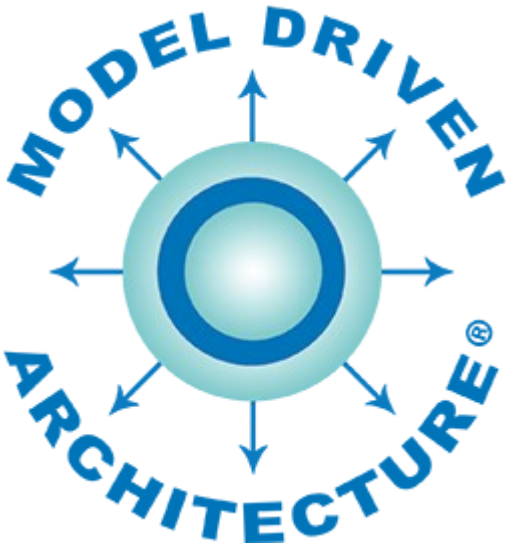


Architektura Sterowana Modelem

Model Driven Architecture

prezentacja 5

Język UML – modelowanie systemu informatycznego

wersja 1.0

dr inż. Paweł Głuchowski

Wydział Informatyki i Telekomunikacji, Politechnika Wrocławska

Treść prezentacji

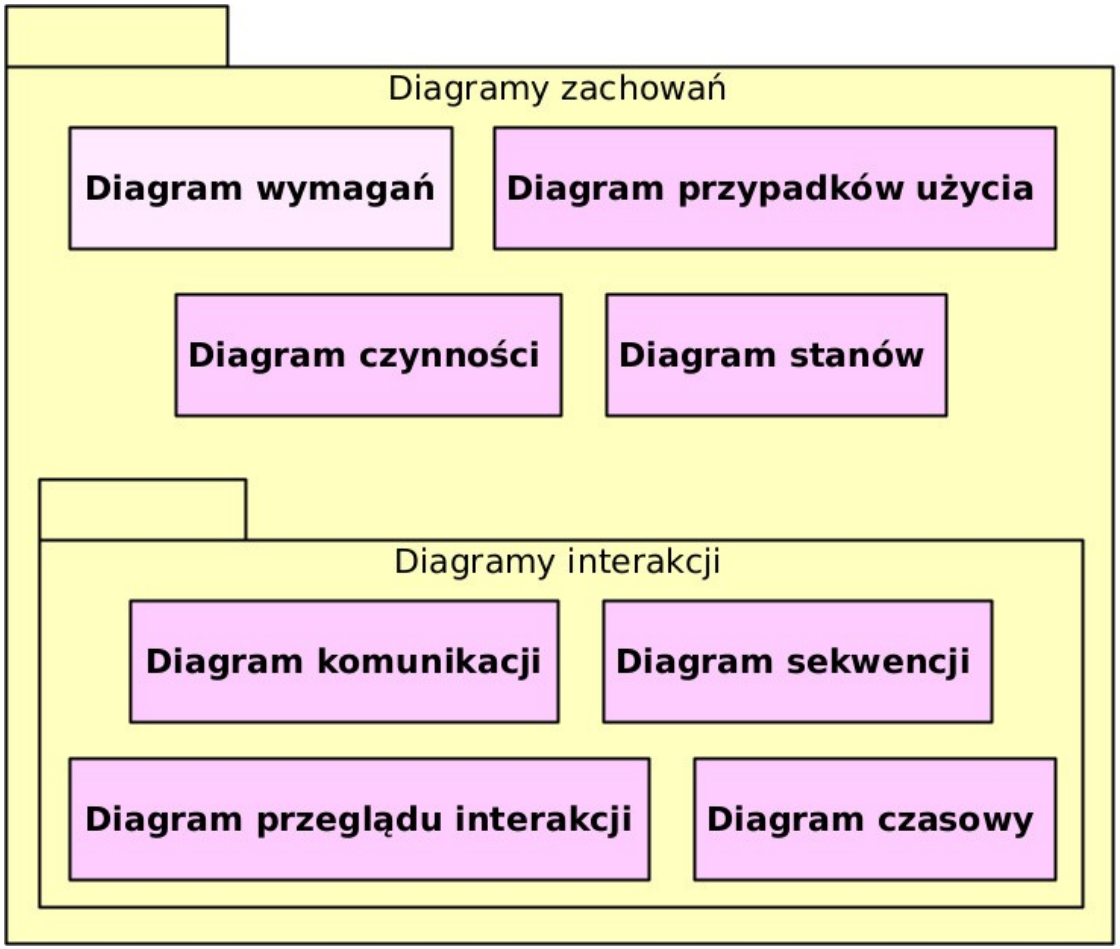
1. Systematyka diagramów UML
2. Relacje między diagramami
3. Obiektowe modelowanie struktury programu
4. Związek kodu z diagramem

