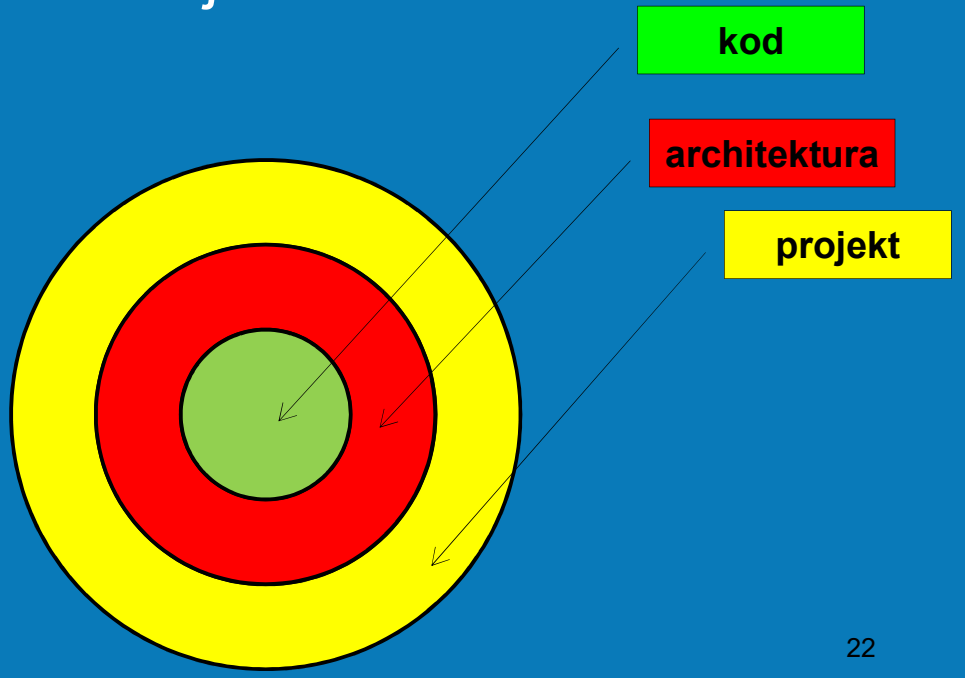




Architektura jest ograniczeniem

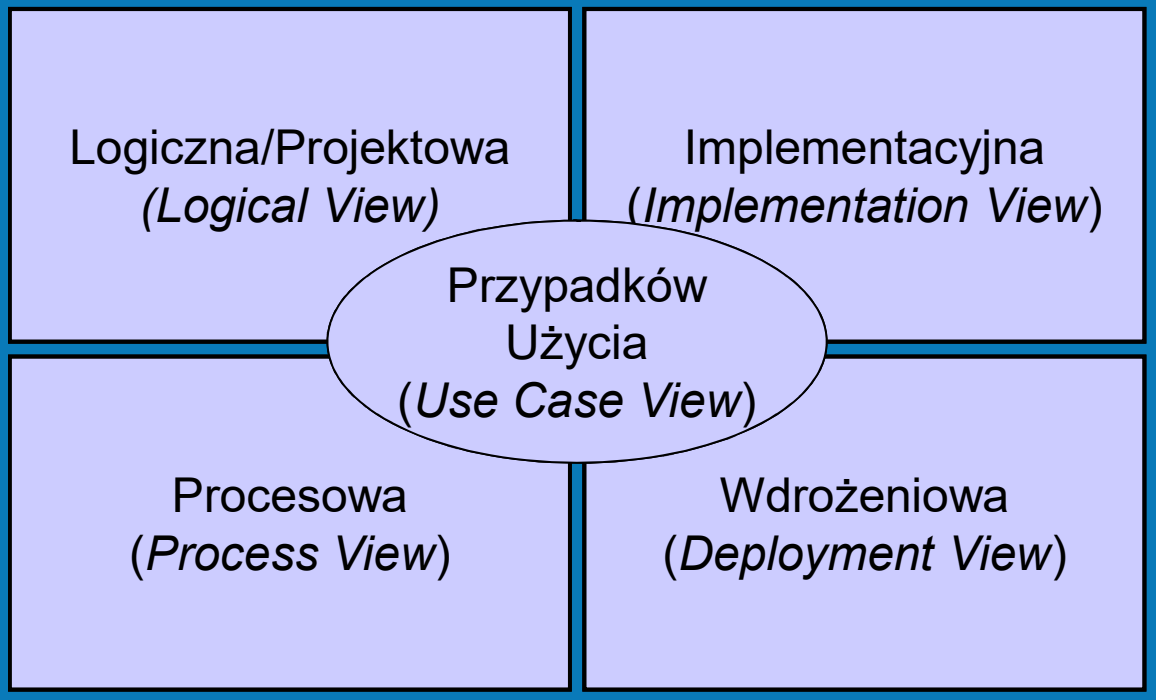
Określanie architektury to podejmowanie strategicznych decyzji, definiowanie reguł i wzorców ograniczających proces projektowania i implementacji

Decyzje związane z architekturą są fundamentem



Perspektywy (model „4+1”)

