Model-Driven Engineering

Radosław Klimek

2015-22

http://home.agh.edu.pl/rklimek

Modelowanie systemów

Problemy z budowaniem systemów

Typowe problemy związane z modelowaniem systemów informatycznych:

- niezrozumienie potrzeb użytkowników
- nieumiejętne reagowanie na zmiany w wymaganiach
- moduły, które do siebie nie pasują
- trudności z konserwacją systemu
- późne wykrywanie poważnych błędów
- słaba jakość
- słaba wydajność
- brak koordynacji w pracy zespołowej
- problemy z wytwarzaniem i wersjonowaniem

Konieczność modelowania

Ogólna zasada:
Modelowanie jest tym ważniejsze im projekt jest bardziej złożony i wymaga większych nakładów.

Modelowanie wizualne wspomaga zespół projektantów na różnych etapach w walce ze złożonością oprogramowania.

Modelowanie, model

Model jest uproszczeniem rzeczywistości, obrazuje wybrane cechy i zasady działania danego zjawiska lub przedmiotu.

- Uproszczenie pozwala pominąć nieistotne szczegóły (dla danego przedsięwzięcia).
- Z drugiej strony modelowanie winno uwypuklać te cechy, które są istotne z punktu widzenia realizowanego celu (struktura statyczna, własności dynamiczne, aspekty czasowe itp.).

Przyczyny budowania modeli

- ukrycia rzeczywistych szczegółów skomplikowanego systemu - w ten sposób projektanci systemów podczas ich tworzenia walczą ze znaczną złożonością tych systemów
- dostosowania poziomu abstrakcji do możliwości i potrzeb odbiorcy
- lepszego zrozumienia i poznania tworzonego systemu
- obserwacji i wnioskowania pracy, funkcjonowania rzeczywistego systemu badając jego model.

Znaczenie modelowania

Modelowanie to najważniejsza czynność sprzyjająca wdrożeniu dobrego oprogramowania.

Modele tworzymy w celu:

- Lepszego zrozumienia systemu.
- Specyfikacji pożądanej struktury i zachowania systemu.
- Opisania architektury systemu i panowania nad nią (dekompozycja, upraszczanie, ponowne użycie).
- Lepszego zarządzania ryzykiem.

Artefakt (*Artifact*) – informacja użyta lub wytworzona w czasie produkcji oprogramowania.

Kategorie modelowania

Notacje i metodyki odnośnie modeli można podzielić na:

- modelowanie do prowadzenia analiz i optymalizacji procesów i zdarzeń gospodarczych (**procesów biznesowych**).
- modelowanie do celów tworzenia oprogramowania.

Stasiak A. (red.) (2009), *Modelowanie systemów informatycznych w języku UML 2.1.*, Wydawnictwo Naukowe PWN, Warszawa

Jak modelować?

Model można opisać za pomocą:
- pseudokodu,
- kodu rzeczywistego,
- diagramów,
- obrazków,
- tekstowych opisów

Elementy tworzące język modelowania nazywane są **notacją**.

Aby zrozumieć znaczenie notacji potrzebny jest jej opis czyli semantyka języka, która jest określona w meta-modelu.

