Politechnika Warszawska

Wprowadzenie do architektury oprogramowania

dr hab. inż. Michał Śmiałek, prof. uczelni
dr inż. Kamil Rybiński

Wydział Elektryczny
POLITECHNIKA WARSZAWSKA

Inżynieria oprogramowania

1

Czym jest architektura oprogramowania?

- Problem: złożoność systemów oprogramowania
  - Musimy zapanować nad złożonością
- Problem: brak powszechnie przyjętej definicji
  - Patrz: Software Eng. Institute (CMU) – What is your definitione of software architecture (ponad 30 definicji)
- Problem: systemy często tworzone BEZ architektury (kodujmy czym prędzej…)
  - Budowa wieżowca bez planów architektonicznych (kopmy fundamenty czym prędzej…)
  - Brak dokumentacji: „Nasz kod sam się dokumentuje”
- Problem: brak wspólnego języka
  - Nieformalne rysunki typu „pudełka i kreski”

2

Architektura oprogramowania: definicja

- W skrócie: architektura oprogramowania jest ogólnym wizualnym modelem systemu oprogramowania zapewniającym spełnienie zadanych wymagań.
- Bass, Clement, Kazman (A. o. w praktyce)

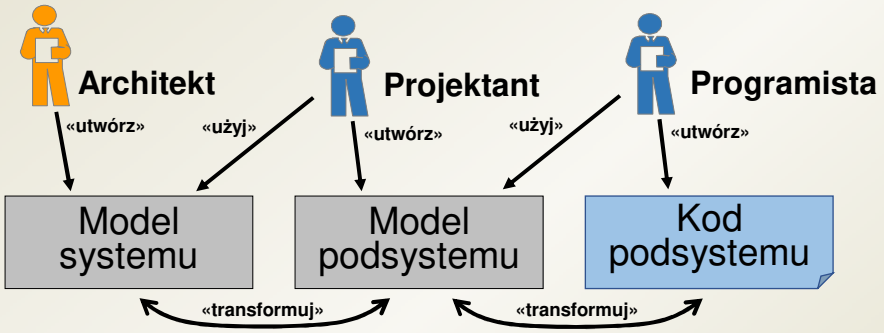
  „Architektura oprogramowania (…) jest strukturą lub strukturami systemu, który zawiera elementy programowe, cechy tych elementów widoczne na zewnątrz i relacje między nimi.”
- Martin (Czysta architektura)

  „(…) polega na zachowaniu jak największej liczby otwartych możliwości przez jak najdłuższy czas”

Architektura oprogramowania: wymagane cechy

- Plany architektoniczne dla oprogramowania:
  - Wyrażony zestaw decyzji podjętych przez architekta.
  - Decyzje podjęte na bazie wymagań klienta i wiedzy na temat technologii wytwarzania oprogramowania
  - Pokazują strukturę i dynamikę (działanie) systemu, który ma zostać (lub został) zbudowany.
  - Biorą pod uwagę ograniczenia ekonomiczne i technologiczne.
  - Biorą pod uwagę konieczność (łatwość możliwości) wprowadzania zmian i rozszerzeń wynikłych ze zmian wymagań.
  - Wyrażone w języku graficznym (wizualnym) zrozumiałym dla innych projektantów i wykonawców systemu.

Role i produkty



- Architekt tworzy ogólny model systemu
  - Z modelu systemu korzysta projektant
  - Programista koduje system na podstawie projektu