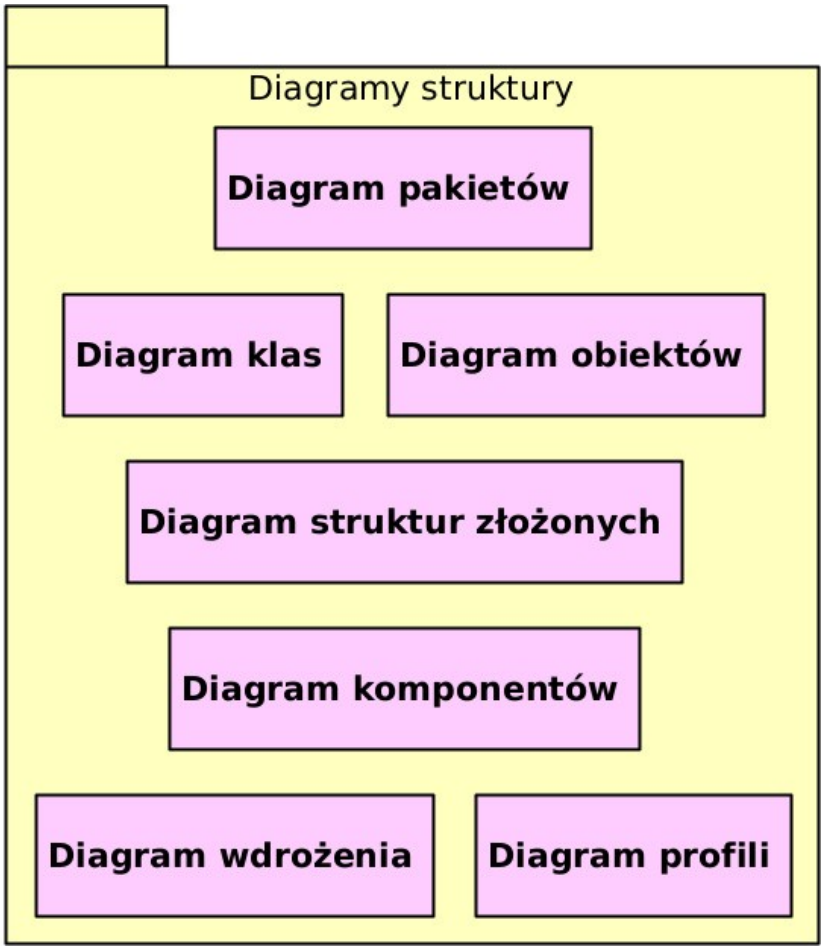
1

Systematyka diagramów UML

Podział diagramów na podstawie celu ich stosowania

- Diagramy UML (i wybrane diagramy SysML) dzielą się na:
  - **diagramy zachowań** /behavioral/ – modelują zachowanie,
  - **diagramy struktury** /structural/ – modelują strukturę.





Diagramy zachowań w procesie wytwarzania oprogramowania SI

- **Diagram wymagań**
  - cele do osiągnięcia przez SI i ograniczenia jego działania.
- **Diagram przypadków użycia**
  - procesy biznesowe SI spełniające wymagania funkcjonalne.
- **Diagram czynności**
  - konceptualny lub implementacyjny algorytm realizacji: przypadku użycia, operacji klasy, przepływu sterowania i danych między częściami SI itp.
- **Diagram stanów**
  - zmienność stanu SI i jego części (komponentów i obiektów).

Diagramy zachowań w procesie wytwarzania oprogramowania SI

- **Diagram komunikacji**
  - interakcja między częściami SI (komponentami, klasami, obiektami) i z jego otoczeniem (aktorami).
- **Diagram sekwencji**
  - interakcja między częściami SI i z jego otoczeniem, z wykorzystaniem osi czasu.
- **Diagram przeglądu interakcji**
  - algorytm realizacji różnych interakcji (komunikacji i sekwencji).
- **Diagram czasowy**
  - diagram czasu zmiany stanów systemu i jego części podczas interakcji między częściami SI i z jego otoczeniem.