Zalety modelowania wg BRJ

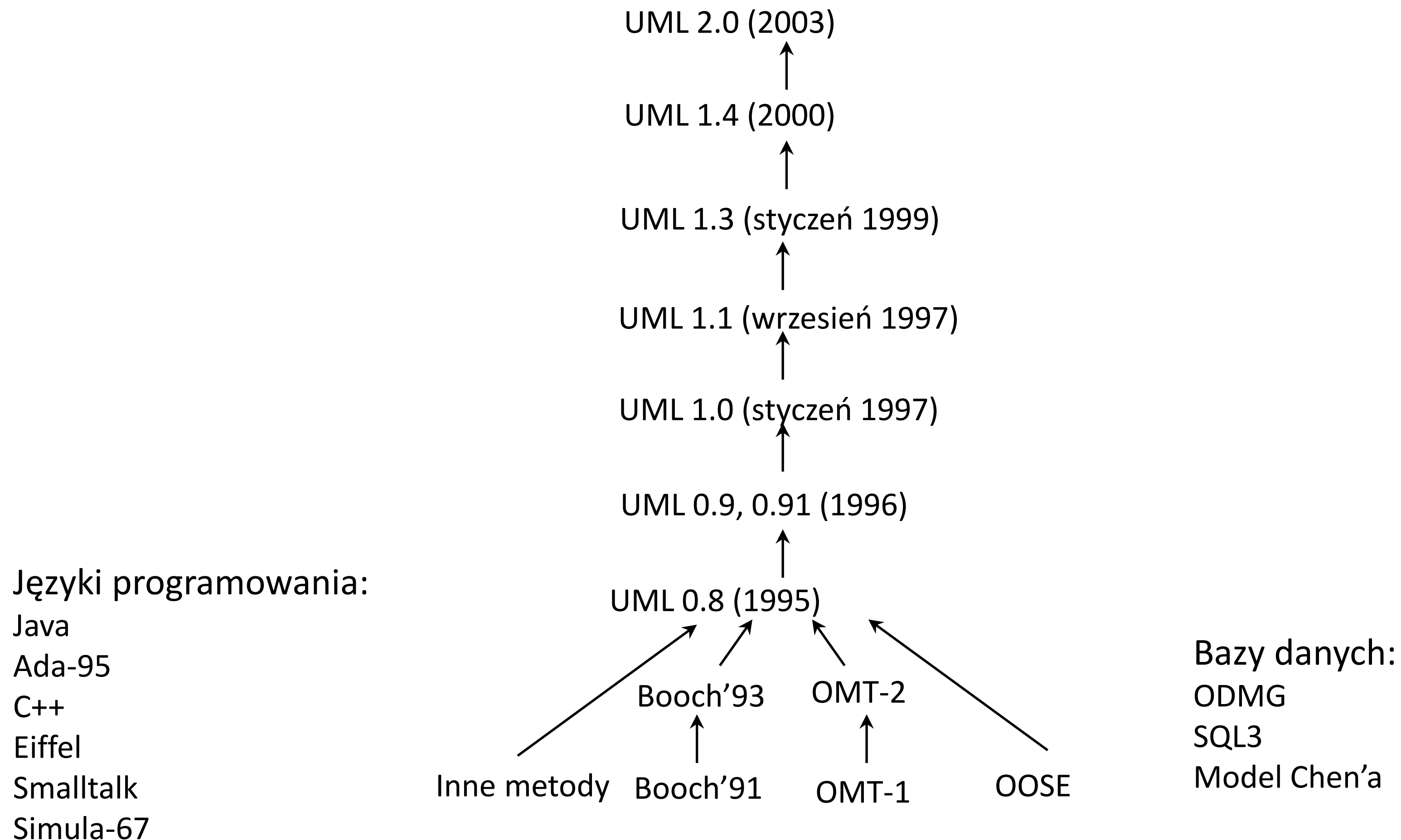
1. Decyzja o wyborze modelu ma istotny wpływ na sposób i jakość rozwiązania problemu.
2. Każdy model może charakteryzować się różnym poziomem szczegółowości.
3. Najlepiej jeśli model odwzorowuje rzeczywistość.
4. Zazwyczaj jeden model nie jest wystarczający. Niewielka liczba możliwie niezależnych modeli jest najlepszym rozwiązaniem przy modelowaniu nietrywialnego systemu.

Dwie (historyczne) strategie modelowania

Wyróżnia się dwa zasadnicze podejścia do tworzenia modeli oprogramowania:

1. **Algorytmiczne/strukturalne** – podstawowym blokiem jest procedura lub funkcja. Programiści koncentrują się na przepływie sterowania i podziale algorytmów.
2. **Obiektowe** – podstawowym blokiem jest obiekt lub klasa, która jest rozwijana w trakcie całego procesu.