5

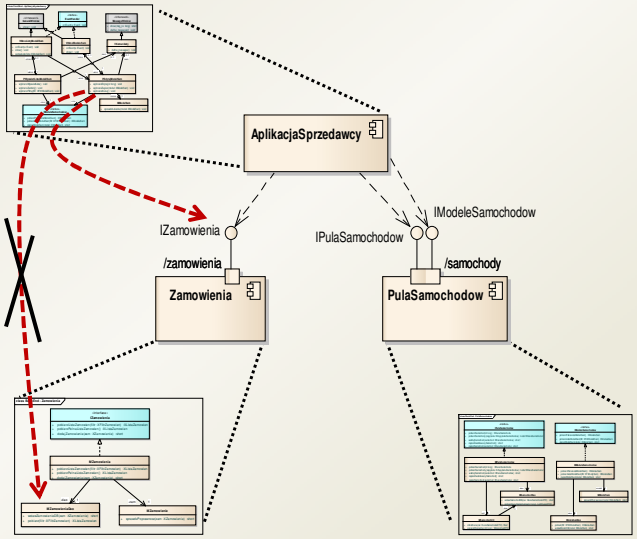
Architektura komponentowa: definicja

- Co to jest komponent?
  - Modularna, zamknięta część systemu (czarna skrzynka)
  - Specyfikuje swoje zachowanie poprzez interfejsy dostarczane i wymagane
  - Wymienny na inne komponenty o tej samej specyfikacji
- Co to jest architektura komponentowa?
  - Architektura oparta na niezależnych, wymiennych komponentach
  - Wyraźne określnie odpowiedzialności komponentów
  - Komponenty zwarte wewnętrznie i luźno powiązane
- Wszystkie architektury oparte o usługi można uznać za architektury komponentowe

6

Jakie cechy powinny mieć komponenty?

- Bardzo spójne wewnętrznie i luźno powiązane z innymi
- Dobre abstrakcje jako „kawałki funkcjonalności systemu”
- Komunikują się poprzez wąskie interfejsy

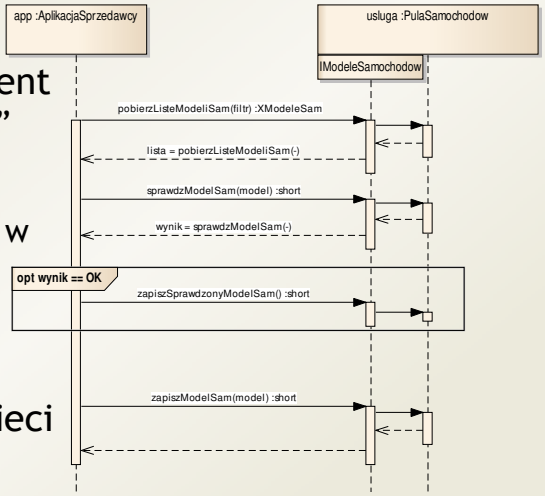


Dlaczego architektury komponentowe?

- Ponieważ:
  - Łatwiej zrozumieć system stosując dobre abstrakcje (komponenty); błyskawiczny ogląd sytuacji; jeśli potrzeba – wejście w szczegóły projektu.
  - Pozwala podzielić precyzyjnie pracę; grupy koncentrują się na komponentach realizujących interfejsy.
  - Duża elastyczność zmian systemu; łatwo dodać nowe komponenty i przenieść komponenty między maszynami.
  - Zapobiega spaghetti w kodzie.
  - Ułatwia znalezienie błędów; jako czarne skrzynki komponenty są bardziej odporne na błędy w innych komponentach; komponenty są „centrami detekcji błędów”.
  - Pozwalają lepiej wykorzystywać kod wielokrotnie; dobrze napisane komponenty znakomicie nadają się do ponownego wykorzystania (klocki „Lego”)

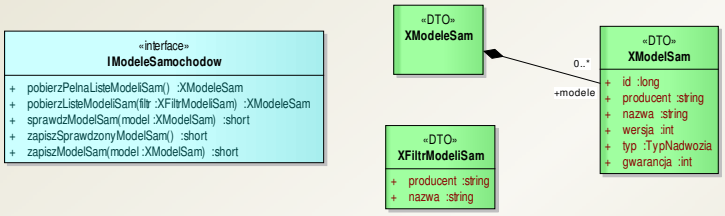
Usługi

- Usługa: dobrze zdefiniowany i reużywalny komponent
  - Dostarcza „kontrakt” dla innych komponentów
  - Może być stosowana w różnych systemach
  - Może pracować w systemach heterogenicznych
- Usługi dostępne w sieci to usługi webowe