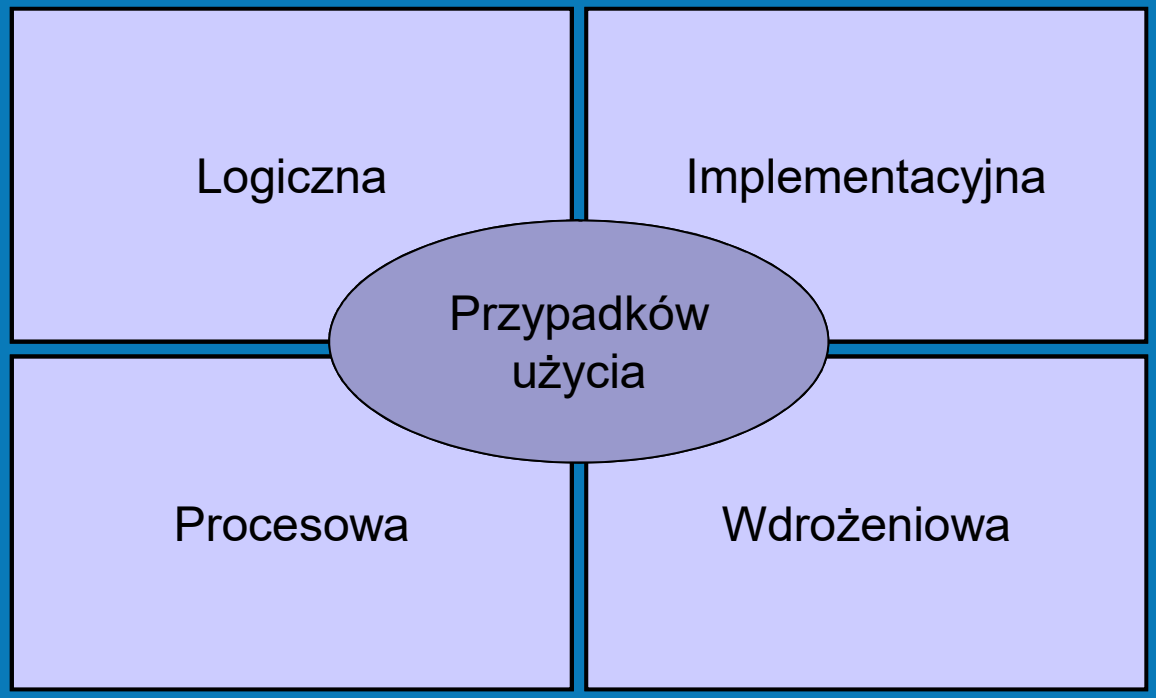
Architektura może być prezentowana z różnych punktów widzenia...



Źródło modelu 4+1 wykorzystywanego w RUP.
Philippe Kruchten 1995, "The 4+1 view model of architecture," IEEE Software. 12(6), November 1995.

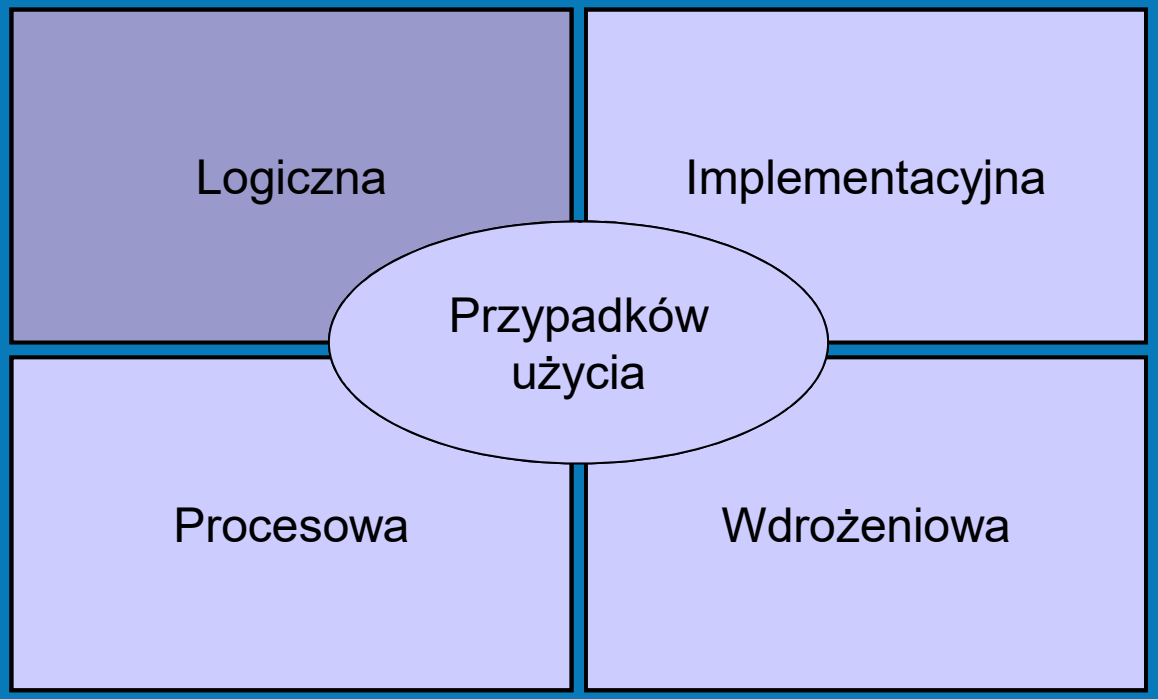
Architektura widziana z perspektywy „Przypadków Użycia”

Podzbiór modelu przypadków użycia obejmujący istotne z punktu widzenia architektury systemu przypadki użycia i scenariusze.