Diagramy struktury w procesie wytwarzania oprogramowania SI

- **Diagram pakietów**
  - logiczne uporządkowanie elementów modelu SI (np. klas).
- **Diagram klas**
  - statyczna obiektowa struktura SI (klasy i relacje między nimi).
- **Diagram obiektów**
  - instancje klas i relacje między nimi.
- **Diagram struktur złożonych**
  - wewnętrzne struktury klas i współdziałanie klas w realizacji przypadków użycia i operacji.

Diagramy struktury w procesie wytwarzania oprogramowania SI

- **Diagram komponentów**
  - struktura SI w postaci układu niezależnych części (komponentów).
- **Diagram wdrożenia**
  - fizyczna i logiczna struktura SI i powiązanie oprogramowania ze sprzętem.
- **Diagram profili**
  - dostosowanie metamodelu UML do zastosowania w modelu SI.

W „nieswoim” diagramie

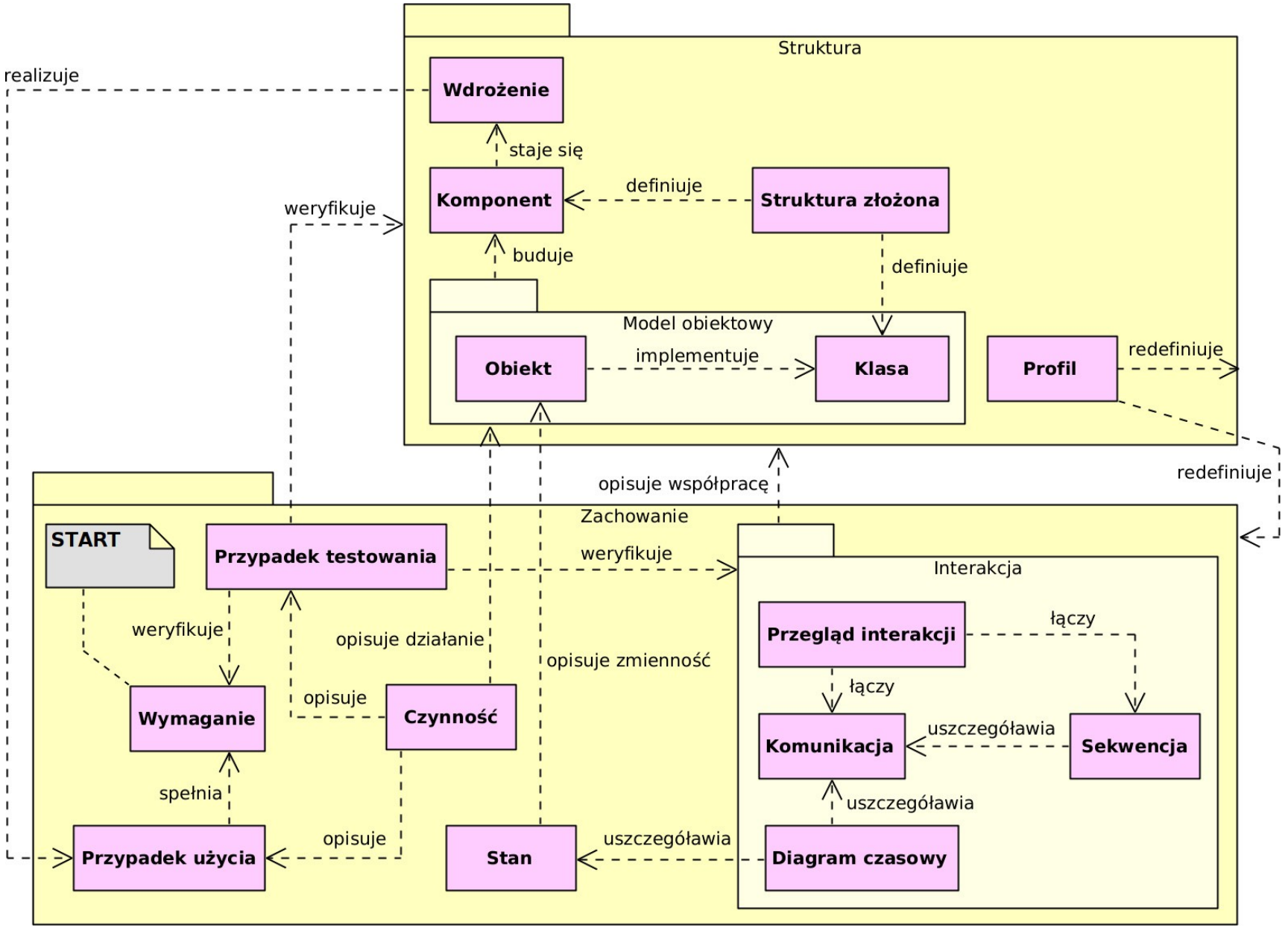
- Niektóre rzeczy pojawiają się na „nieswoich” diagramach, np.
  - **przypadki użycia** – na diagramie wymagań,
  - **pakiety** – na diagramie klas (i prawie każdym innym),
  - **klasy** – na diagramie komponentów,
  - **współdziałania** – na diagramie przypadków użycia,
  - **komponenty** – na diagramie wdrożenia.