Ewolucja metodyk obiektowych



Historyczne podejścia do modelowania obiektowego

Podejście OOAD (ang. *Object-Oriented Analysis and Design*) (G.Booch) – język modelowania skonstruowany na podstawie diagramów przedstawiających klasy, stany, przejścia stanów, interakcje, moduły oraz procesy. **Metodyka użyteczna w projektowaniu oraz określaniu związków ze środowiskiem implementacji, nie wspiera jednak dostatecznie dobrze fazy rozpoznania i analizy wymagań użytkowników.**

Podejście OMT (ang. *Object Modeling Technique*) (J. Rumbaugh) – jest to technika modelowania obiektów, notacja diagramów klas i obiektów. Model OMT posiadał również notację dla diagramów przedstawiających dynamiczne właściwości programu czyli diagramy sekwencji. **Metodyka użyteczna w modelowaniu dziedziny przedmiotowej, nie wspiera jednak w wystarczającym stopniu modelowania użytkowników systemu oraz implementacji**.`

Podejście OOSE (ang. *Object Oriented Software Engineering*) (I.Jacobson) – metoda ta jest skoncentrowana przede wszystkim na dokładnym uchwyceniu oraz zamodelowaniu dziedziny problemu. Modele: analizy, projektu, wdrożenia, testowania. **Metodyka użyteczna w modelowaniu aspektu użytkowników i cyklu życiowego systemu, nie wspiera jednak w wystarczającym stopniu implementacji**.

Inne historyczne podejścia

Istniało wiele podejść do modelowania oprogramowania. Każde miało swoją własną metodę tworzenia oprogramowania oraz język modelowania z własną odmienną notacją. Przykładowo:

- metodologia Odella,
- metodologia Shlaera i Mellora.

Grupy projektujące oprogramowanie musiały wybierać język modelowania, co mogło doprowadzić do wykluczenia z prac projektowych innych Zespołów, nieporozumień, itd.

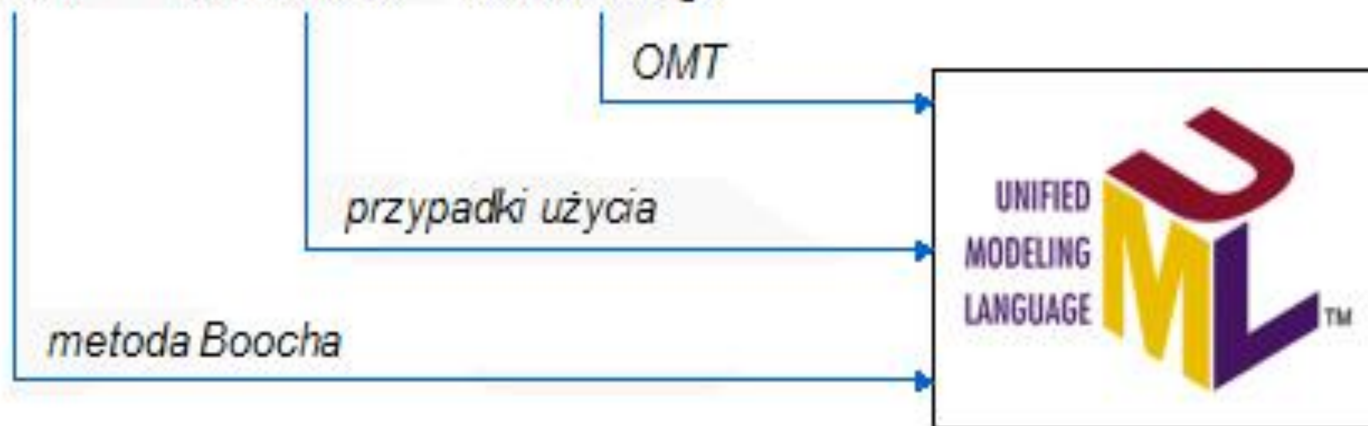
Standaryzacji dokonała grupa Object Management Group (OMG).
Członkami byli przedstawiciele 21 firm a przewodniczącym grupy
byli Mary Loomis i Ashwin Shah.

UML – trzy osoby

G. Booch

I. Jacobson

J. Rumbaugh



UML – zastosowania (dziedzinowo)

1. Systemy informacyjne przedsiębiorstw.
2. Usługi bankowe i finansowe.
3. Telekomunikacja.
4. Transport.
5. Przemysł obronny i lotniczy.
6. Sprzedaż detaliczna.
7. Elektronika i medycyna.
8. Systemy wspomagające badania naukowe.
9. Rozproszone usługi internetowe.