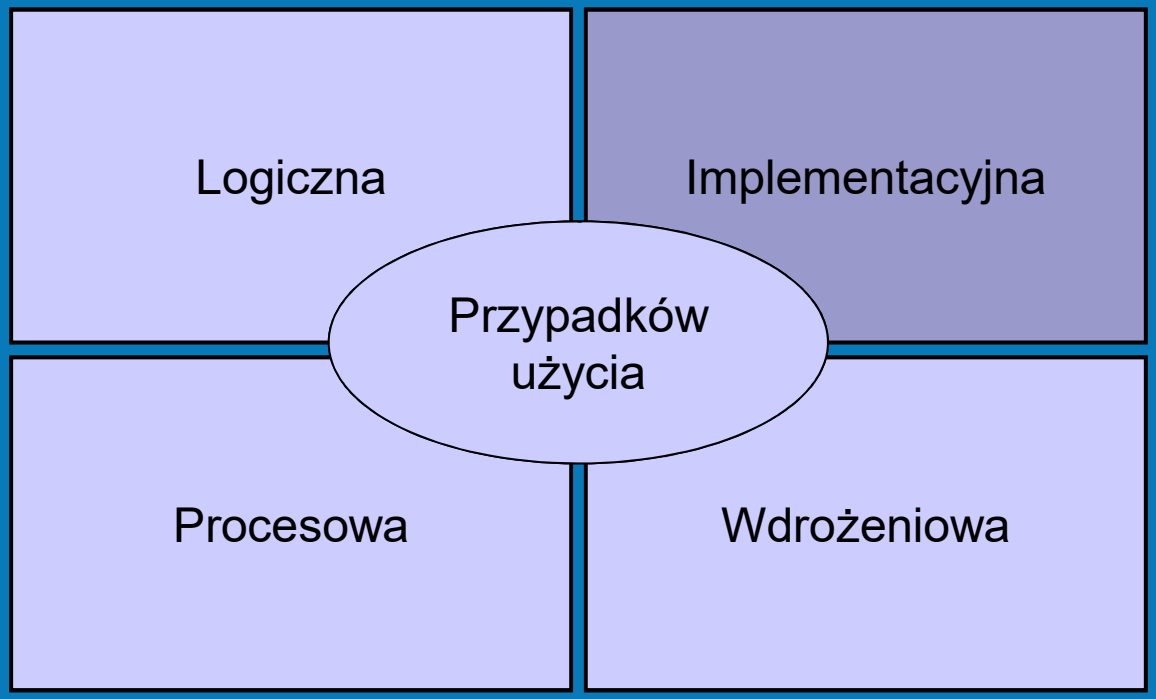
Architektura widziana z perspektywy modelu logicznego

Zawiera najważniejsze klasy projektowe pogrupowane w pakiety i podsystemy. Pakiety i podsystemy są zorganizowane w warstwy (np. podsystem aplikacji, podsystem biznesowy, middleware, oprogramowanie systemowe)



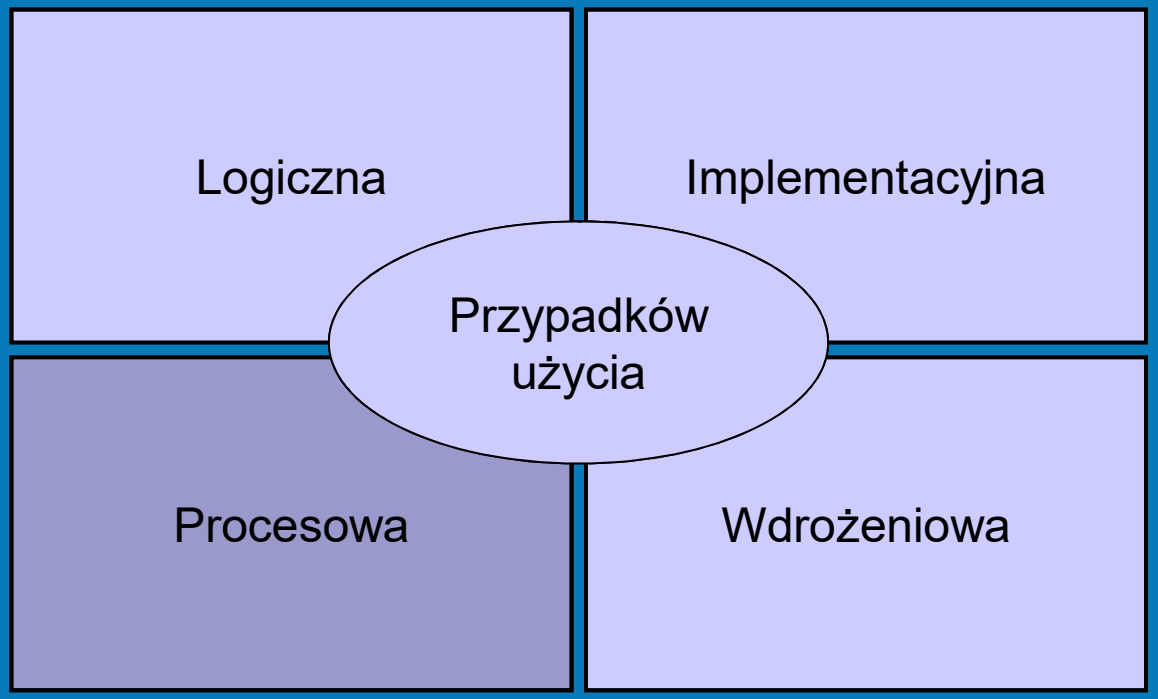
Architektura widziana z perspektywy implementacji

Opisuje organizacje modułów oprogramowania w pakietach. Opisuje również powiązania pomiędzy pakietami i klasami z modelu logicznego a pakietami i modułami z modelu implementacyjnego.