9

Obiekty transferu danych (DTO)



- Spójny pakiet danych przesyłanych między komponentami (usługami)
  - Definiuje parametry operacji interfejsów dla usług
  - W języku UML DTO modelujemy jako klasy (nadajemy im odpowiedni stereotyp)
  - Konwencje nazewnicze: np. przedrostek „X” lub przyrostek „DTO”

10

Architektury zorientowane na usługi (SOA)

- Cechy Service Oriented Architecture
  - Standaryzacja kontraktów między usługami
  - Luźne powiązania między usługami (minimalizacja zależności)
  - Stosowanie abstrakcji (ukrywanie implementacji)
  - Reużywalność (ponowne wykorzystanie komponentów)
  - Możliwość odkrywania (wyszukiwanie wg. metadanych)
  - Możliwość komponowania systemów z usług (tzw. orkiestracja usług)
- Architektura mikro-usługowa
  - Usługi dostarczające niewielkich, silnie skupionych zestawów operacji

11

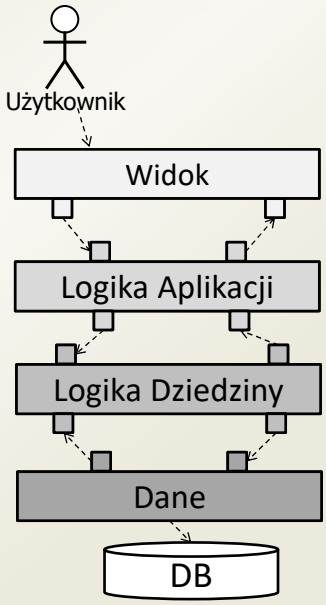
Realizacja interfejsów

- Interfejsy programowania aplikacji (API)
  - Interfejs webowy komponentu (usługi) zrealizowany w standardowej technologii
- Technologia REST
  - Korzysta ze standardowego protokołu HTTP
  - Podstawowe zasady: rozdzielenie obowiązków, bezstanowość usług, warstwowość, jednolitość
  - Interfejsy (API) udostępniane pod odpowiednim adresem URI
  - Adres składa się z adresu bazowego oraz punktu końcowego (endpoint)
  - Dane przesyłane w sposób tekstowy (JSON, XML)
- Technologie RPC
  - Zdalne wywołania procedur (np. gRPC)
  - Naśladowanie lokalnego wywoływania procedur
  - Dane przesyłane w plikach binarnych (oszczędność transferu)

12

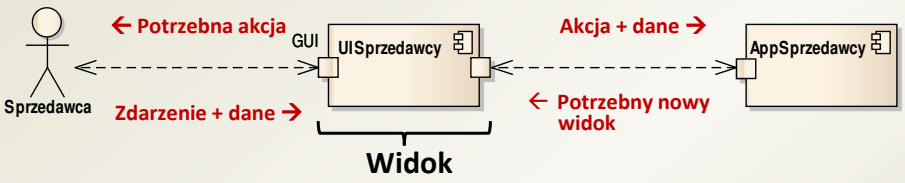
Architektura warstwowa

- Warstwy: „odległość” od aktorów (obiektów poza systemem)
- Architektura klient-serwer
  - Górna warstwa prezentuje dane (frontend)
  - Dolna warstwa przetwarza dane (backend)
- Architektura wielowarstwowa
  - Każda warstwa komunikuje się tylko z warstwą niżej i wyżej)
  - Górne warstwy są szybko zmienne (kontrolki, formularze,…)
  - Warstwy środkowe zmieniają się wraz ze zmianą wymagań funkcjonalnych i dziedziny problemu
  - Dolne warstwy zmieniają się w miarę zmian technologii (SZBD)