Język UML jest przeznaczony do:

- obrazowania dla komunikacji między członkami zespołu, śledzenia (język graficzny) i interpretacji oprogramowania,
- specyfikowania i tworzenia – inżynieria wprzód i wstecz,
- dokumentowania: wymagania, architektura, projekt, kod źródłowy, plany projektu, testy, prototypy, kolejne wersje artefaktów powstałych podczas budowania systemu informatycznego.

Modelowanie wizualne w UML

- Ułatwienie komunikacji między członkami zespołu
- Jeden wspólny język (UML)
- Ukrywanie bądź eksponowanie szczegółów
- Zarządzanie spójnością pomiędzy różnymi artefaktami systemu
- Zwiększanie poziomu abstrakcji
- Wsparcie narzędziowe
- Wiele punktów widzenia na system

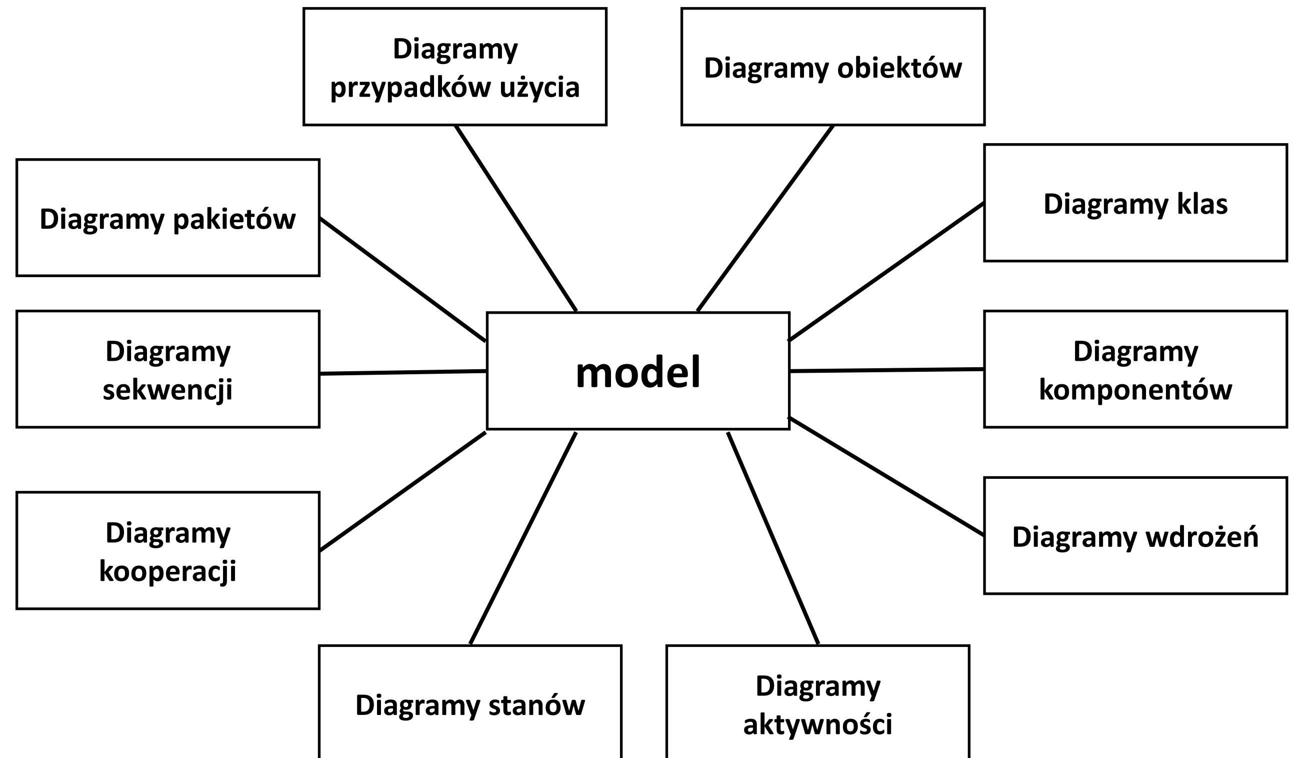
Tworząc system potrzebujemy spojrzenia na jego własności z różnych punktów widzenia – poprzez różne diagramy umożliwiające opis systemu z różnych perspektyw.