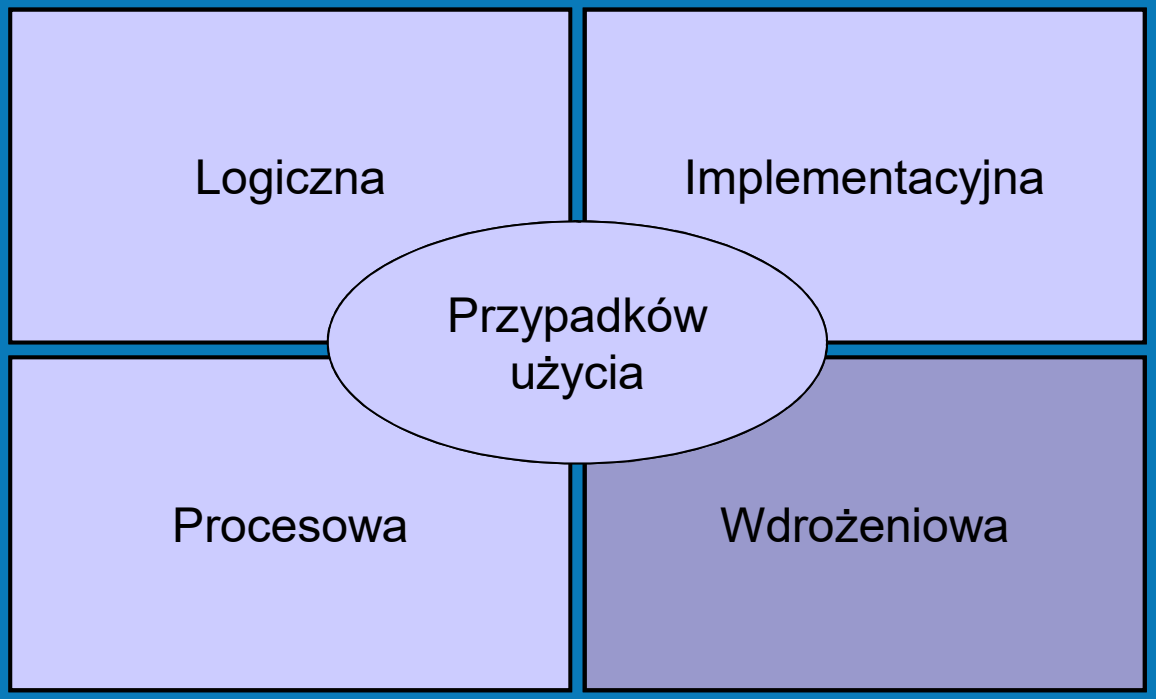
Architektura widziana z perspektywy procesów i wątków

Opis procesów i wątków w terminach ich konfiguracji i i współdziałania oraz przypisania obiektów i klas do konkretnego procesu bądź wątku.



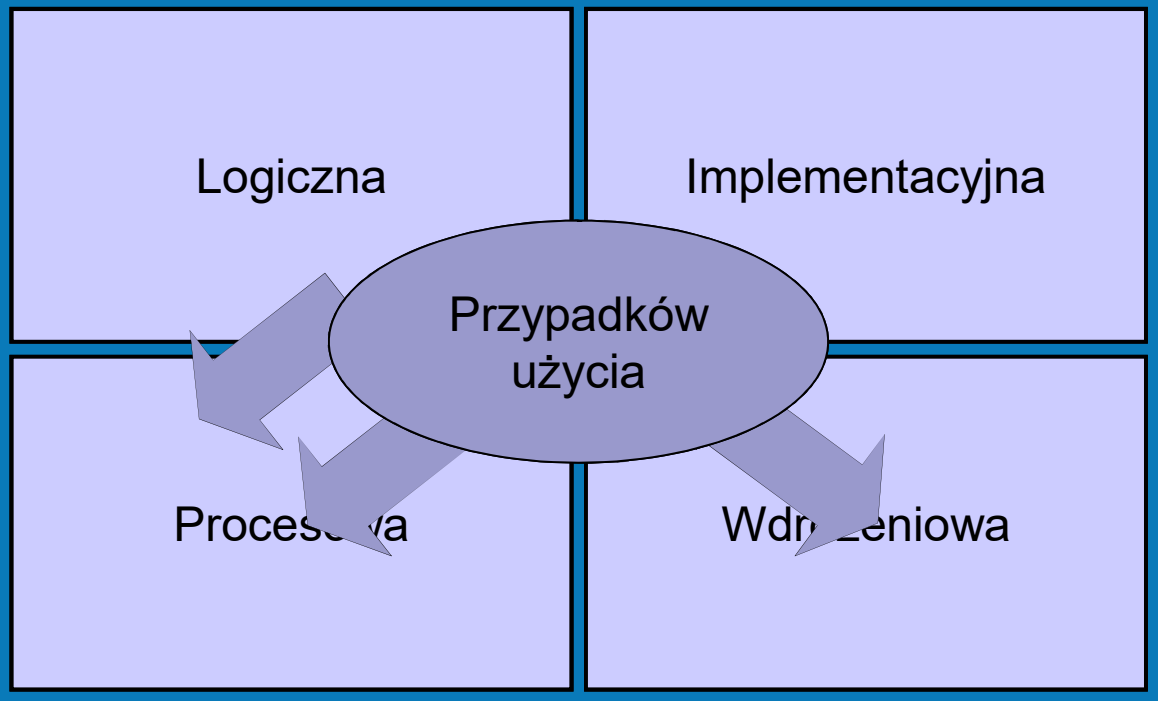
Architektura widziana z perspektywy wdrożeniowej

Opis fizycznych węzłów systemu dla typowej konfiguracji. Przypisuje procesy i wątki z modelu w perspektywie procesowej do poszczególnych węzłów. Używana w przypadku systemu rozproszonego.