13

Warstwa widoku (skł. frontendu)



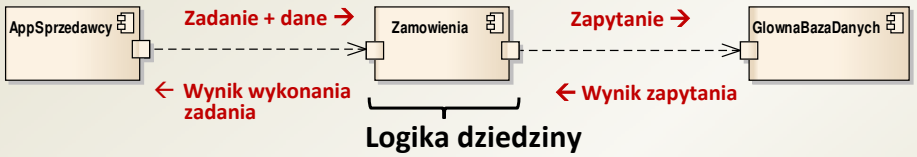
- Prezentuje wyniki (dane) i otrzymuje dane (polecenia) od użytkowników
  - Zawiera kod (np. klasy) dla poszczególnych ekranów; generuje widok okien i odbiera interakcje z oknami (przyciski itp.)
  - Odpowiada tylko za prezentowanie (wyświetlanie) i przyjmowanie danych do użytkownika – nic więcej!
  - Dostosowana do konkretnej technologii
- Częsty błąd: aktywna prezentacja
  - Nie umieszczamy logiki aplikacji/biznesu w klasach ekranowych
- Częsty błąd: sterowanie nawigacją
  - Pokazuje ekrany i czeka na dane tylko jeśli poproszona przez LA!

14

Warstwa logiki aplikacji (skł. frontendu)

- Steruje innymi warstwami
  - Decyduje o sekwencji akcji wykonywanych przez inne warstwy
  - Realizuje scenariusze wymagań funkcjonalnych
  - Decyduje o nawigacji UI (ekrany)
- Wie jak zareagować na akcje użytkownika
  - Reakcja w zależności od decyzji (danych) od użytkownika
  - Reakcja w zależności od stanu (danych) systemu
- Zawiera kod instruujący „wszystkich” co mają robić („kierownik”)

Warstwa logiki dziedzinowej (skł. backendu)

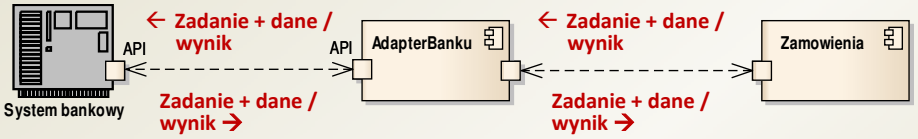


- Wykonuje przetwarzanie danych na życzenie LA
  - Akcje wykonywane zgodnie z logiką biznesu zawartą w słowniku (reguły biznesowe)
  - Uruchamia zapis/odczyt trwałych danych
- NIE realizuje nawigacji między ekranami itp.
- NIE obsługuje bazy danych (SQL)
- Integracja z innymi systemami
  - Udostępnianie/korzystanie z usług zewnętrznych
- Zawiera podstawowy kod „wykonawczy” systemu („robotnik”)

Warstwa przechowywania danych („brudna”)

- Zapewnia trwałe przechowywanie danych przetwarzanych przez LB
  - Dostosowana do konkretnej technologii: relacyjna baza danych (konkretny typ), system plików, repozytorium (np. grafowe), baza wiedzy (np. ontologia), itp.
- Wykonuje operacje typu CRUD
  - Create/ Read / Update / Delete
- Tłumaczy operacje CRUD na zapytania odpowiednie dla technologii
  - Różne języki zapytań: SQL (wiele dialektów), OBQL, OCL, serializacja XML, …
- NIE zalecane: integracja z innymi systemami poprzez warstwę PD
  - Częsta zmiana struktury (tabele, relacje) / technologii