Zalety UML:
- Jest językiem formalnym (zależy z jakiego punktu widzenia)
- wykorzystanie zalet podejścia obiektowego
- Jest zwięzły, spójny i wyczerpujący
- Jest skalowalny
- Został utworzony na podstawie wniosków z doświadczeń
- Jest de facto standardem i ułatwia wymianę informacji pomiędzy przyszłymi użytkownikami systemu, menadżerami, analitykami, projektantami, programistami i testerami

Modelowanie w UML

Projektując system informatyczny rozpoczyna się przeważnie od tworzenia diagramów w następującej kolejności:

- przypadków użycia
- klas i obiektów
- behawioralnych
- implementacyjnych

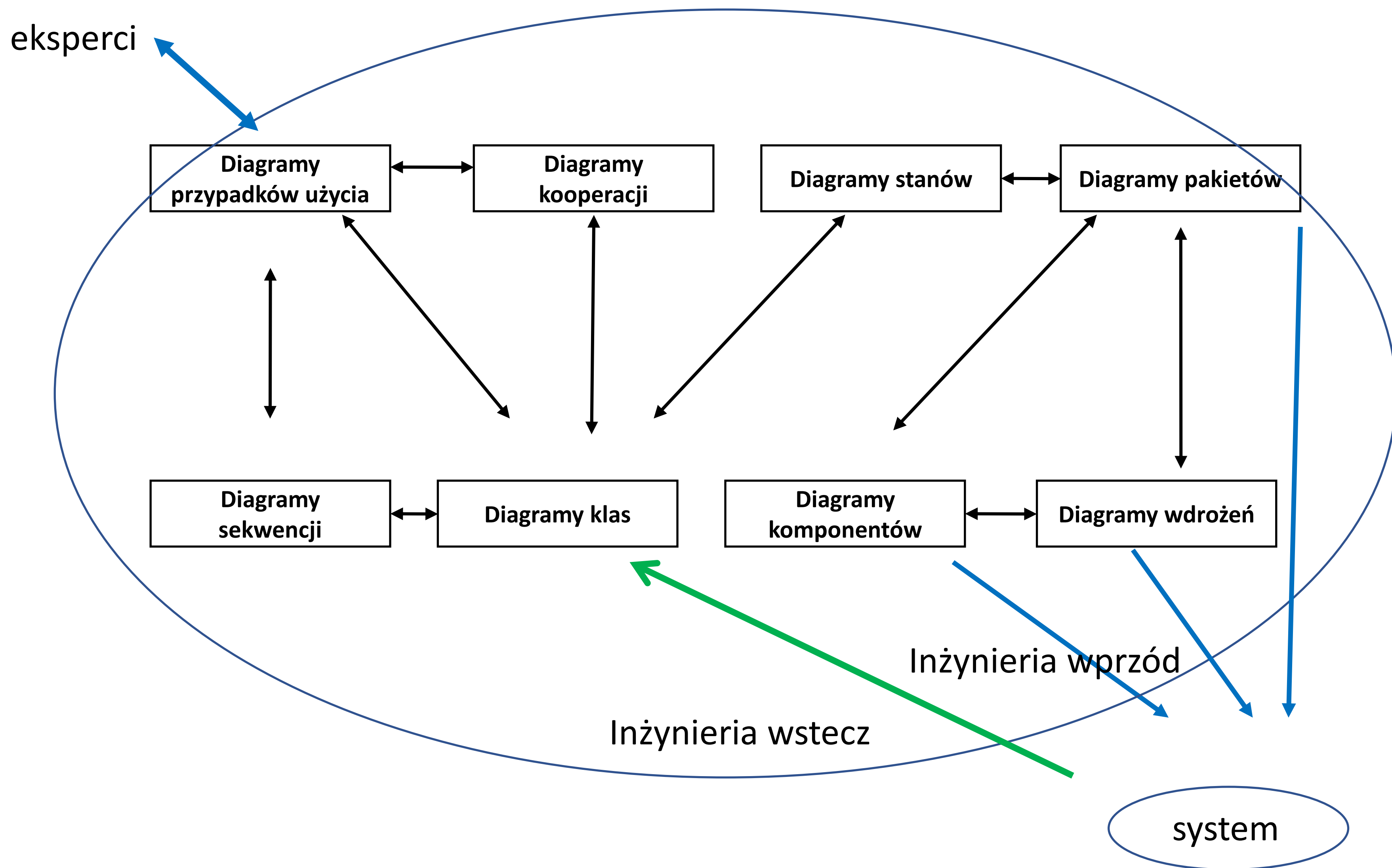


Cześć z nich bywa pomijana, zwłaszcza przy budowaniu niedużych systemów informatycznych.