Perspektywy - wnioski

Definiowanie architektury jest sterowane przypadkami użycia.



Perspektywy

- **Perspektywy** mogą reprezentować cały projekt systemu a **architektura** skupia się tylko na wybranych aspektach
- Struktura modelu – wzorce organizacji systemu
- Kluczowe elementy – krytyczne przypadki użycia, główne klasy systemu, wspólny mechanizm
- Kluczowe scenariusze pokazujące podstawowy przepływ sterowania w systemie
- Podstawowe moduły usługodawcze, opcjonalne własności.

Proces architekturo-centryczny

Artefakty w RUP odnoszące się do architektury:

1. Podstawowe:

   Opis architektury oprogramowania (*software architecture description*, SAD).

2. Perspektywy architektury odnoszące się do danego przedsięwzięcia

   Prototyp architektury – sprawdzenie i baza odniesienia dla dalszego tworzenia

Proces architekturo-centryczny

Artefakty w RUP odnoszące się do architektury:

- Pochodne:
- Wytyczne projektu właściwe dla danej architektury – podstawa do użycia wzorców
- Struktura produktu w środowisku tworzenia wyznaczona przez perspektywę implementacyjną
- Struktura zespołu wyznaczona przez strukturę perspektywy implementacyjnej

Role związane z architekturą systemu

- **Architekt oprogramowania** – odpowiedzialny za architekturę
- **Projektanci** zajmujący się znaczącymi dla architektury klasami
- **Integratorzy** scalający główne komponenty oprogramowania
- **Testerzy** testujący prototyp architektury

Przeznaczenie architektury

Opis architektury systemu jest istotny dla:

- Procesu ewolucji systemu
- Powtórnego użycia
- Uzyskania dodatkowych cech jakościowych: wydajności, dostępności, przenośności, bezpieczeństwa.
- Przypisania zadań projektowych zespołom i/lub podwykonawcom.
- Wspomagania procesu podejmowania decyzji o wykorzystaniu gotowych komponentów
- Wbudowania systemu w szerszy kontekst.

Pojęcia związane z architekturą

- Styl architektoniczny
  - Zależy od wyboru zrębu (framework), oprogramowania middleware, narzędzi, zaleceń.
- Mechanizm architektoniczny
- Klasa, grupa klas, wzorzec dający wspólne rozwiązanie wspólnego problemu
- Wzorzec architektoniczny
- Gotowa forma

Wzorce architektoniczne

- Gotowe do wykorzystania formy rozwiązujące popularne problemy architektoniczne.

- Rama/zrąb architektury (ang. architectural framework) lub infrastruktura (ang. middleware) – zbiór komponentów na których buduje się określony rodzaj architektury

Przykłady wzorców

Kategoria	*Wzorzec*
Struktura	Layers
	Pipes and filters
	Blackboard
Systemy rozproszone	Broker
Systemy interaktywne	Model-View-Controller
	Presentation-Abstraction-Control
Systemy adaptacyjne	Reflection
	Microkernel

Przykłady wzorców

Kategoria	*Wzorzec*
Struktura	Layers
	Pipes and filters
	Blackboard
Systemy rozproszone	Broker
Systemy interaktywne	Model-View-Controller
	Presentation-Abstraction-Control
Systemy adaptacyjne	Reflection
	Microkernel

Dekompozycja aplikacji na podzadania pogrupowane w poziomy abstrakcji

Przykłady wzorców

Kategoria	*Wzorzec*
Struktura	Layers
	Pipes and filters
	Blackboard
Systemy rozproszone	Broker
Systemy interaktywne	Model-View-Contro
	Presentation-Abstr
Systemy adaptacyjne	Reflection
	Microkernel

Wyjście jednego elementu przetwarzającego (wątku, procesu, ...) staje się wejściem kolejnego.
Znany także jako *pipelining*.

Przykłady wzorców

Kategoria	*Wzorzec*
Struktura	Layers
	Pipes and filters
	Blackboard
Systemy rozproszone	Broker
Systemy interaktywne	Model-View-Co
	Presentation-Abs
Systemy adaptacyjne	Reflection
	Microkernel

Grupa wyspecjalizowanych podsystemów (agentów) operujących na wspólnej strukturze danych

Przykłady wzorców

Kategoria	*Wzorzec*
Struktura	Layers
	Pipes and filters
	Blackboard
Systemy rozproszone	Broker
Systemy interaktywne	Model-View
	Presentation-A
Systemy adaptacyjne	Reflection
	Microkernel

Strukturalizacja systemu rozproszonego, składającego się z niezależnych komponentów współdziałających z wykorzystaniem komunikacji zdalnej

Przykłady wzorców

Kategoria	*Wzo*
Struktura	Layers
	Pipes and filters
	Blackboard
Systemy rozproszone	Broker
Systemy interaktywne	Model-View-Controller
	Presentation-Abstraction-Control
Systemy adaptacyjne	Reflection
	Microkernel

Metoda separacji interfejsu od danych oraz sposobu ich przetwarzania. Dzieli system na trzy dziedziny: model, logikę biznesową i widok

