Inżynieria Oprogramowania

Modelowanie Systemu Informatycznego

prezentacja 2

Modelowanie w projekcie programistycznym i język UML

wersja 1.0

dr inż. Paweł Głuchowski

Wydział Informatyki i Telekomunikacji, Politechnika Wrocławska

Treść prezentacji

1

Projekt informatyczny

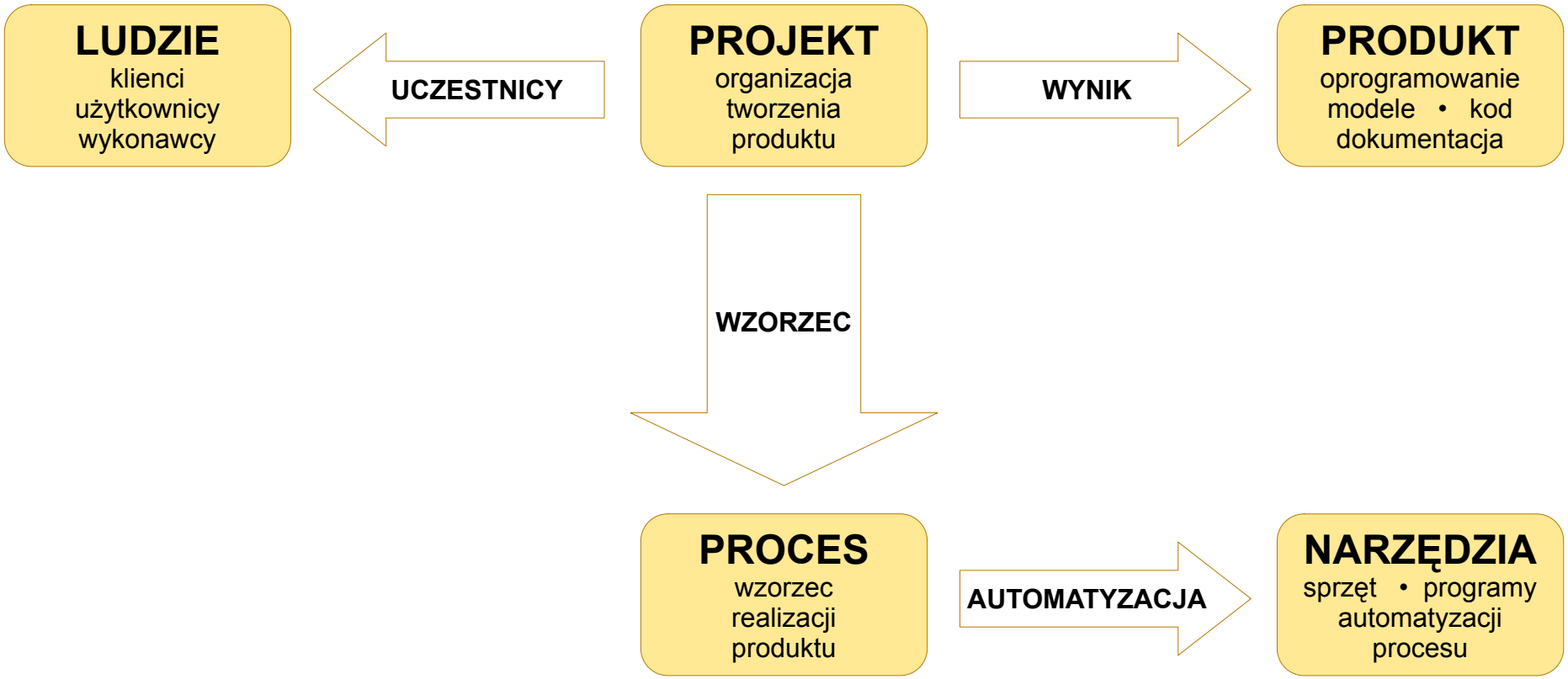
Projekt informatyczny

na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- To **organizacja tworzenia produktu** (np. oprogramowania):
  - planowanie, organizowanie, monitorowanie i kontrolowanie.
- **Elementy**: ludzie, produkt, proces.
- **Pojęcia:**
  - wykonalność projektu,
  - zarządzanie ryzykiem,
  - struktura grup projektowych,
  - szeregowanie zadań projektowych,
  - zrozumiałość projektu,
  - sensowność działań w projekcie.

Projekt informatyczny

na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr



2

Proces tworzenia oprogramowania

Proces tworzenia oprogramowania

na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- **Kompletny zbiór czynności odwzorowania:**

  wymagania użytkownika
  →
  oprogramowanie.

- Czynniki procesu:
  - organizacyjne,
  - dziedzinowe,
  - cykl życia,
  - techniczne.

Proces tworzenia oprogramowania

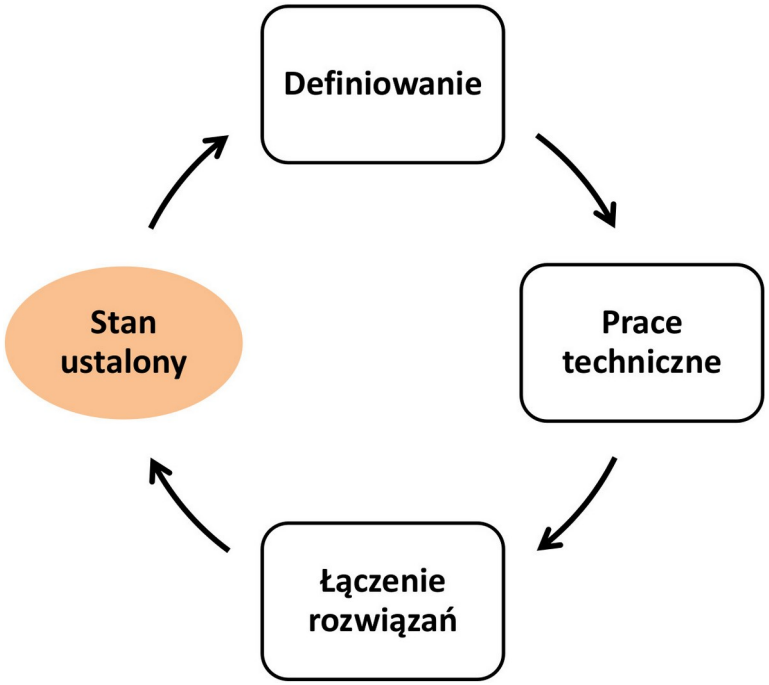
na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- **Czynności:**
  - modelowanie struktury i zachowania użytkownika;
  - wymagania użytkownika i przypadki użycia oprogramowania;
  - analiza i projektowanie – różne perspektywy;
  - implementacja – tworzenie oprogramowania, testowanie modułów i scalanie oprogramowania;
  - testowanie – opisanie danych testowych, procedur i metryk poprawności;
  - wdrożenie – ustalenie konfiguracji gotowego oprogramowania;
  - zarządzenie zmianami i dbanie o spójność elementów oprogramowania;
  - zarządzanie projektem – opisanie różnych strategii prowadzenia procesu tworzenia oprogramowania;
  - określenie środowiska – opisanie struktury niezbędnej do opracowania oprogramowania.

