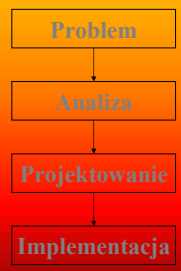
Object Modeling Technique

OMT

Podstawowe założenia OMT

- Postawienie problemu
- Zrozumienie wymagań - **analiza**
- Planowanie rozwiązania - **projektowanie**
- **Implementacja** w języku programowania



Charakterystyka OMT

Podejście projektowania bazujące na OMT opiera się na procesie modelowania dziedziny aplikacji:

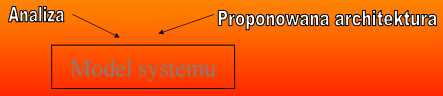
- Identyfikuje i adoptuje byty świata rzeczywistego jako elementy modelu domeny
- Koncentruje się tylko na istotnych właściwościach rozwiązywanego problemu
- Kolejne etapy projektowania uszczegóławiają model wprowadzony w fazie analizy
- Nie wprowadza ograniczeń wynikających z nieelastyczności języków programowania – końcowa realizacja może być wyrażona w języku obiektowym, proceduralnym, implementacji struktury bazy danych, jak również implementacji sprzętowej

Analiza

- Koncentracja na tym, co ma być zrobione, a nie jak ma być zrobione
- Modelowanie sytuacji świata rzeczywistego i wskazanie najważniejszych ich właściwości
- Model powinien zostać zweryfikowany przez ekspertów aplikacji, a nie programistów
- Nie używa pojęć związanych z implementacją
- Posługuje się pojęciami z zakresu dziedziny aplikacji, a nie struktur danych
- Stanowi punkt wyjścia kolejnych etapów tworzenia oprogramowania

Projektowanie systemowe

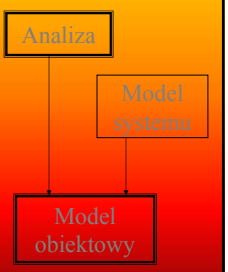
- Ustalenie architektury systemu
  np. wielowarstwowa
- Dekompozycja systemu na podsystemy



- Wybór strategii rozwiązania problemu
- Decyzje odnośnie wydajności, skalowalności i optymalności wybranych wskaźników
- Rozsądna alokacja zasobów

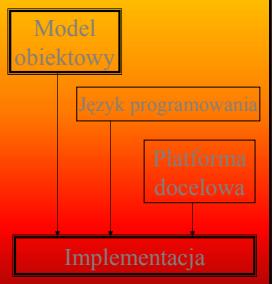
Projektowanie obiektowe

- Budowa i projektowanie modelu bazującego na modelu domeny ale wzbogaconym o szczegóły implementacyjne
- Myślenie w kategoriach struktur danych i algorytmów, optymalizacja wybranych właściwości
- Obiektowy model dziedziny i model komputerowy są te same, ale usytuowane w innych wymiarach (używają tej samej notacji)



Implementacja

- Translacja modelu obiektowego z poprzedniego kroku do języka programowania, bazy danych lub implementacji sprzętowej
- Implementacja powinna być automatyczna z minimum nakładu pracy
- Powinna być elastyczna i rozszerzalna



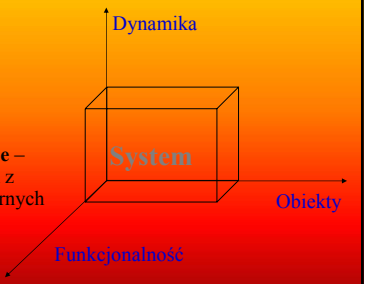
Wnioski

- Zachowanie struktury modeli pomiędzy etapami projektowania – w kolejnych krokach uszczegóławiane są elementy modelu dziedziny
- Zróżnicowane są tylko poziomy abstrakcji
- Część struktur danych wprowadzana jest sztucznie (np.: lista, drzewo, mapa) i nie ma odpowiednika w rzeczywistości
- Realizuje podejście „Top-Down" – nie jest metodą iteracyjną
- Nie obejmuje fazy testowania aplikacji