Język UML - charakterystyka

- UML nie jest językiem programowania graficznego, jednak modele w nim zapisane mogą być wprost powiązane z wieloma językami programowania.
- Model utworzony w języku UML można przekształcić w taki język, jak Java, C++ czy Visual Basic, albo w tabele relacyjnej bazy danych.
- To przekształcenie umożliwia **inżynierię w przód** (ang. *forward engineering*), to znaczy generowanie kodu w języku programowania na podstawie modelu UML.
- Możliwe jest także odwrotne przekształcenie, czyli rekonstrukcja modelu na podstawie implementacji – **inżynieria wstecz** (ang. *reverse engineering*).
- Przy przejściu od modelu do kodu każda informacja niezakodowana w implementacji jest tracona, dlatego inżynieria wstecz wymaga odpowiednich narzędzi i udziału człowieka.
- UML jest na tyle wyrazisty i jednoznaczny, że umożliwia nie tylko bezpośrednie przekształcanie modeli, ale także symulację systemów oraz dostrajanie elementów wdrożonych systemów.

Język UML - ilustracja



Inżynieria wprzód i wstecz – uwagi

- Inżynieria naprzód polega na automatycznym przekształcaniu modelu do kodu źródłowego.
- Inżynieria wstecz jest procesem odwrotnym, umożliwia odzyskiwanie informacji o szczegółach projektu systemu na podstawie kodu źródłowego.
- Warto stosować inżynierię wprzód gdy zaczynamy kodowanie a inżynierię wstecz gdy projekt jest już gotowy.
- Inżynierię wstecz stosuje się aby np.:
    - zaprezentować model
    - dokonać modyfikacji modelu i kodu (np. na postawie wzorców).

Odwzorowanie modelu na kod

- Określ zasady mapowania na język implementacji lub wybrane języki.
- W zależności od semantyki wybranych języków można ograniczyć korzystanie z niektórych funkcji UML np. wielokrotne dziedziczenie. Można tworzyć idiomy, które przekształcają dodatkowe funkcje na język implementacji.
- Można oznaczać wartości, aby sterować wyborami dotyczącymi implementacji w języku docelowym. Można to robić na poziomie poszczególnych klas albo na wyższym dla przypadku współpracy lub pakietów.
- Następnie należy użyć odpowiednich narzędzi, które umożliwiają generowanie kodu.

Odwzorowanie kodu na model

- Określ zasady mapowania na język implementacji lub wybrane języki.
- Następnie za pomocą narzędzi należy wskazać kod który ma być poddany inżynierii wstecznej. Można stworzyć nowy model albo zmodyfikować istniejący.
- Następnie za pomocą narzędzi należy utworzyć diagram klas wysyłając zapytanie do modelu. Można zacząć od jednej klasy a następnie rozwinąć diagram, postępując według określonych relacji lub innymi sąsiednimi klasami.
- Na koniec należy ręcznie dodać do modelu informacje o projekcie, aby wyrazić zamiar projektu, którego brakuje lub jest ukryty w kodzie.

Widoki modelu – widoki 4+1 Kruchtena

System widoków 4+1 Kruchtena rozbija właściwy zestaw widoków, z których każdy przedstawia konkretny aspekt systemu.