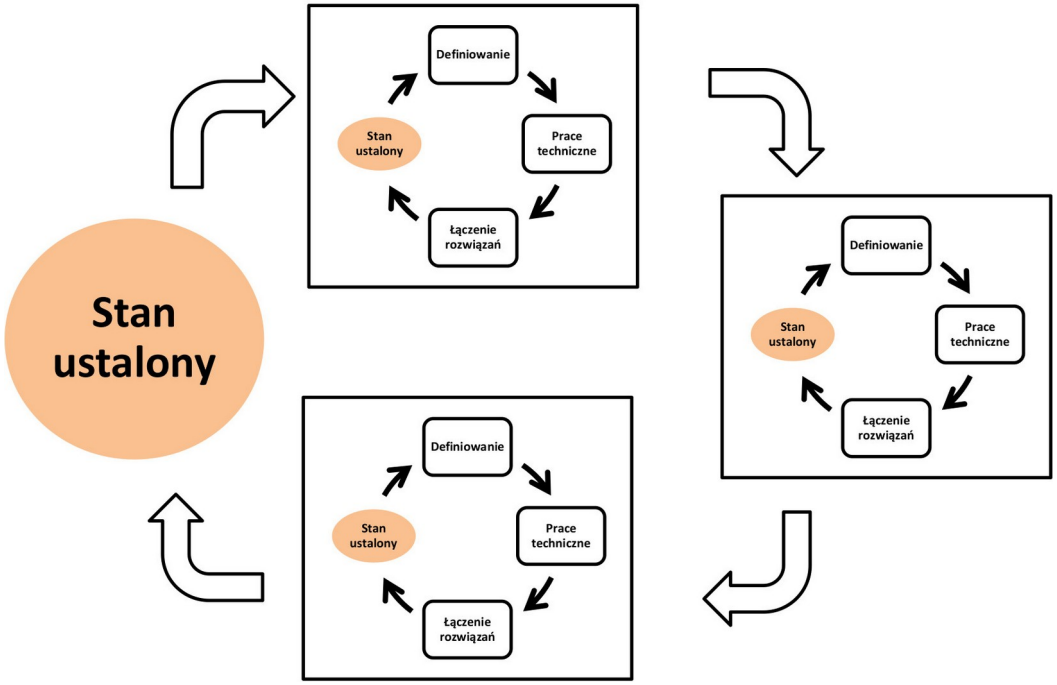
Modele cyklu życia oprogramowania

źródło: Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- Ogólny model cyklu życia oprogramowania:



Iteracyjny model cyklu życia oprogramowania:



Modele cyklu życia oprogramowania

- **Model kaskadowy.**
- **Model kaskadowy typu V.**
- Modele ewolucyjne:
  - **model przyrostowy,**
  - **model spiralny,**
  - model spiralny Win-Win,
  - model równoległy.
- Model szybkiej rozbudowy aplikacji.
- Model oparty na prototypowaniu.
- Model oparty na programowaniu odkrywczym.
- Model oparty na metodach formalnych.
- Model oparty na komponentach.
- Techniki czwartej generacji.

Model kaskadowy

- Liniowa, sekwencyjna kolejność realizacji etapów:
  - określenie wymagań → projektowanie → implementacja → testowanie → wdrożenie → konserwacja
- **Zalety modelu:**
  - proste zależności między jego etapami,
  - łatwo nim zarządzać.
- **Wady modelu:**
  - nie można cofnąć się do etapu wymagającego zmian,
  - trudno usunąć błędy z początkowych etapów,
  - powstałe oprogramowanie może nie spełnić wymagań.

Model kaskadowy typu V

- Liniowa, sekwencyjna kolejność realizacji etapów,
  gdzie rozwojowi oprogramowania towarzyszy testowanie:
  - określenie wymagań (i testów akceptacyjnych)
    → projektowanie systemu (i jego testów integracyjnych)
    → projektowanie komponentów systemu (i ich testów integracyjnych)
    → implementacja → testy jednostkowe → testy integracyjne komponentów → testy integracyjne systemu → testy akceptacyjne
    → wdrożenie → konserwacja
- **Zalety modelu:**
  - weryfikacja i walidacja planowana jest na każdym jego etapie,
  - błędy z początkowych etapów są mniejsze.
- **Wady modelu:**
  - nie można cofnąć się do etapu wymagającego zmian,
  - trudno usunąć błędy z początkowych etapów.

Model przyrostowy

- Ewolucyjny model kaskadowo-iteracyjny.
- Każda iteracja dotyczy innego komponentu składowego SI:
  - specyfikacja wymagań → ogólny projekt → kolejne iteracje: (wybór komponentu → szczegółowy projekt komponentu → implementacja komponentu → testowanie → wdrożenie) → konserwacja całego SI
- **Zalety modelu:**
  - stopniowe testowanie, walidacja i wdrażanie (komponent po komponencie) – a nie dopiero na końcu procesu.
- **Wady modelu:**
  - dodatkowy koszt niezależnej realizacji komponentów SI.

Model spiralny

- Ewolucyjny model z cykliczną kolejnością realizacji etapów:
  - specyfikacja wymagań → projektowanie → implementacja → testowanie → wdrożenie
- **Zalety modelu:**
  - w kolejnym cyklu można wrócić do dowolnego etapu i „naprawić” co trzeba,
  - kolejne etapy projektu uwzględniają rozwiązania alternatywne i ryzyko realizacji.

Etapy tworzenia oprogramowania

1. **Modelowanie struktury i dynamiki systemu**
   – perspektywa koncepcji.

- Określa, **co** trzeba wykonać jako SI.
- Zawiera:
  - model odbiorcy,
  - analizę wymagań,
  - koncepcję testów.

Etapy tworzenia oprogramowania

2. **Implementacja struktury i dynamiki systemu i wytworzenie kodu**
   – perspektywa specyfikacji.

- Określa, **jak** trzeba będzie **używać** SI.
- Zawiera:
  - model projektu – struktura i zachowanie SI,
    w tym: architektura sprzętu i oprogramowania, dostęp użytkownika, przechowywanie danych itp.;
  - testy funkcjonalne i akceptacyjne projektu.