Przykłady wzorców

Kategoria	*Wzo*
Struktura	Layers
	Pipes and filters
	Blackboard
Systemy rozproszone	Broker
Systemy interaktywne	Model-View-Controller
	Presentation-Abstraction-Control
Systemy adaptacyjne	Reflection
	Microkernel

System przedstawiony w postaci struktury interaktywnych systemów (agentów). Każdy z agentów jest odpowiedzialny za część funkcjonalności i składa się z trzech warstw: prezentacji, abstrakcji i kontroli

Przykłady wzorców

Kategoria	*Wzo*
Struktura	Layers
	Pipes and filters
	Blackboard
Systemy rozproszone	Broker
Systemy interaktywne	Model-View-Controller
	Presentation-Abstraction-Control
Systemy adaptacyjne	Reflection
	Microkernel

Wzorzec umożliwiający dynamiczną zmianę struktury i zachowania systemu. Modyfikacja może dotyczyć tak fundamentalnych elementów jak system typów czy mechanizm wywoływania funkcji

Przykłady wzorców

Kategoria	*W*
Struktura	Layers
	Pipes and filters
	Blackboard
Systemy rozproszone	Broker
Systemy interaktywne	Model-View-C
	Presentation
Systemy adaptacyjne	Reflection
	Microkernel

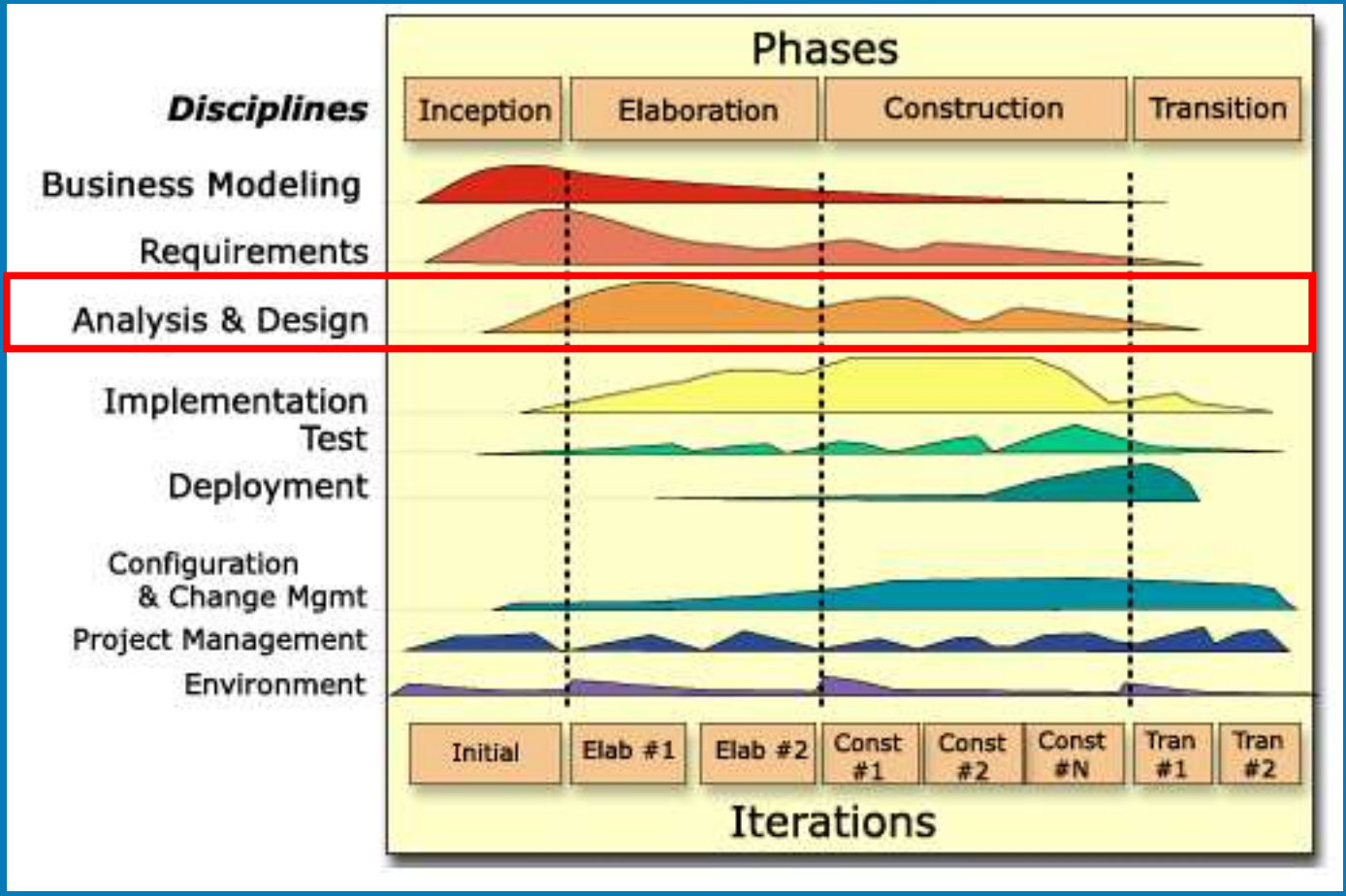
Wzorzec dla systemów, które muszą być zdolne do adaptacji w zmieniających się wymaganiach systemowych. Separuje minimalną podstawową funkcjonalność od rozszerzeń specyficznych dla określonego systemu. Mikrojądro służy również jako gniazdo dla rozszerzeń wraz mechanizmem umożliwiającym ich koordynację.