1. **Widok logiczny** – przedstawia abstrakcyjny opis części systemu oraz sposobów, w jaki one ze sobą współdziałają (**diagramy klas, obiektów, maszyny stanowej, interakcji**).
2. **Widok procesu** – opisuje procesy w systemie. Jest on szczególnie przydatny przy wizualizacji wszystkich przypadków, jakie muszą zajść w systemie (**diagramy czynności**).
3. **Widok konstrukcji** – opisuje sposób, w jaki części systemu są zorganizowane w moduły oraz komponenty. Jest on bardzo przydatny do zarządzania warstwami w obrębie architektury systemu (**diagramy pakietów i komponentów**).
4. **Widok fizyczny** – wyjaśnia, w jaki sposób projekt systemu opisany w trzech poprzednich widokach jest powoływany do życia w postaci zestawu rzeczywistych obiektów. Ukazane jest w jaki sposób części abstrakcyjne odwzorowywane są do postaci rzeczywistego wdrożonego systemu (**diagramy wdrożenia**).
5. **Widok przypadków użycia** – widok ten opisuje funkcjonalność modelowanego systemu z perspektywy zewnętrznej. Ten widok jest niezbędny w celu zaprezentowania przeznaczenia systemu (**diagramy przypadków użycia**).

Wszystkie inne widoki bazują na widoku przypadków użycia, dlatego właśnie ten model jest nazywany 4+1.

Diagramy UML

Diagram to widok zawartości modelu. Sposób prezentacji niektórych małych części informacji zawartych w całym modelu.

Diagram w języku UML jest grafem, gdzie najczęściej wierzchołkami są elementy, a krawędziami związki.

Diagram jest pewnego rodzaju rzutem (projekcją) systemu. Pojedynczy diagram daje zatem niepełną informację dotyczącą elementów (bytów) tworzących system.

Ten sam byt może występować we wszystkich diagramach. Poszczególne typy diagramów opisują oddzielne perspektywy (punkty widzenia) systemu (podsystemów).

Rodzaje diagramów

Klasycznie w języku UML wyróżnia się 9 rodzajów diagramów:

1. Diagram przypadków użycia (*use case diagram*).
2. Diagram klas (*class diagram*).
3. Diagram obiektów (*object diagram*).
4. Diagram sekwencji/przebiegu (*sequence diagram*).
5. Diagram kooperacji/kolaboracji (*collaboration diagrams*).
6. Diagram stanów (*statechart diagram*).
7. Diagram czynności (*activity diagram*).
8. Diagram komponentów (*component diagram*).
9. Diagram wdrożenia (implementacji) (*deployment diagram*).

Nowe diagramy

Rodzaje diagramów:

- diagram przypadków użycia,
- diagram czynności,
- diagram klas,
- diagram obiektów,
- diagram sekwencji,
- diagram kooperacji,
- diagram uwarunkowań czasowych,
- diagram przeglądu interakcji,
- diagram struktur złożonych.
- diagram komponentów,
- diagram pakietów,
- diagram maszyny stanowej,
- diagram wdrożenia.

Inna klasyfikacja:

- diagramy strukturalne:
  - diagram klas,
  - diagram obiektów,
  - diagram pakietów,
  - diagram komponentów,
  - diagram wdrożenia,
  - diagram struktur złożonych,
- diagramy dynamiki:
  - diagram przypadków użycia,
  - diagram maszyny stanowej,
  - diagram sekwencji,
  - diagram czynności,
  - diagram kooperacji
  - diagram przeglądu interakcji,
  - diagram uwarunkowań czasowych.