Etapy tworzenia oprogramowania

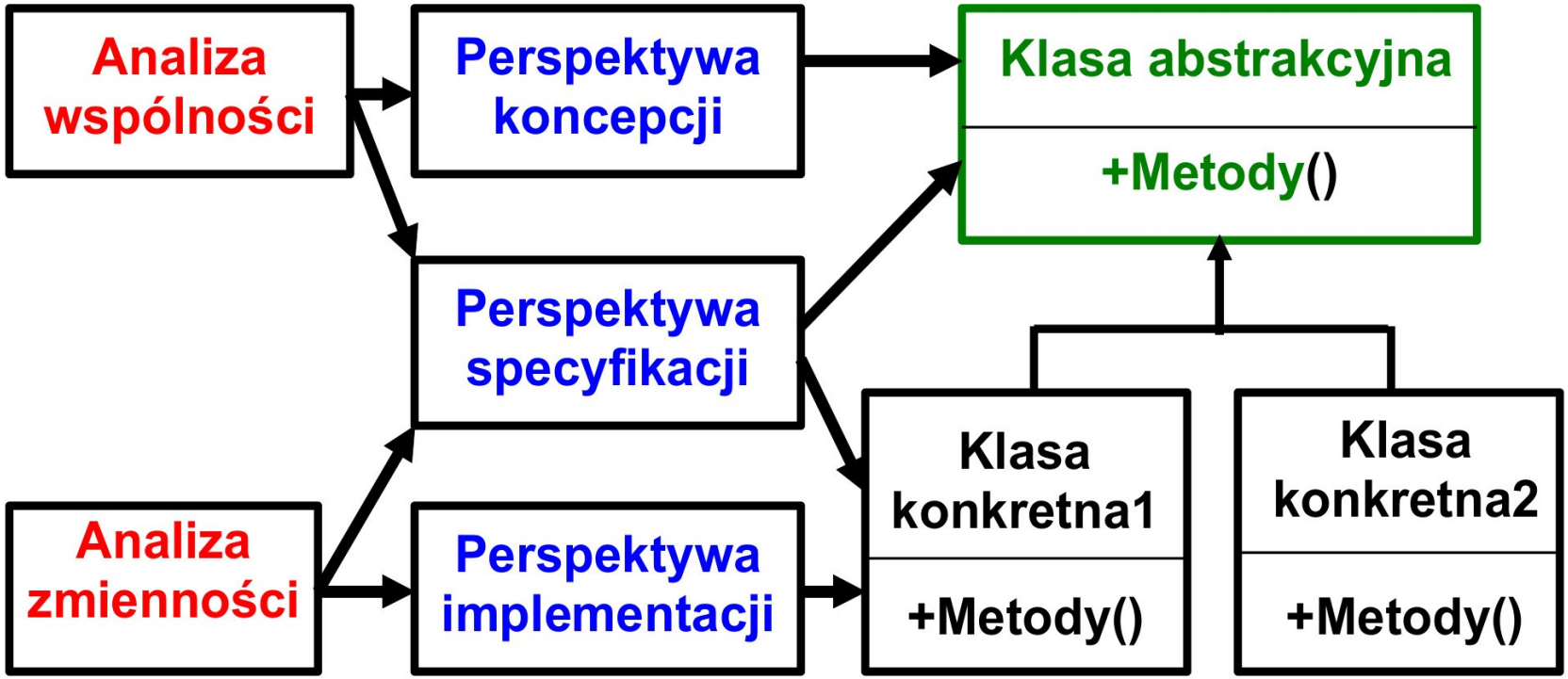
3. **Implementacja struktury i dynamiki systemu i wytworzenie kodu** – perspektywa implementacji.

- Określa, **jak** trzeba będzie **wykonać** SI.
- Zawiera:
  - oprogramowanie SI – uzupełnienie modelu SI o dodatkowe deklaracje, definicje, struktury danych, bazy danych itp.;
  - testy oprogramowania (w tym: jednostkowe, integracyjne);
  - wdrożenie SI;
  - testy systemu (w tym: akceptacyjne, wdrożeniowe).

Związki między perspektywami

źródło: Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- Przykład dla definicji klas:



Inne perspektywy

na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- **Perspektywa tworzenia i zarządzania obiektami** (model implementacji):
  - obiekt A w roli fabryki obiektów tworzy obiekt B i / lub zarządza nim.
- **Perspektywa używania obiektów** (model implementacji):
  - obiekt A tylko używa obiektu B, nie może go tworzyć;
  - wykorzystanie hermetyzacji i polimorfizmu obiektu B.

3

Ludzie w tworzeniu oprogramowania

Uczestnicy

- **Zespół projektowy:**
  - kierownik projektu,
  - analitycy,
  - projektanci,
  - testerzy.
- **Inne osoby:**
  - klienci,
  - użytkownicy,
  - eksperci.

4

Produkty w tworzeniu oprogramowania

Produkt

- **Wytworzone oprogramowanie.**
- Ale także:
  - wymagania,
  - testy,
  - produkcja,
  - wdrożenie,
  - kod źródłowy i wynikowy,
  - modele (w tym diagramy),
  - dokumentacja (w tym diagramy).

- **Proces tworzenia oprogramowania musi być dopasowany do żądanego produktu:**
  - konkretny produkt,
  - dla konkretnego użytkownika (organizacji),
  - realizujący konkretne zadania.

Model jako produkt

- **Model** – odwzorowanie oprogramowania przy pomocy:
  - klasyfikatorów,
  - zdarzeń,
  - zachowań.
- Abstrakcja jego wybranych cech.

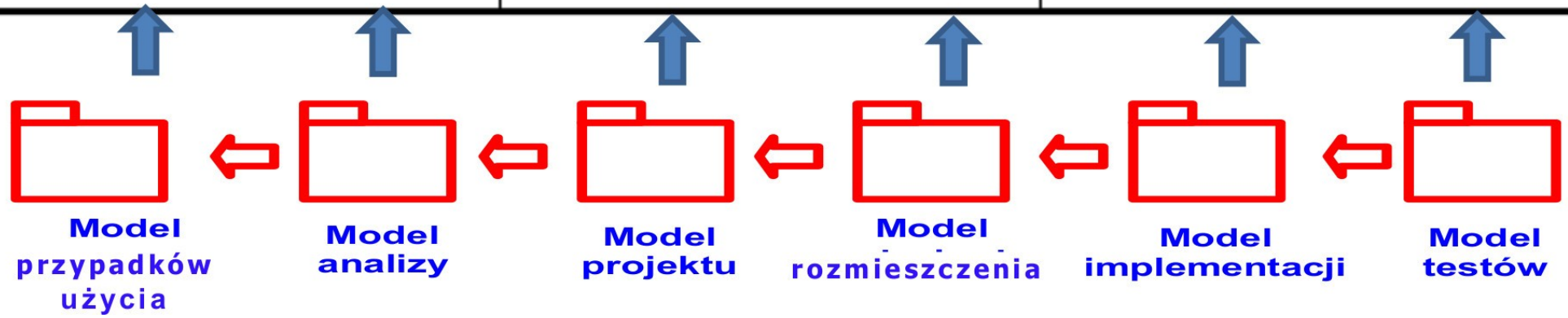
- Model zawsze jest modelem czegoś!

- Modelowana rzecz to system w pewnej domenie pojęciowej.
- **Model istniejącego oprogramowania** – analiza jego cech i zachowania.
- **Model planowanego oprogramowania** – specyfikacja jego konstrukcji i zachowania.

Związek modelu z perspektywą

źródło: Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

Modelowanie struktury i dynamiki systemu	Implementacja struktury i dynamiki systemu, 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: diagram przypadków użycia, diagram klas, diagramy sekwencji,) • testy modelu	• projektowanie (model projektowy: architektura sprzętu i oprogramowania; dostęp użytkownika; przechowywanie danych) • testy projektu	• programowanie, wdrażanie (specyfikacja programu : deklaracje, definicje; dodatkowe struktury danych: struktury „pojemnikowe”, pliki, bazy danych) • testy oprogramowania • wdrażanie • testy wdrażania



Oprogramowanie jako produkt

na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- Są to:
  - **rozkazy** – ich wykonanie pozwala spełnić określone funkcje oprogramowania w oczekiwany sposób;
  - **struktury danych** – umożliwiają przetwarzanie informacji przez oprogramowanie;
  - **dokumenty** – opisują działanie i sposób użytkowania oprogramowania.

- Oprogramowanie jest wytwarzane ale nie jest konstruowane fizycznie.
- Oprogramowanie z czasem niszczeje.
- Oprogramowanie może korzystać z gotowych komponentów, a komponenty mogą podlegać wymianie na nowe.
- Wadliwy proces tworzenia oprogramowania może spowodować powstanie niskiej jakości produktu.