Modele w OMT

- Model obiektowy
- Model dynamiki
- Model funkcjonalny

Powyższe modele są wzajemnie **ortogonalne** – opisują ten sam system z różnych, komplementarnych perspektyw



Modele budowane są w fazie **analizy**

Model obiektowy

- Wyraża statyczną strukturę obiektów, powiązań pomiędzy nimi i zobowiązań
- Określa podmioty operowania dla modeli *dynamiki* i modeli *funkcjonalnych*
- Najbardziej stabilny model procesu projektowania
- Składa się z diagramu obiektów w którym:
  - Obiekty są wierzchołkami grafu
  - Krawędzie obrazują relacje pomiędzy obiektami

Obiekty i klasy

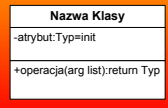
- **Klasa** – grupuje obiekty o podobnych właściwościach, zachowaniu, relacji do pozostałych klas i podobnej semantyce
- Jest wzorcem, na podstawie którego budowane są instancje klasy

- **Obiekt** - rzeczownik, który ma sens w kontekście aplikacji

- Obiekt „wie" do jakiej klasy przynależy

Atrybuty i operacje

- Atrybuty – pamiętają stan obiektu klasy
  - Każdy atrybut posiada wartość dla każdej instancji obiektu
  - Nazwy atrybutów są unikalne w obrębie klasy
  - Atrybutami są typy proste nie posiadające identyfikacji



- Operacja – funkcja lub transformacja, która może być wykonana przez obiekt lub na obiekcie
  - Posiada powiązany obiekt jako obowiązkowy argument
  - Jest operatorem polimorficznym
  - Może posiadać dodatkowe argumenty

Połączenie i Związek

- Wyodrębnione elementy modelu dziedziny łączone są w *grafy instancji* obiektów i *diagramy klas*
- Połączenie – fizyczne lub pojęciowe połączenie pomiędzy **instancjami** obiektów

- Związek – grupa połączeń o jednakowej strukturze i semantyce – łączy **klasy** obiektów
  - Przeważnie są dwukierunkowe

Związki mnogościowe - binarne

Specyfikują jak wiele instancji obiektów jednej klasy może być połączonych z pojedynczą instancją innej klasy

- Pojedyncze:

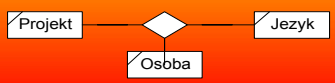
- Opcjonalne

- Wielokrotne

- Na końcu powiązania może wystąpić liczba określająca jego liczebność

Związki wyższego rzędu

- Powiązanie „ternary” jest jednostką niepodzielną – bez straty informacji nie daje się przedstawić jako zestaw powiązań binarnych
- Diagram klas



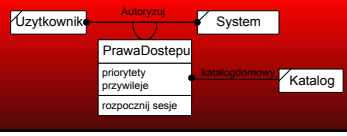
- Diagram obiektów

Atrybuty połączenia

Podobnie jak *atrybut* jest własnością klasy, tak **atrybut połączenia** jest właściwością związku pomiędzy klasami

- Atrybut połączenia jako własność

- Atrybut połączenia jako klasa

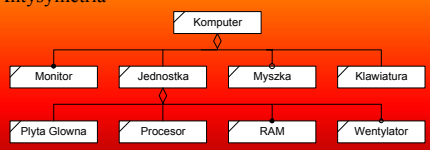


Kwalifikacja

- Specjalny atrybut redukujący wielokrotność powiązania – zmniejsza ilość kombinacji powiązania
- Stosowany dla powiązań typu: jeden-do-wielu i wiele-do-wielu
- Stanowi kontekst w którym nazwa nabiera sensu
- Stosowana jest głównie przy przeszukiwania zbiorów

Agregacja

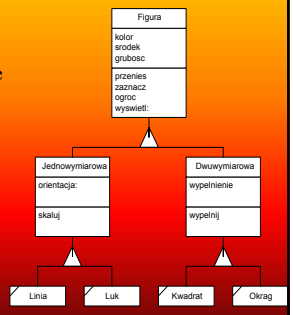
- Specjalna forma *powiązania*
- Agregacja jest relacją zawierania i cechuje ją:
  - Przechodniość
  - Antysymetria