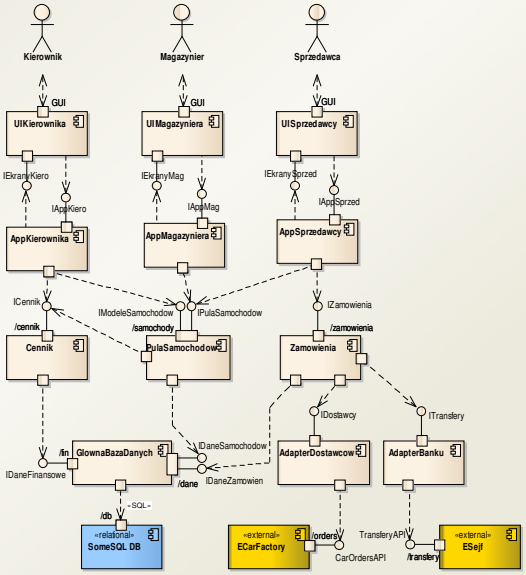
Integracja z innymi systemami



- Najkorzystniej wykonać w warstwie logiki dziedzinowej
- Projektujemy komponenty adaptujące
  - Dostosowanie formatu danych oraz kontraktu do wymagań systemu zewnętrznego

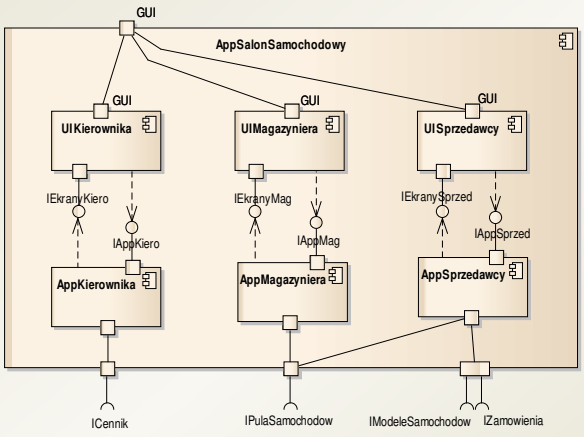
Przykład architektury warstwowej

- Aplikacje dla kilku aktorów (frontend UI+App)
- Cztery API dostępne przez 3 punkty końcowe
- Komponenty bazy danych
- Komponenty integracyjne



19

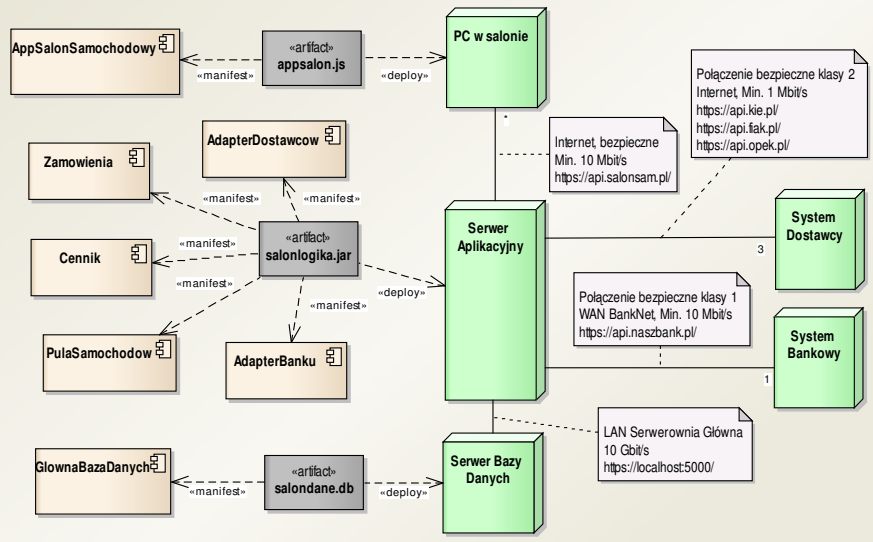
Komponenty złożone



- Wyraźnie określony zbiorczy komponent frontendu zawierający kilka komponentów składowych