Architektura oprogramowania

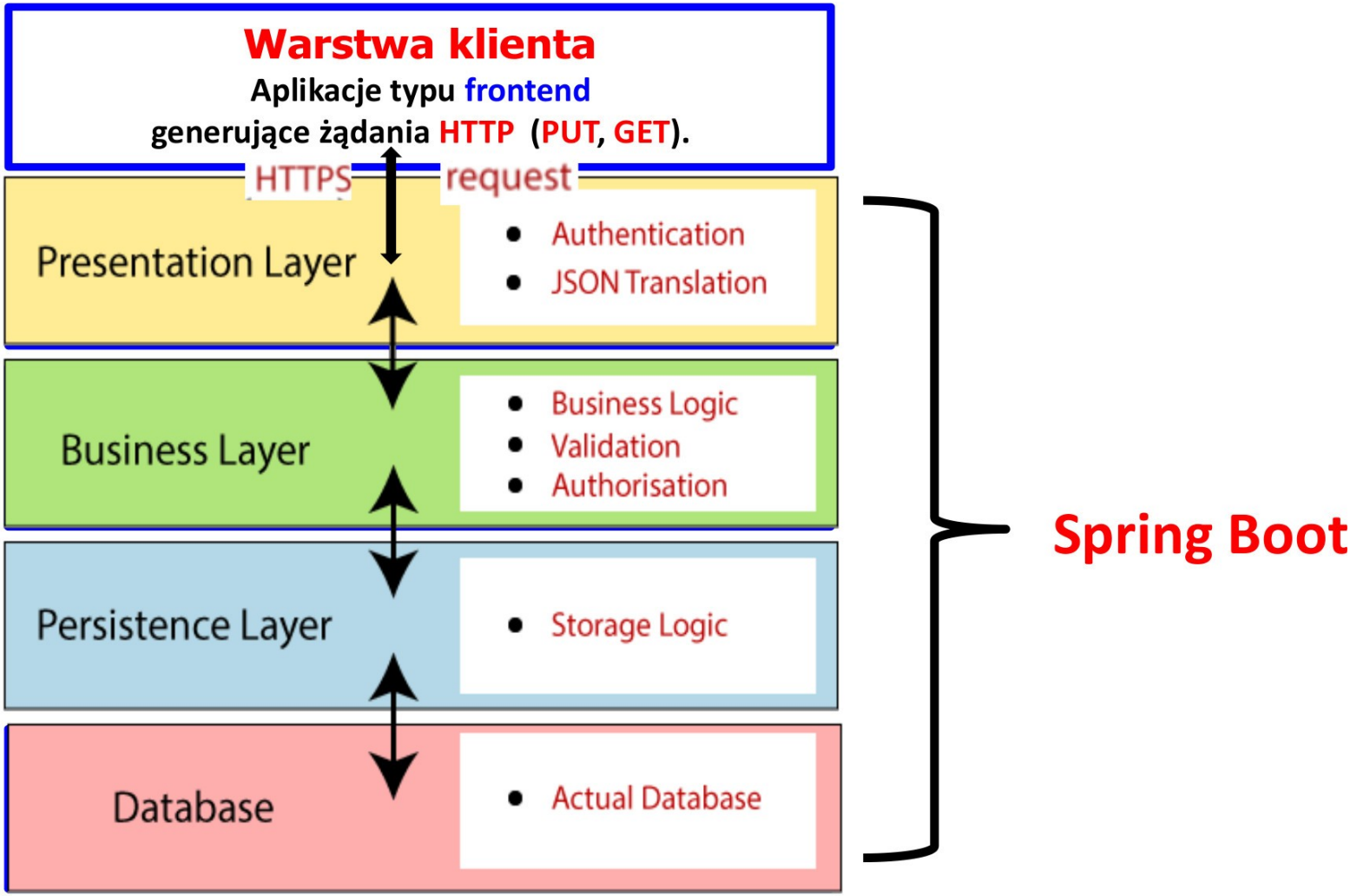
źródło: Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr
na podst. Core J2EE. Wzorce projektowe, D. Alur et al.

- Pięciowarstwowy model logicznego rozdzielenia zadań oprogramowania:

Architektura oprogramowania

źródło: Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr
na podst. Spring Boot Architecture – javatpoint

- Pięciowarstwowy model logicznego rozdzielenia zadań oprogramowania:



Mity kierownictwa projektu

na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- Standardy i procedury zapewniają tworzenie dobrego oprogramowania.
- Dobry program narzędziowy, działający na dobrym sprzęcie, gwarantuje wykonanie dobrego oprogramowania.
- Jeśli prace nad projektem się opóźniają, wystarczy przydzielić mu więcej programistów.
- Jeśli oprogramowanie wykona ktoś inny (outsourcing), to można pozbyć się problemów.

Mity klientów

na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- Ogólne określenie wymagań klienta wystarczy do rozpoczęcia prac, a szczegóły można dopracować później.
- Wymagania wobec oprogramowania wciąż się zmieniają, ale jest ono elastyczne i łatwo je zmieniać.

Mity informatyków

na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- Po napisaniu i uruchomieniu oprogramowania praca jest skończona.
- Dopóki oprogramowanie nie działa, nie można ocenić jego jakości.
- Jedynym wynikiem pracy nad oprogramowaniem jest działający program komputerowy.
- Inżynieria oprogramowania zmusi nas do tworzenia przepastnych, zbędnych dokumentów i spowolni pracę.

Wybrane dziedziny zastosowania produktu

- **System informatyczny.**
- System czasu rzeczywistego.
- System wbudowany.
- System sztucznej inteligencji.
- Oprogramowanie systemowe.
- Oprogramowanie inżynierskie.
- Oprogramowanie naukowe.
- Oprogramowanie sieciowe.