- Agregacja podkreśla, że jeden obiekt jest częścią innego – wyraża zależność „ma”

Dziedziczenie i generalizacja

- Dzielenie funkcjonalności z zachowaniem indywidualności klasy
- Instancja klasy jest jednocześnie instancją wszystkich swoich przodków
- Przechodniość pomiędzy poziomami dziedziczenia
- Zastosowanie przy budowie modelu pojęciowego i implementacji
- Dziedziczenie może być rozpatrywane jako rozszerzenie lub jako restrykcja klasy bazowej

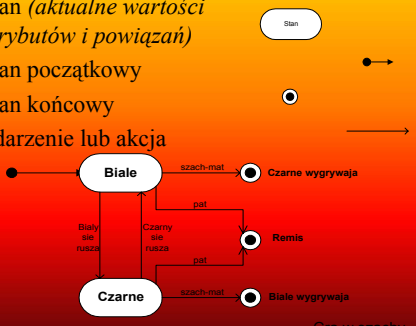


Model dynamiki

- Obrazuje zmiany stanu systemu w czasie, dotyczy aspektu sterowania systemem
- Operuje na *modelu obiektowym*
- Opisuje aspekt „logiki” obiektu
- Ustala zbiór możliwych wartości dla atrybutów obiektu
- Łączy zdarzenia ze stanami obiektów
- Składa się z diagramów stanów w którym:
  - Jeden diagram (maszyna stanowa) odnosi się do jednej klasy o istotnej dynamice
  - Wierzchołki są zbiorem osiąganych stanów (wartości atrybutów i powiązań danej klasy)
  - Krawędzie obrazują zmiany stanów w odpowiedzi na zdarzenia (operacje modelu obiektowego) – zjawisko zachodzące w czasie i przestrzeni

Diagram stanów

- Stan *(aktualne wartości atrybutów i powiązań)*
- Stan początkowy
- Stan końcowy
- Zdarzenie lub akcja



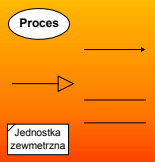
Gra w szachy

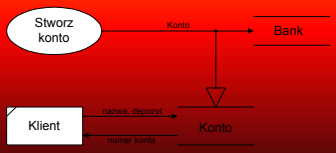
Model funkcjonalny

- Obrazuje transformację wartości danych *modelu obiektowego*
- Specyfikuje rezultat obliczeń bez podawania jak i kiedy został on osiągnięty
- Uwydatnia znaczenie **operacji** modelu obiektowego i **zdarzeń** modelu dynamiki
- Uwzględnia: dane wejściowe, wyjściowe i tymczasowe wartości obliczane przez system
- Składa się z diagramów przepływu danych w którym:
  - Wierzchołki – procesy transformujące dane
  - Krawędzie obrazują przepływ danych

Diagram przepływu danych (DFD)

- Proces *(np. funkcja)*
- Przepływ danych
- Ujście danych
- Magazyn danych
- Jednostka zewnętrzna





DFD - uczestnicy

- **Proces** – zazwyczaj jest to funkcja obliczająca pewne wartości i zwracająca je do systemu
- **Przepływ danych** – łączy wyjście jednego procesu z wejściem następnego, nie zmienia wartości przekazywanych danych, dopuszcza rozgałęzienia i selekcję danych
- **Jednostka zewnętrzna** (aktor) – obiekt aktywny, który steruje przepływem danych
- **Magazyn danych** – przechowuje dane w celu późniejszego ich wykorzystania

Powiązania pomiędzy modelami

Każdy z modeli opisuje jeden z aspektów systemu, ale zawiera odwołania do pozostałych modeli:

- Model obiektowy opisuje struktury danych na których operuje model funkcjonalny i dynamiki
- Operacje w modelu obiektowym korespondują ze zdarzeniami z modelu dynamiki i funkcjami z modelu funkcjonalnego
- Model dynamiki obrazuje strukturę logiki obiektu, opisuje zależności pomiędzy wartościami atrybutów obiektu i transformacjami jego stanu
- Model funkcjonalny dostarcza funkcji wywoływanych w operacjach i zdarzeniach oraz ustala ograniczenia w wartościach obiektu