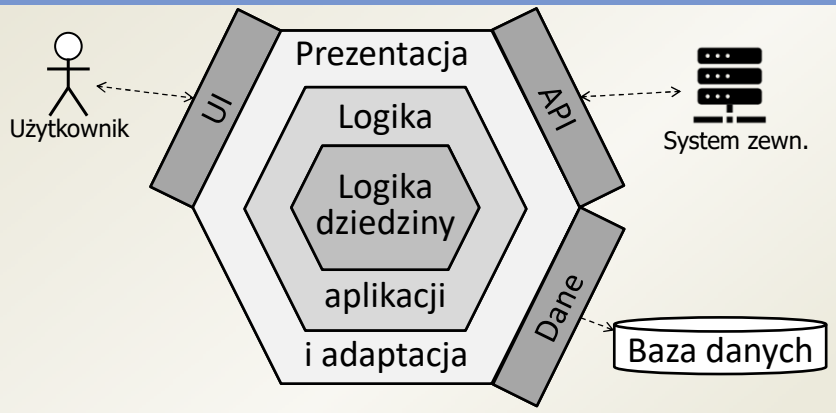
20

Przykładowa architektura fizyczna



21

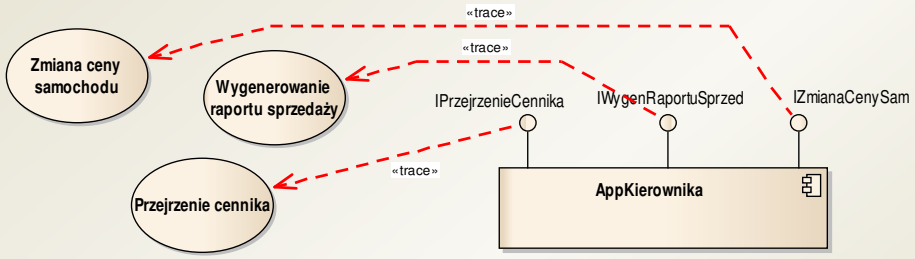
Architektura heksagonalna



- Warstwy ułożone „na cebulę”
  - Warstwa zewnętrzna umożliwia komunikację z użytkownikami, systemami zewnętrznymi i bazami danych

22

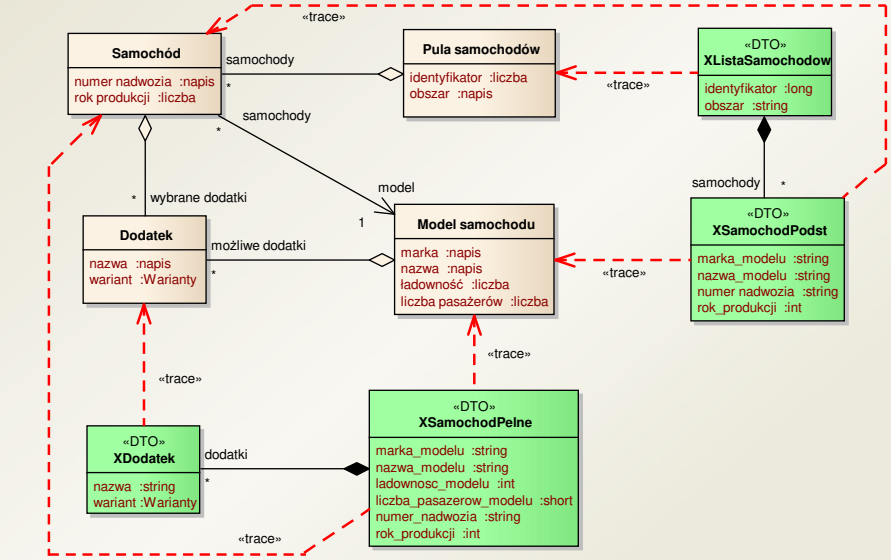
Projektowanie komponentów na podstawie wymagań



- Przykładowa reguła: dla każdego przypadku użycia stwórz interfejs warstwy logiki aplikacji

23

Projektowanie obiektów transferu danych na podstawie modelu dziedziny



24

Projektowanie dynamiki na podstawie scenariuszy