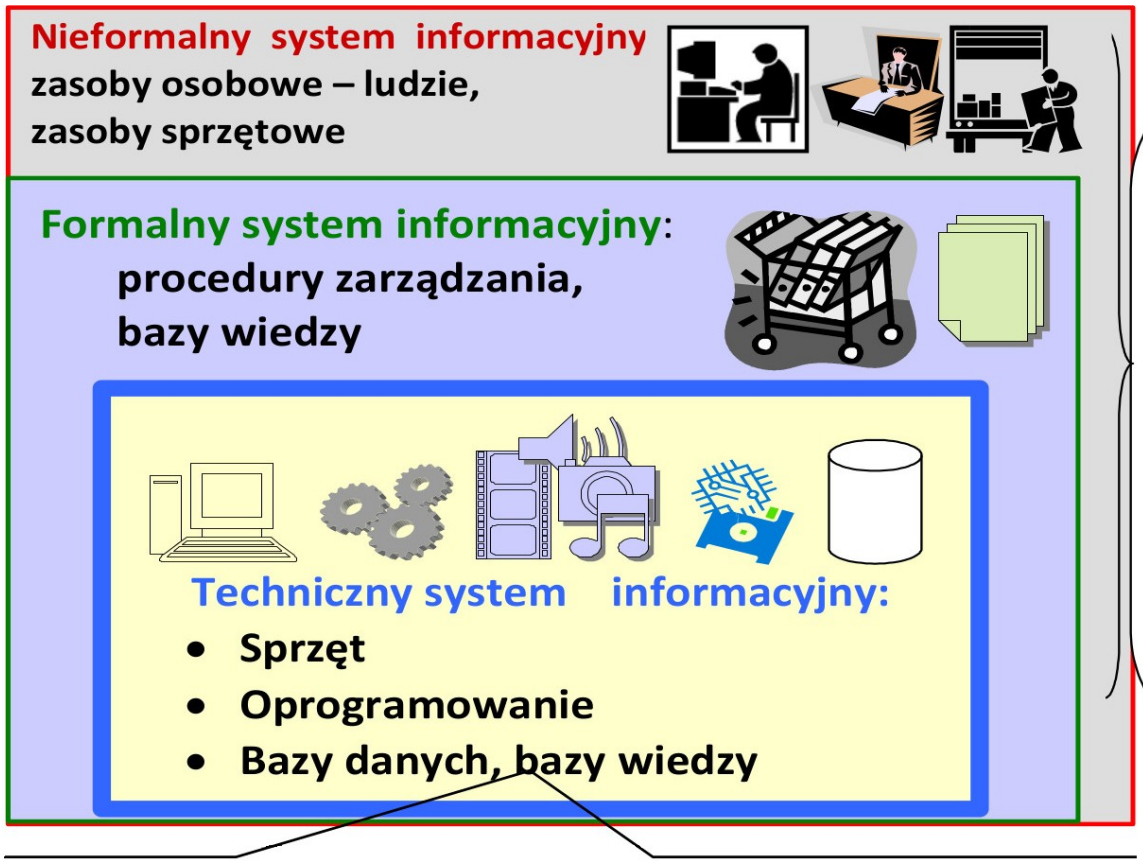
System Informatyczny

- **System wspierający działanie użytkownika (organizacji) w realizacji biznesowych zadań** przez:
  - przechowywanie, przesyłanie, zarządzanie i przetwarzanie danych (artefakty, np. informacje);
  - zarządzanie i wykonywanie funkcji i procesów organizacyjnych (biznesowych) użytkownika (organizacji);
  - w celu spełnienia/spełniania określonych wymagań.

- Składa się z powiązanych ze sobą:
  - elementów nieformalnych (użytkownicy i ich role);
  - elementów formalnych (procedury organizacyjne, kontrola bezpieczeństwa w sensie security, bazy wiedzy, tutoriale itp.);
  - elementów technicznych (sprzęt, oprogramowanie systemowe, bazy danych itp.).

System Informatyczny

źródło: Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr
na podst. Inżynieria Systemów Informacyjnych, P.B. Davies



System informatyczny jest to zbiór powiązanych ze sobą elementów nieformalnych, formalnych i technicznych, którego funkcją jest przetwarzanie danych przy użyciu techniki komputerowej

Techniczny system informacyjny

- zorganizowany zespół środków technicznych (komputerów, oprogramowania, urządzeń teletransmisyjnych itp.)
- służący do gromadzenia, przetwarzania i przesyłania informacji.

5

Narzędzia w tworzeniu oprogramowania

Zadania dla używanego sprzętu i programów

na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- Automatyzacja procesu tworzenia oprogramowania.
- Funkcjonalne wsparcie cyklu życia oprogramowania:
  - specyfikacja wymagań;
  - wizualne modelowanie i projektowanie
    (kontrola poprawności diagramów, nawigacja po elementach modeli);
  - programowanie;
  - testowanie;
  - inżynieria wprost;
  - inżynieria odwrotna.

Zadania dla używanego sprzętu i programów

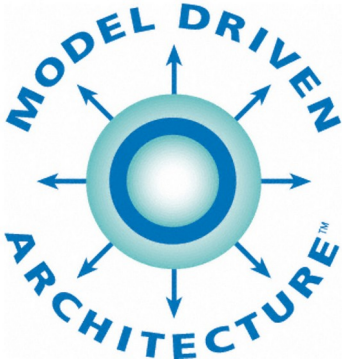
na podst. Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr

- Standaryzacja procesu i produktów tworzenia oprogramowania.
- Współpraca między narzędziami (import, eksport, integracja modeli).
- Zarządzanie jakością oprogramowania.
- Monitorowanie i kontrolowanie postępu prac.
- Repozytorium i inne wsparcie pracy zespołowej.

6

Architektura Sterowana Modelem

Architektura Sterowana Modelem
/Model Driven Architecture, MDA/



na podst. MDA - The Architecture Of Choice For A Changing World

- Metodyczne podejście do inżynierii oprogramowania (projektowania, rozwijania i implementowania oprogramowania).
  - Oparte na budowie modeli (abstrakcji systemu) i ich transformacji.
  - Opracowane przez *OMG Standards Development Organization.*
- Graficzne modelowanie funkcjonalności i biznesowego zachowania:
  - głównie w języku UML,
  - niezależnie od wybranej technologii ich implementacji.
- Izolacja rdzenia oprogramowania od techniki jego implementacji i cyklu zmian:
  - logika biznesowa pozostaje zgodna z biznesowymi wymaganiami,
  - a technologia oprogramowania może się rozwijać i zmieniać.
- **„Wszystko jest modelem”**

Poziomy abstrakcji modelu

1. Koncepcja

2. Model konceptualny

3. Model implementacyjny

4. Implementacja

5. Wdrożenie

MDA

Poziomy abstrakcji modelu

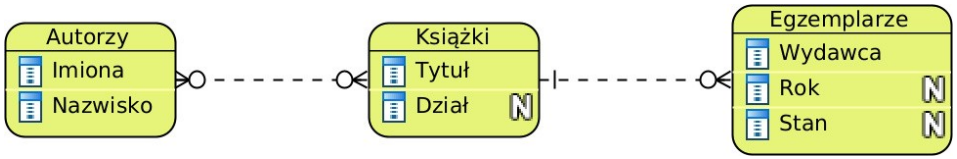
1. **Koncepcja** (CIM /Computation-Independent Model/)
   - perspektywa koncepcji,
   - model niezależny od środków informatycznych,

2. **Model konceptualny** (PIM /Platform-Independent Model/)
   - perspektywa specyfikacji,
   - model niezależny od platformy programistycznej i systemowej,

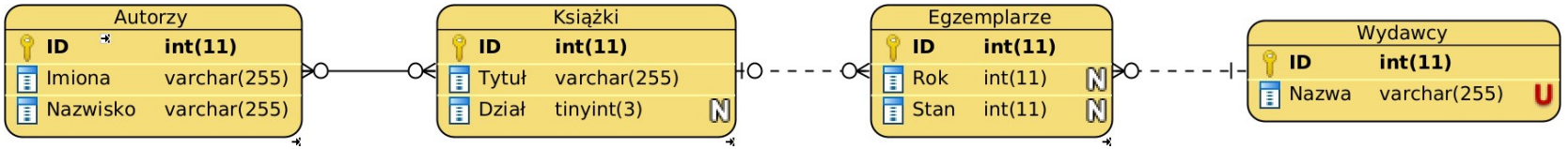
3. **Model implementacyjny** (PSM /Platform-Specific Model/)
   - perspektywa implementacji,
   - model specyficzny dla platformy programistycznej i systemowej,
   - na podstawie tego modelu powstaje kod oprogramowania.