OMT a inne standardy modelowania

- *Unified Modeling Language* (UML) adoptuje bez większych modyfikacji wszystkie 3 modele proponowane w OMT
- *Model Driven Architecture* (MDA) jest rozwinięciem stosu czynności tworzenia oprogramowania z OMT
- Podobnie jak „*The Booch Methodology*" – OMT koncentruje się głównie na fazie analizy i projektowania

Przykład

»Stacja kontroli lotów

Motoryka przykładu

- Zamodelowanie symulatora lotniska i stacji kontroli lotów
- Lotnisko przyjmuje i startuje samoloty
- Start i lądowanie samolotu zależy od warunków pogodowych i stanu dostępnych pasów
- Stacja kontroli lotów kontroluje także loty tranzytowe

Obiekty dziedziny i ich odpowiedzialność

- **Lotnisko** – obiekt fizyczny posiadający strukturę organizacyjną i procedury obsługi
- **Samolot** – obiekty wymieniane i kontrolowane przez stacje, w danej placówce identyfikowany jest przez numer lotu
- **Pasy startowe** – wpływają na zdolności przyjmowania i startowania samolotów
- **Stacja meteo** – obiekt pomocniczy lotniska, modeluje warunki meteorologiczne

Diagramy klas

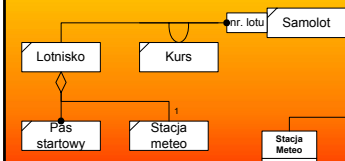


Diagram kooperacji

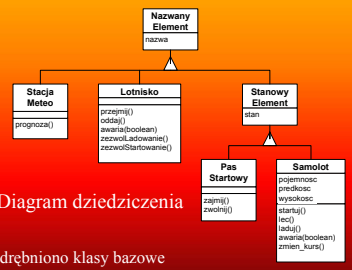


Diagram dziedziczenia

W diagramie dziedziczenia wyodrębniono klasy bazowe „Nazwany Element" i „Stanowy Element"

Diagramy stanów

- Wyodrębnione stany **lotniska**:
  - **Lądowanie** – kontrola nad samolotem podczas lądowania
  - **Startowanie** – kontrola nad samolotem podczas startowania
  - **Kontrola** – nadzór nad samolotami w strefie
  - **Oczekiwanie** – stan spoczynkowy
- Wyodrębnione stany **samolotu:**
  - **Startuje**
  - **Leci**
  - **Ląduje**

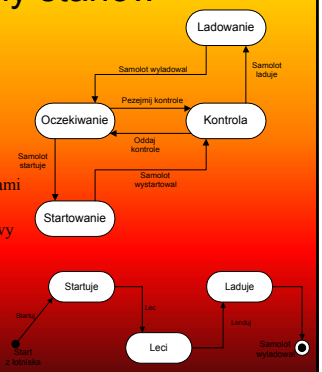
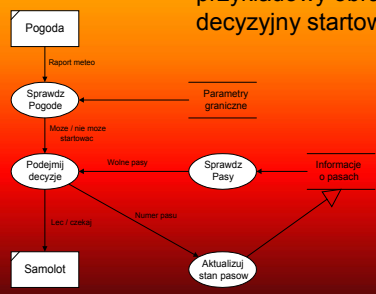


Diagram funkcjonalny

Przedstawiony diagram przykładowy obrazuje proces decyzyjny startowania samolotu



Wnioski

- OMT opisuje w sposób kompletny proces tworzenia oprogramowania
- Wydłuża czas życia systemu informatycznego poza czas życia technologii
- Wprowadza duży stopień abstrakcji w początkowej fazie projektowania, co pozwala na uniezależnienie się od języka programowania i platformy systemowej
- Większość wiodących obecnie standardów oprogramowania wywodzi się, bądź bezpośrednio adoptuje OMT (np. *UML, MDA*)
- Kładzie mały nacisk na proces implementacji
- Nie uwzględnia procesu *testowania* i zarządzania *cyklem życia* aplikacji

Koniec