Czynności wykonywane w fazie analizy i projektowania

Analiza i projektowanie

- Czynności związane z analizą i projektowaniem mają na celu zdefiniowanie architektury systemu, stanowiącej rozwiązanie problemu, który był podstawą do rozpoczęcia prac nad systemem.
- Podstawowymi artefaktami procesu analizy i projektowania są:
  - Model analizy (opcjonalny) – określający założenia projektowe
  - Model projektu – określający słownictwo problemu i jego rozwiązanie

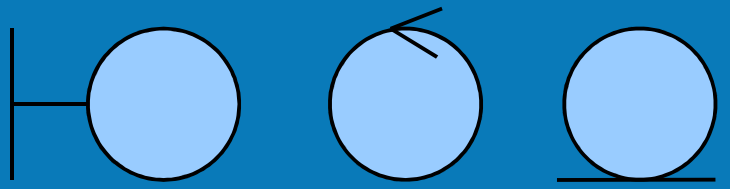
Analiza i projektowanie



Aktywność: Wstępna definicja architektury

- Opracowanie szkicu architektury systemu
- Wstępny zbiór elementów istotnych z punktu widzenia architektury
- Wstępna organizacja systemu (podział na warstwy)
- Wybór przypadków użycia do realizowania w bieżącej iteracji
- Identyfikacja klas analizy

Klasy analizy



Reprezentują wstępny koncepcyjny model elementów w systemie, które mają określoną odpowiedzialność i zachowanie.

W UML reprezentowane przez stereotypy:

- <<boundary>>
- <<entity>>
- <<control>>

są sposobem wyrazu pewnych założeń architektonicznych

Aktywność: Analiza zachowania

- Dla przypadku użycia
- Identyfikacja klas analizy zaspokajających wymagania dotyczące zachowania.
- Wynik: realizacja przypadku użycia wyrażona w terminach współpracy klas analizy.

Aktywność: Projektowanie komponentów

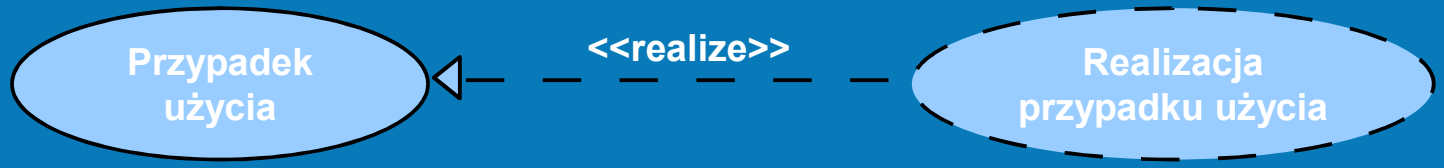
- Identyfikacja elementów projektowych (klasy projektowe, interfejsy, podsystemy)
- Jak elementy projektowe realizują zachowanie
- Aktualizacja realizacji przypadków użycia w oparciu o elementy projektowe

Aktywność: Projektowanie bazy danych

- Identyfikacja klas trwałych
- Projekt odpowiedniej struktury bazy danych dla przechowywania trwałych klas
- Określenie mechanizmów i strategii przechowywania i pobierania trwałych danych w sposób zapewniający odpowiednią wydajność.

Realizacja przypadku użycia

- Opis sposobu, w jaki konkretny przypadek użycia jest realizowany w modelu projektowym. Powiązanie pomiędzy modelem przypadków użycia a modelem projektowym.
- W fazie analizy celem przypadków użycia jest identyfikacja klas analizy realizujących przepływ sterowania w przypadku użycia.



Realizacja przypadku użycia

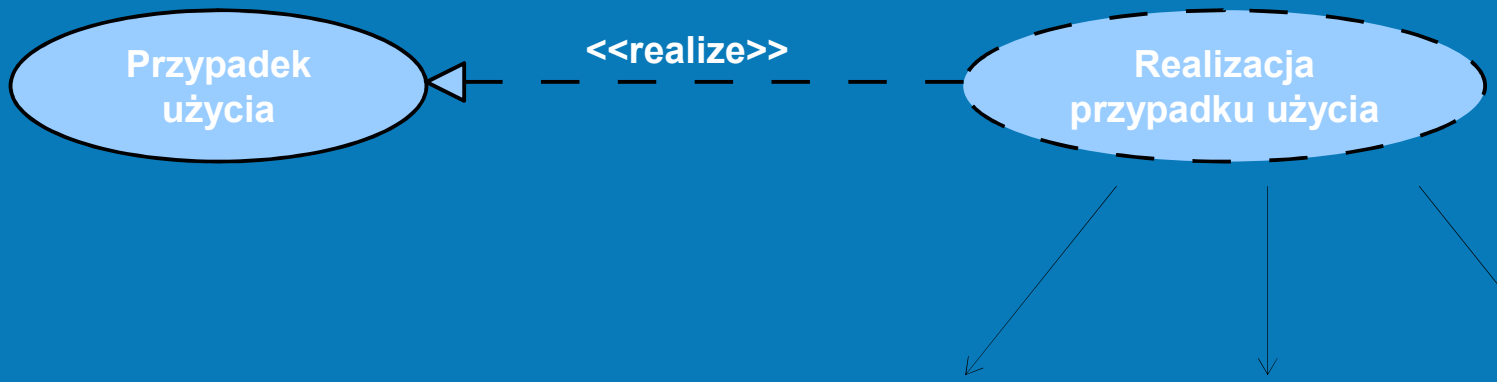
W UML-u realizacja przypadku użycia może być reprezentowana przez zbiór diagramów, które modelują współpracę obiektów:

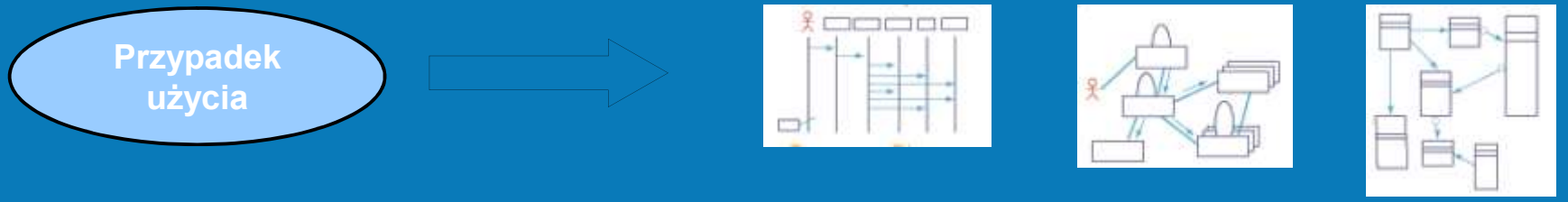
- diagram klas – opis klas i związków pomiędzy nimi,
- diagramy interakcji (współpracy i sekwencji) – modelują sposób interakcji obiektów w celu realizacji przypadku użycia.

Elementy realizacji przypadku użycia

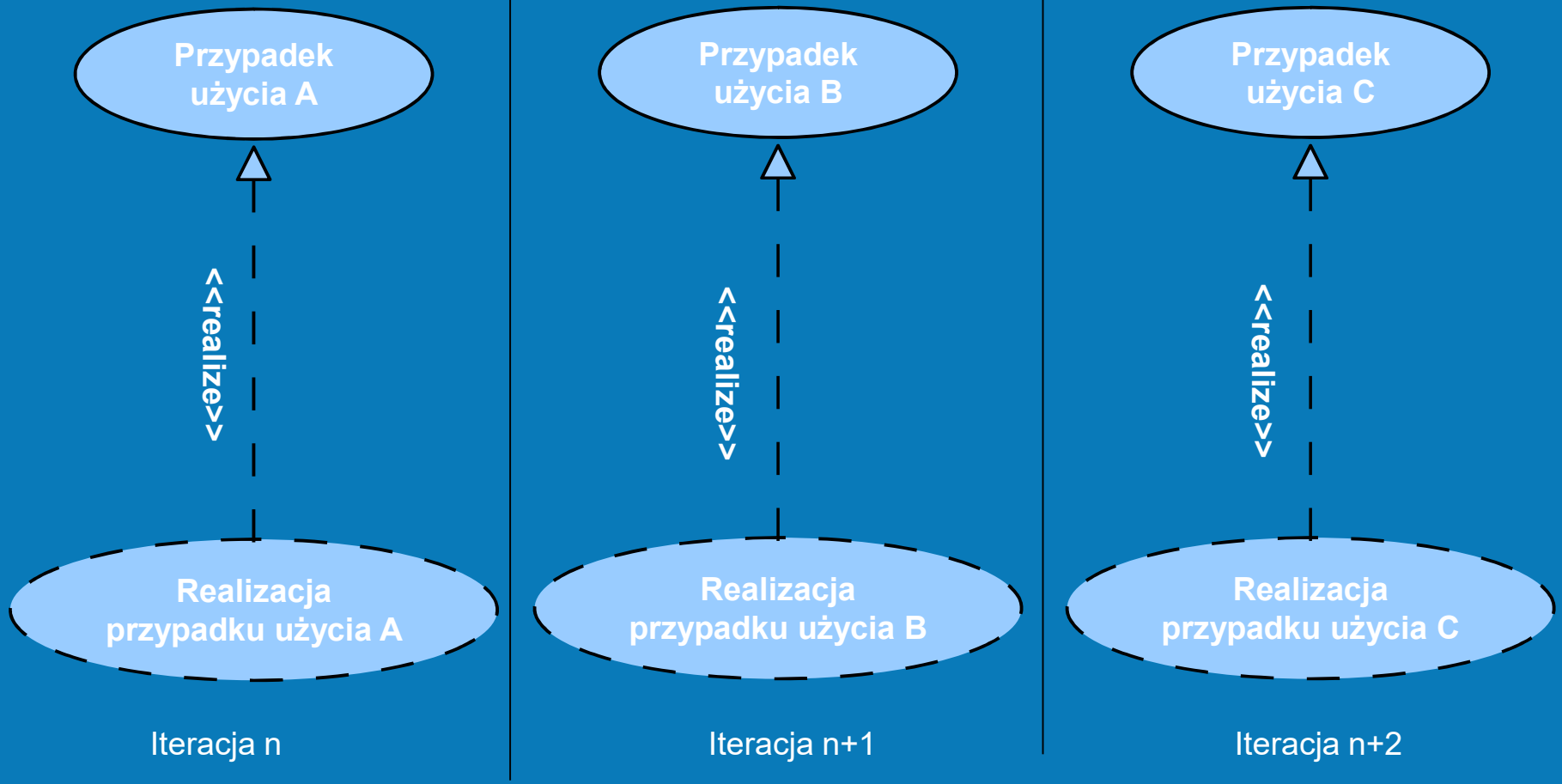
Model przypadków użycia

Model projektowy





Proces iteracyjny*



* Oczywiście jest to możliwe tylko w przypadku, gdy architektura systemu jest elastyczna i odpowiednio wcześniej zdefiniowana.