Poziomy abstrakcji modelu (relacyjna baza danych)

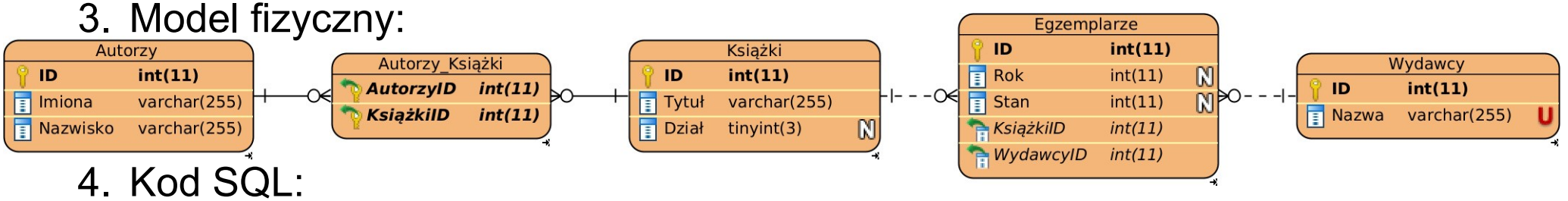
1. Model konceptualny:



2. Model logiczny:



3. Model fizyczny:



4. Kod SQL:

```
CREATE TABLE Autorzy (
 ID int(11) NOT NULL AUTO_INCREMENT,
 Imiona varchar(255) NOT NULL,
 Nazwisko varchar(255) NOT NULL,
 PRIMARY KEY (ID));

CREATE TABLE Książki (
 ID int(11) NOT NULL AUTO_INCREMENT,
 Tytuł varchar(255) NOT NULL,
 Dział tinyint(3),
 PRIMARY KEY (ID));

CREATE TABLE Autorzy_Książki (
 AutorzyID int(11) NOT NULL,
 KsiążkiID int(11) NOT NULL,
 PRIMARY KEY (AutorzyID, KsiążkiID));
```

```
ALTER TABLE Autorzy_Książki ADD CONSTRAINT
 FKAutorzy_Ks998910 FOREIGN KEY (AutorzyID)
 REFERENCES Autorzy (ID);

ALTER TABLE Autorzy_Książki ADD CONSTRAINT
 FKAutorzy_Ks72080 FOREIGN KEY (KsiążkiID)
 REFERENCES Książki (ID);
```

7

Język UML i SysML

UML

- **Unified Modeling Language (UML)**
  – graficzny język modelowania i analizowania systemów.

SysML

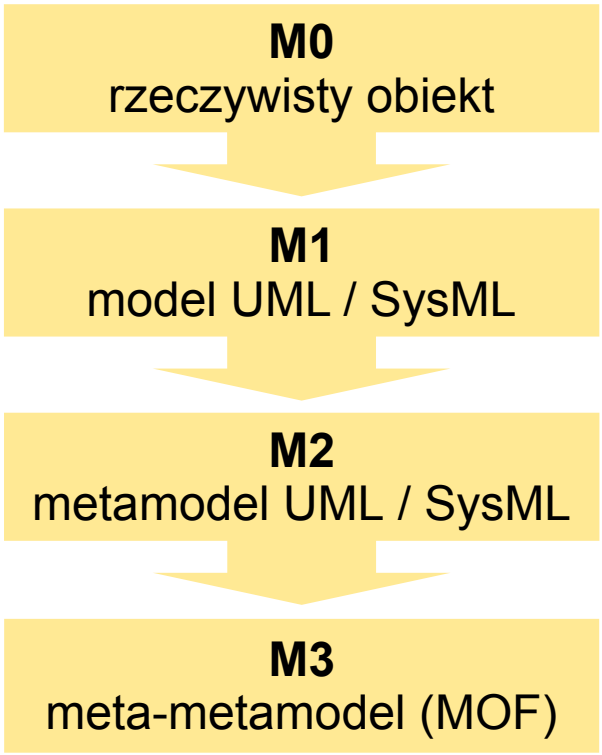
- **Systems Modeling Language (SysML)**
  – profil, rozszerzenie języka UML.

UML i SysML

- Służą do specyfikacji, analizowania, projektowania i weryfikowania złożonych systemów, w tym systemu informatycznego:
  - definicja struktury i zachowania systemu i jego części na różnych poziomach abstrakcji i na różnych etapach cyklu życia oprogramowania: pół-/formalna, standaryzowana, graficzna i precyzyjna;
  - wsparcie pracy zespołowej nad systemem;
  - dekompozycja systemu (jego struktury i zachowania) – pokonanie jego złożoności;
  - tworzenie i stosowanie wzorców projektowych;
  - projektowanie testów oprogramowania.
- Posiadają elastycznie określoną składnię (reprezentację graficzną) i semantykę (jej zastosowanie).
  - W elemencie diagramu ma znaczenie: jego kształt, położenie, sposób powiązania z innym elementem.
  - W elemencie diagramu NIE ma znaczenia: jego kolor (z wyjątkiem koloru grotów kilku związków – biały / czarny).

4-warstwowa architektura OMG

- **M0 – rzeczywisty obiekt** (np. *Książka*):
  - modelowany system lub jego element,
  - ↓ instancja modelu UML.
- **M1 – model UML** (np. *klasa Książka*):
  - abstrakcja rzeczywistości,
  - ↓ instancja metamodelu UML.
- **M2 – metamodel UML** (np. *klasa*):
  - definicja składni modelu,
  - ↓ instancja meta-metamodelu UML.
- **M3 – meta-metamodel (MOF)**:
  - definicja struktury metamodelu,
  - przy pomocy języka metamodelu (diagram klas).