2

Relacje między diagramami

Relacje między diagramami

Relacje między elementami różnych typów diagramów



Typowe przejścia między diagramami

- Na każdy diagram wpływa słowna specyfikacja systemu!
- **Wymagania** → **Przypadki użycia**:
  - proces biznesowy spełnienia wymagania.
- **Przypadki użycia** → **Czynności**:
  - scenariusz realizacji przypadku użycia i przypadku testowania.
- **Przypadki użycia** → ***Interakcje***:
  - przepływ sterowania i danych w realizacji przypadku użycia.
- **Wymagania**, **Przypadki użycia** i **Czynności** → ***Struktury***:
  - podział systemu na części i warstwy oprogramowania (→ Komponenty),
  - zastosowanie wzorców projektowych (→ Klasy),
  - wdrożenie systemu (→ Wdrożenie).
- **Przypadki użycia** → **Struktury złożone**:
  - współdziałanie klas w realizacji przypadku użycia (→ Współdziałanie).

Typowe przejścia między diagramami

- **Komponenty** → **Klasy** i **Obiekty**:
  - relacje między klasami,
  - zastosowanie wzorców projektowych.
- **Klasy** → **Obiekty** i **Struktury złożone**:
  - rola części i instancji klasy i relacja między nimi,
  - współdziałanie klas i ich instancji.
- **Komponenty**, **Klasy** i **Obiekty** → **Czynności**:
  - algorytm wykonania operacji klasy,
  - przepływ sterowania i danych w realizacji operacji klasy.
- **Komponenty**, **Klasy** i **Obiekty** → ***Interakcje***:
  - przepływ sterowania i danych w realizacji operacji klasy,
  - współpraca i komunikacja klas, obiektów i komponentów.

3

Obiektowe modelowanie struktury programu

Podstawowe założenia

- **Klasy i obiekty** (instancje klas):
  - reprezentują elementy rzeczywistego świata;
  - są definiowane na podstawie tego, za co są odpowiedzialne;
  - wykonują operacje, komunikują się, mają atrybuty i zmieniają swój stan.
- **Abstrakcja** – ogólna definicja klasy i operacji:
  - określa tylko funkcjonalność,
  - pomija szczegóły implementacji,
  - wydziela najważniejsze cechy obiektu lub rodziny obiektów,
  - przy pomocy abstrakcyjnych klas i operacji oraz interfejsów.
- **Dziedziczenie** – definiowanie klasy na bazie innej, ogólniejszej klasy:
  - kopiuje wybrane atrybuty i operacje klasy bazowej,
  - redefiniuje wybrane atrybuty i operacje klasy bazowej,
  - dodaje własne atrybuty i operacje.

Podstawowe założenia

- **Polimorfizm** – wielopostaciowość klasy bazowej (klasy dziedziczonej lub zapewnianego interfejsu):
  - różne zachowanie obiektów różnych klas klasy bazowej (gdy są używane jako obiekty klasy bazowej),
  - przez zapewnienie tego samego interfejsu lub dziedziczenie tej samej klasy.
- **Hermetyzacja** – ograniczenie dostępu do klasy i obiektu:
  - ukrywa szczegóły implementacji klasy;
  - określa dostępność atrybutów i operacji klasy z zewnątrz;
  - ogranicza możliwość zmiany stanu obiektu przez inny obiekt (tylko on sam ma pełny dostęp do swoich atrybutów i operacji);
  - przy pomocy widoczności atrybutów i operacji.