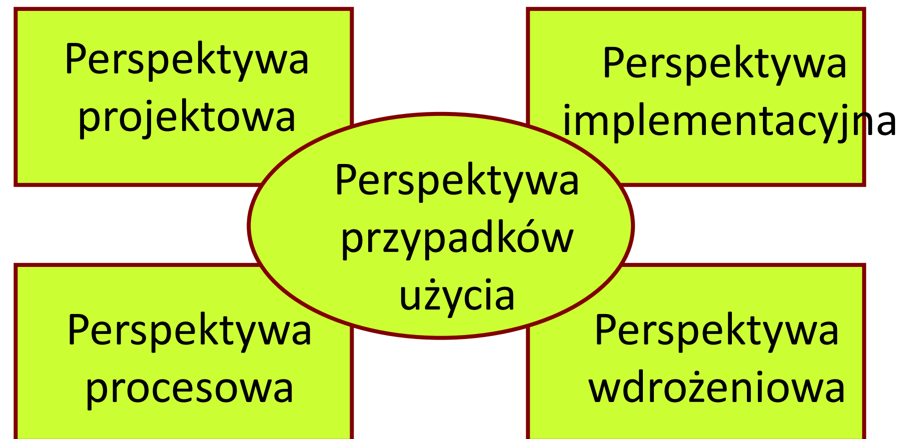
Architektura systemu

Architektura to zbiór znaczących decyzji dotyczących:
• organizacji systemu komputerowego,
• wyboru elementów strukturalnych i ich interfejsów, z których system jest zbudowany,
• zachowania tych elementów, opisanego w kooperacjach,
• składania elementów strukturalnych i czynnościowych w coraz większe podsystemy,
• stylu architektonicznego, według którego tworzy się konstrukcję systemu,
to znaczy charakterystycznych elementów statycznych i dynamicznych oraz ich interfejsów, kooperacji i składania.

Modelowanie architektury systemu

Słownictwo
Funkcjonalność

Scalanie systemu
Zarządzanie konfiguracją



Efektywność
Skalowalność

Topologia systemu
Rozmieszczenie
Dostarczenie
Instalacja

Perspektywy modelowania

Perspektywa przypadków użycia – zachowanie systemu widziane oczyma użytkowników, analityków i osób wykonujących testy. Nie definiuje się rzeczywistej organizacji systemu, lecz opisuje się czynniki wpływające na kształt architektury systemu.

Aspekty statyczne – diagramy przypadków użycia.
Aspekty dynamiczne – diagramy interakcji, diagramy stanów, diagramy czynności.

Perspektywa projektowa – nacisk kładzie się na klasy, interfejsy i kooperacje, które składają się na słownictwo i rozwiązanie danego zadania. Perspektywa ta wspiera specyfikację funkcjonalną – usług udostępnianych użytkownikom.

Aspekty statyczne – diagramy klas i diagramy obiektów.
Aspekty dynamiczne – diagramy interakcji, diagramy stanów i diagramy czynności.

Perspektywy modelowania (cd)

Perspektywa procesowa – zwraca się uwagę na wątki i procesy kształtujące mechanizmy współbieżności w systemie.
Dotyczy to głównie efektywności, skalowalności i przepustowości systemu.

Aspekty statyczne i *dynamiczne* wyrażane są diagramami tego samego rodzaju jak w przypadku perspektywy projektowej, przy czym główny nacisk kładzie się na klasy aktywne reprezentujące procesy i wątki.

Perspektywa implementacyjna - znaczenie mają komponenty i pliki użyte do scalania i udostępniania systemu fizycznego.
Wiąże się z zarządzaniem konfiguracjami poszczególnych wersji systemu, złożonych z rozmaitych komponentów i plików, które można scalić na wiele sposobów, aby otrzymać działający system.

Aspekty statyczne wyraża się tu przez diagramy komponentów, a *dynamiczne* za pomocą diagramów interakcji, diagramów stanów i diagramów czynności.

Perspektywy modelowania (dok)

Perspektywa wdrożeniowa kładzie nacisk na węzły, specyfikujące sprzęt, na którym system będzie uruchamiany.
Wiąże się głównie z rozmieszczeniem, dostarczeniem i instalacją części systemu fizycznego.

Aspekty statyczne wyraża się tu za pomocą diagramów wdrożenia, *dynamiczne* za pomocą diagramów interakcji, diagramów stanów i diagramów czynności.

Architektura semantyczna

• Na najwyższym poziomie abstrakcji odróżnia się trzy warstwy kompozytowe.
• Warstwą podstawową jest warstwa strukturalna. Odzwierciedla to założenie, że każde zachowanie jest konsekwencją działania jednostek strukturalnych.
• Środkowa behawioralna warstwa stanowi podstawę dla semantycznych opisów formalizmów behawioralnych wyższego poziomu.
Podzielona jest na trzy niezależne obszary:

1. *inter-object behavior base* (komunikacja pomiędzy podmiotami strukturalnymi)
2. *intra-object behavior base* (komunikacja wewnątrz)
3. *actions* – definiują semantykę poszczególnych działań. Służą do definiowania drobiazgowych zachowań. Są one dostępne dla formalizmów wyższego poziomu.

• Najwyższa warstwa w hierarchii semantycznej definiuje semantykę wyższego poziomu formalizmów behawioralnych UML.

Architektura semantyczna – ilustracja