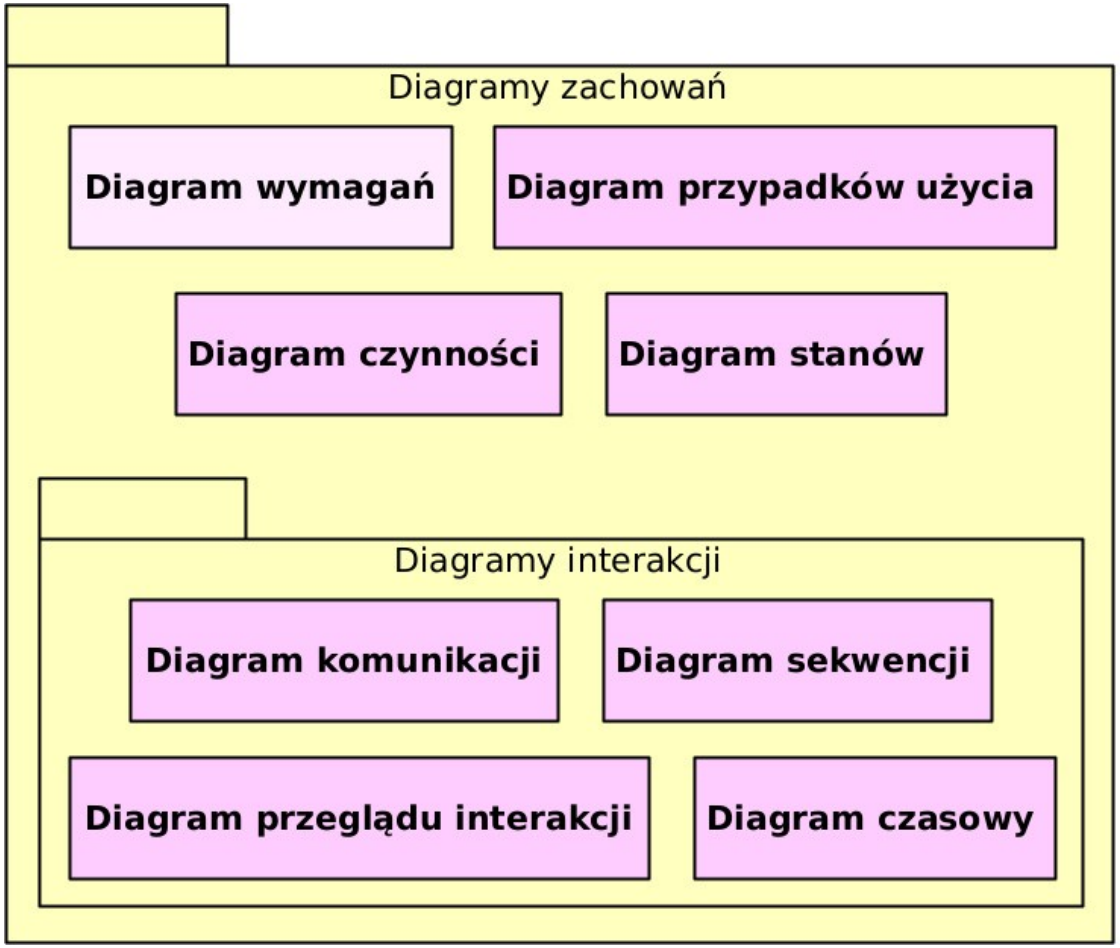


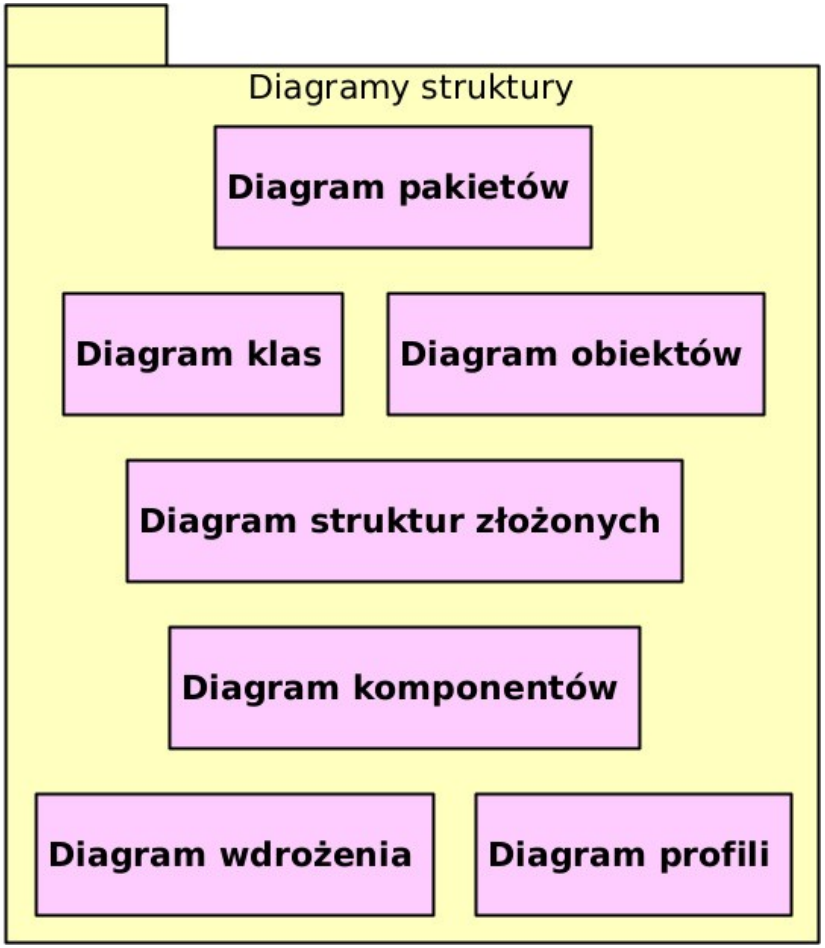
8

Diagramy UML i SysML

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

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

Diagramy zachowań w procesie wytwarzania oprogramowania

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

Diagramy struktury w procesie wytwarzania oprogramowania

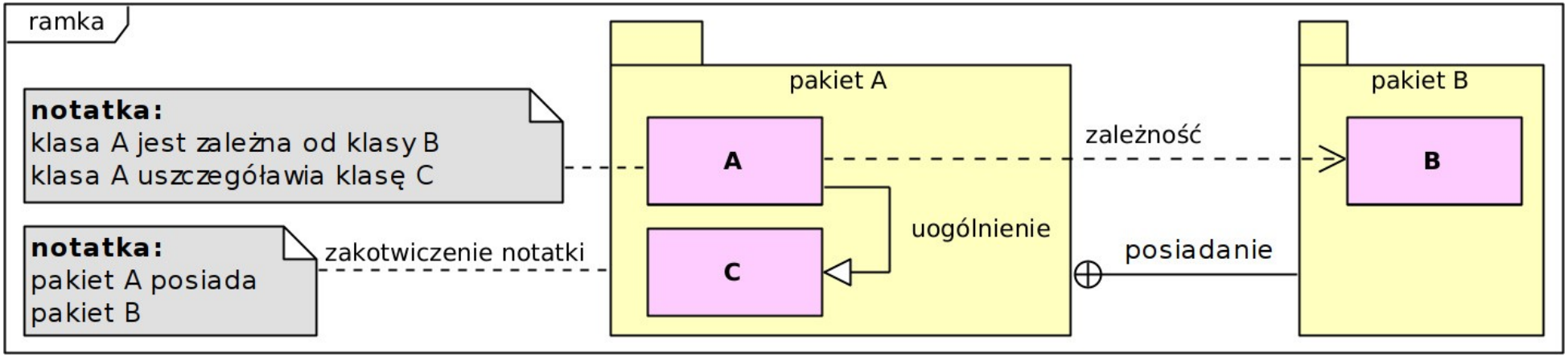
- **Diagram pakietów**
  - logiczne uporządkowanie elementów modelu oprogramowania (np. klas).
- **Diagram klas**
  - statyczna obiektowa struktura oprogramowania (klasy i relacje między nimi).
- **Diagram obiektów**
  - instancje klas i relacje między nimi.
- **Diagram struktur złożonych**
  - wewnętrzne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