Identyfikacja klas, obiektów, atrybutów i operacji

- **Analiza słownej specyfikacji** wymagań i przypadków użycia:
  - rzeczownik może określać klasę, jej instancję lub atrybut;
  - czasownik może określać operację klasy.
- **Nazwa** klasy, atrybutu i operacji – jednoznacznie zrozumiała i określająca przeznaczenie.
- **Dostępność** do klasy, atrybutu i operacji (widoczność).
- **Spójność klasy** – wykonuje tylko swoje zadania.
- **Rozdzielenie operacji** na tworzące obiekt i używające obiekt.
- **Normalizacja operacji** klasy – różna funkcjonalność różnych operacji klasy.
- **Stereotyp klasy** – zastosowanie odpowiedniego typu klasy.
- **Kompozycja i dekompozycja klas**:
  - podzielenie klasy, która ma za dużo zobowiązań, na mniejsze klasy;
  - połączenie klas, które mają za mało zobowiązań, w większą klasę.

Identyfikacja relacji między klasami i obiektami

- **Zależność**:
  - gdy operacja klasy ma dostęp do innej klasy,
    ale NIE przez swój atrybut (związek niestrukturalny).
- **Uogólnienie**:
  - gdy różne klasy mają wspólne atrybuty, operacje lub relacje;
  - ich wspólna część trafia do ich klasy bazowej (dziedziczonej);
  - łańcuch dziedziczenia NIE może się zapętlać.
- **Związek strukturalny**:
  - **asocjacja** – gdy operacja klasy ma dostęp do innej klasy
    przez swój atrybut,
  - **agregacja** / **kompozycja** – gdy dodatkowo klasa zawiera instancję
    tej drugiej klasy (nie na wyłączność / na wyłączność),
  - określa ilościowy stosunek między instancjami powiązanych klas,
  - określa jaką rolę pełnią instancje powiązanych klas względem siebie.

Identyfikacja wzorców projektowych
(temat następnej prezentacji)

- **Wzorzec projektowy** – koncepcyjny model rozwiązania powtarzającego się problemu projektowego:
  - **złożony** – wiąże ze sobą warstwy oprogramowania,
  - **prosty** – wiąże ze sobą klasy (zwykle w ramach jednej warstwy oprogramowania).

Szczegółowość modelu

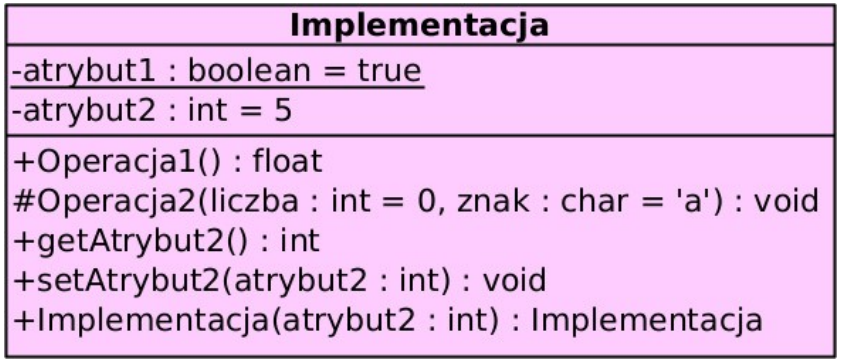
- **Zbyt ogólny model**
  - zbyt dowolna i nieudokumentowana implementacja modelu,
  - brak wykrycia bloków ponownego użycia.
- **Zbyt szczegółowy model**
  - zbyt sztywna i skomplikowana implementacja modelu,
  - nieistotne szczegóły utrudniają analizę modelu.

Poziomy szczegółowości klasy

- **Koncepcja** – definicja zadań klasy:
  - ogólna koncepcja klasy w modelu domeny,
  - nazwanie klasy, głównych atrybutów i głównych operacji.

- **Specyfikacja** – definicja zachowania klas:
  - wstępna, ogólna definicja atrybutów, operacji, stereotypów i relacji klasy.

- **Implementacja** – definicja „kodu” klasy:
  - końcowa, pełna definicja atrybutów, operacji, stereotypów i relacji klasy.



4

Związek kodu z diagramem

Związek kodu z diagramem

TODO

- Przykład: diagram klas → kod klas
- Przykład: diagram sekwencji → kod operacji klasy
- Przykład: diagram czynności → kod operacji klasy.

Temat następnej prezentacji

Wzorce projektowe w budowie modelu oprogramowania