Diagramy struktury w procesie wytwarzania oprogramowania

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

Elementy UML stosowane na różnych diagramach

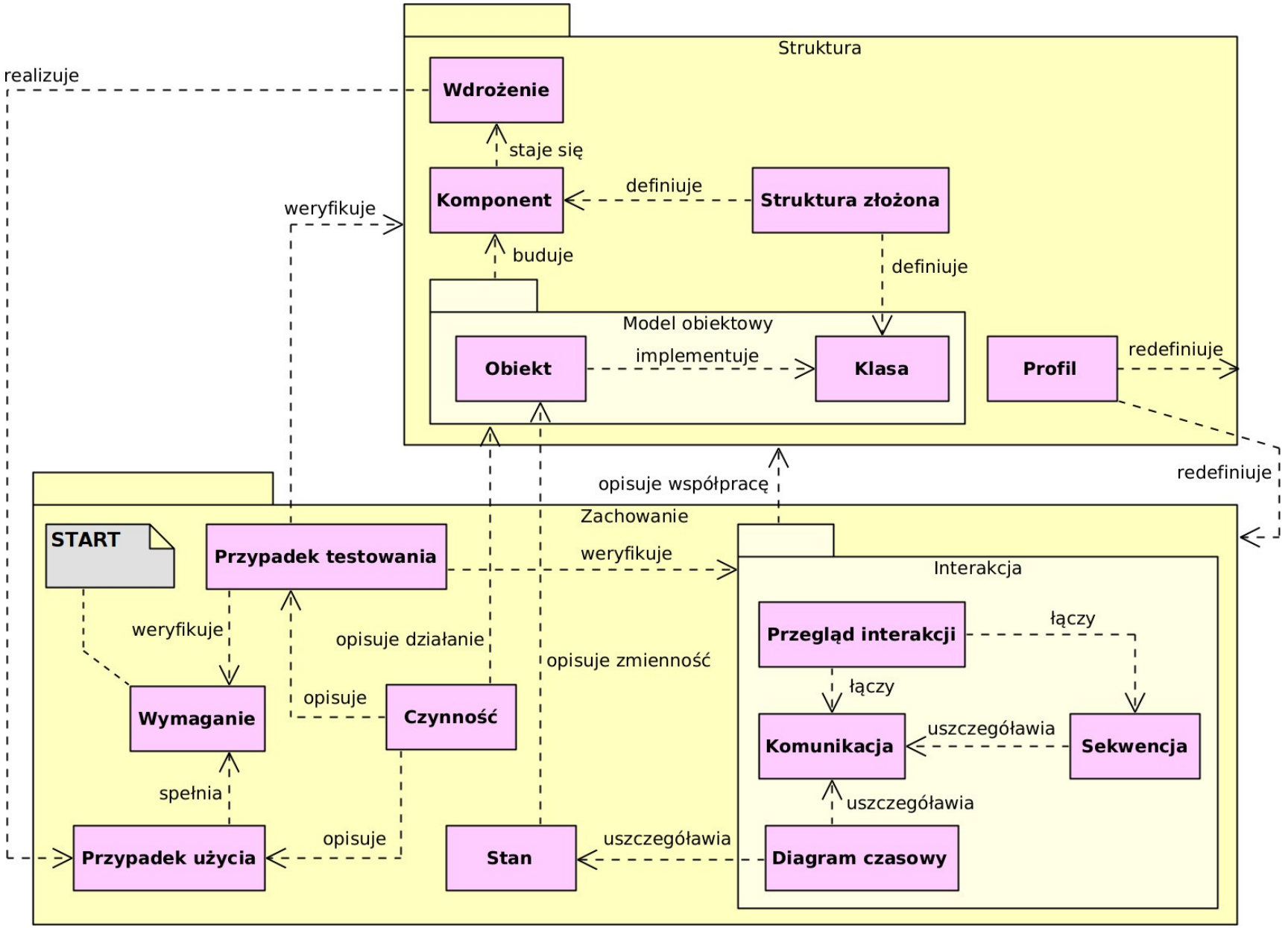
- **notatka** /note/ – słowne wyjaśnienie diagramu lub elementu, do którego jest zakotwiczona;
- **ramka** /frame/ – semantyczne grupowanie elementów;
- **pakiet** /package/ – funkcjonalne grupowanie elementów;
- **relacja zależności** /dependency/ – wskazujący element jest zależny od wskazanego elementu;
- **relacja uogólnienia** /generalization/ – wskazujący element jest uszczegółowieniem wskazanego elementu;
- **relacja posiadania** /containment/ – element z grotem ⊕ na końcu relacji zawiera element z drugiego jej końca.



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.

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.

9

Opinie o UML

Opinie o stosowaniu UML

na podst. An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles, A.M. Fernández-Sáez et at., Empir Software Eng 23 (2018)

- **W związku z tworzeniem i konserwacją oprogramowania:**
  - ograniczona aktualizacja modeli do aktualnej wersji UML,
  - „cel projektowy” jest ważniejszy niż „plan projektowy” (abstrahowanie od szczegółów projektu),
  - ulepszenie przedwdrożeniowego projektu może pogorszyć synchronizację modelu z już wykonanym kodem,
  - stosowanie jedynie modeli inżynierii odwrotnej może zniekształcić główne wymagania oprogramowania.

Opinie o stosowaniu UML

na podst. An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles, A.M. Fernández-Sáez et at., Empir Software Eng 23 (2018)

- **Korzyści dla procesu tworzenia oprogramowania:**
  - lepsza komunikacja
    szczególnie podczas globalnego procesu tworzenia oprogramowania
    lub podczas outsourcingu,
  - ulepszenie projektu przed wdrożeniem
    (łatwość jego recenzowania),
  - możliwość utrwalania wiedzy o utworzonym oprogramowaniu,
  - możliwość diagnozowania problemów oprogramowania.
- **Korzyści dla jakości oprogramowania:**
  - 89% inżynierów uważa, że
    UML poprawia jakość oprogramowania
    dzięki użyciu go w procesie jego wytwarzania.

Opinie o stosowaniu UML

na podst. An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles, A.M. Fernández-Sáez et at., Empir Software Eng 23 (2018)

- **Przeszkody dla procesu tworzenia oprogramowania:**
  - błędna interpretacja wytycznych Manifestu Agile („działające oprogramowania zamiast obszernej dokumentacji”, ”kod źródłowy to dokumentacja”),
  - brak projektantów w zespołach tworzenia oprogramowania,
  - różne zrozumienie roli diagramów UML i uczestników zespołu (projektanci tworzą modele UML, a programiści kod),
  - trudność ustalenia poziomu szczegółowości diagramów UML i ich odwzorowania w kodzie oprogramowania.

Opinie o stosowaniu UML

na podst. An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles, A.M. Fernández-Sáez et at., Empir Software Eng 23 (2018)

- **Najlepsze praktyki w modelowaniu i używaniu diagramów UML:**
  - cel: użycie diagramów UML i wynikający z niego poziom ich szczegółowości;
  - proces: aktualizacja dokumentacji związanej z aktualnym stanem kodu oprogramowania;
  - narzędzia: do poprawnej i szybkiej aktualizacji dokumentacji związanej z aktualnym stanem kodu oprogramowania;
  - szkolenia: z używania UML, zwłaszcza dla programistów;
  - standaryzacja i zarządzanie: na centralnym poziomie firmy aby ujednolicić praktyki stosowania UML.

Wady i zalety stosowania UML

na podst. Exploring Costs and Benefits of Using UML on Maintenance: Preliminary Findings of a Case Study in a Large IT Department, Fernández-Sáez et at., EESSMod@ MoDELS (2013)

- **Zalety:**
  - wysoki poziom abstrakcji;
  - wysoka przydatność w modelowaniu systemu (szczególnie dla systemu zorientowanego obiektowo);
  - prezentowanie różnych punktów widzenia;
  - standaryzacja (międzynarodowy język, ułatwia komunikację między firmami);
  - lepsza precyzja procedur;
  - wsparcie sposobu modelowania;
  - poprawa dokumentacji.

Wady i zalety stosowania UML

na podst. Exploring Costs and Benefits of Using UML on Maintenance: Preliminary Findings of a Case Study in a Large IT Department, Fernández-Sáez et at., EESSMod@ MoDELS (2013)

- **Wady:**
  - wysoki poziom abstrakcji;
  - niejasna notacja i semantyka;
  - brak standaryzacji modelowania (nazewnictwo, warstwowość oprogramowania, poziom szczegółowości modelowania, związek między modelem a kodem);
  - statyczny model (brak jego uruchamiania);
  - brak punktu widzenia użytkownika;
  - niska zdolność projektowania SOA (Service-Oriented Architecture);
  - brak rozdzielenia tego, co trzeba zrobić, od tego, jak to zrobić;
  - trudność modelowania złożonych elementów;
  - za mała siła wyrażania.

Wady i zalety stosowania UML

na podst. Exploring Costs and Benefits of Using UML on Maintenance: Preliminary Findings of a Case Study in a Large IT Department, Fernández-Sáez et at., EESSMod@ MoDELS (2013)

- **Wnioski:**
  - Pozytywny stosunek do modelowania mają: architekci, programiści i inżynierowie serwisu.
  - Negatywny stosunek mają osoby, nie znały lub słabo znały UML.
  - Niektóre wady pozwoliły zidentyfikować niedopracowane procedury użycia UML do budowy poszczególnych modeli oprogramowania.

Temat następnej prezentacji

Modelowanie wymagań – diagram wymagań i diagram przypadków użycia