Inżynieria oprogramowania II

Wykład 6:

"Architektura oprogramowania"

Marek Krętowski
e-mail: mkret@wi.pb.edu.pl
http://aragorn.pb.bialystok.pl/~mkret

Dlaczego potrzebujemy architektury?

- **Zrozumienie systemu** - systemy oprogramowania są duże, skomplikowane i muszą spełniać sprzeczne wymagania; architektura zapewnia szkielet rozwiązania, który abstrahuje od szczegółów implementacyjnych, ale ustawia wzajemnie podstawowe elementy tak aby możliwe byłoby spełnienie wymagań
- **Organizowanie rozwoju oprogramowania** - pozwala odseparować różne części, tak aby ich tworzenie było możliwie niezależne; zależności pomiędzy częściami są jasno określone i punkty styku dokładnie wyspecyfikowane
- **Promowanie ponownego wykorzystania** - architektura pomaga w zidentyfikowaniu na możliwie wysokim poziomie analogicznych systemów krytycznych i podsystemów, dzięki czemu ponowne wykorzystanie nie ogranicza się do poziomu klas; wspólne podsystemy mogą być przystosowane na etapie rozwoju do późniejszego wyk.
- **Promowanie ciągłego rozwoju** - większość systemów musi reagować na nowe potrzeby i wymagania przed nimi stawiane; dobra architektura pozwala zapanować nad ewolucją systemu w czasie i kontrolować jej przebieg; zapoznanie się z architekturą pozwala na zrozumienie działania systemu przez przyszłych twórców i eliminuje potencjalne zagrożenia

Tradycyjne spojrzenie na architekturę

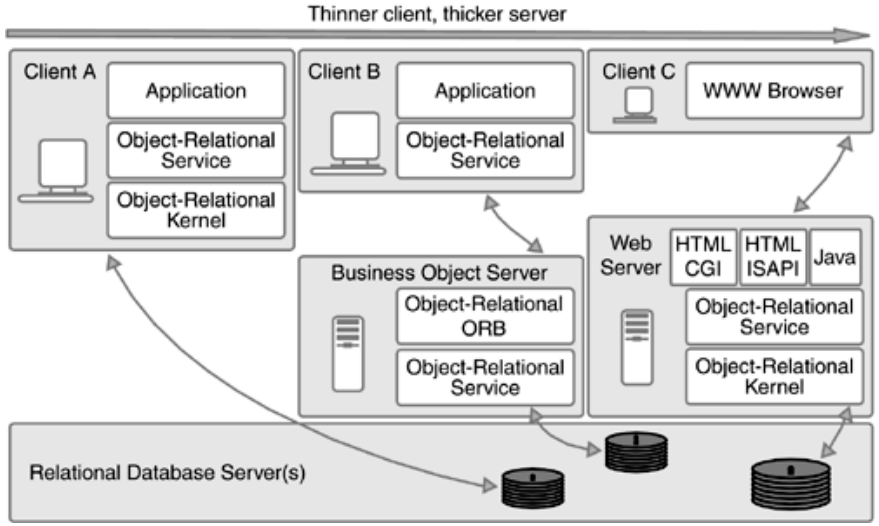
Proces projektowania architektonicznego zależy od wiedzy o zastosowaniu oraz umiejętności i intuicji architekta; wspólne czynności dla wszystkich procesów (czynności te zwykle nakładają się na siebie):

- **strukturalizacja systemu** - system jest dzielony na kilka podstawowych podsystemów (podsystem - niezależna jednostka oprogramowania, jej usługi nie zależą od usług oferowanych przez inne podsystemy); identyfikuje się komunikację między podsystemami
  - podział podsystemów na moduły (moduł jest zwykle komponentem systemu, który oferuje co najmniej jedną usługę innym modułom; zwykle nie jest niezależny)
- **modelowanie sterowania** - określa się ogólny model związków sterowania między częściami systemu; uzupełnia użyty model struktury

Standardowe modele struktury

- **Repozytorium** - wszystkie współdzielone dane są umieszczone w centralnej bazie danych, z której mogą korzystać wszystkie podsystemy; zwykle generowane przez jeden podsystem a wykorzystywane przez pozostałe
  - systemy sterowania i kontroli, systemy zarządzania informacjami, systemy CAD, zestawy narzędzi CASE
- **Klient-serwer** - model rozproszonego systemu, w którym dane i przetwarzanie są rozdzielone między zbiór procesorów (samodzielne serwery oferujące usługi + klienci korzystający z usług oferowanych przez serwery + sieć, dająca dostęp klientom do usług)
  - **cienki klient** - całość przetwarzania i zarządzania danymi na serwerze, zadaniem klienta jest uruchomienie oprogramowania prezentacyjnego
  - **gruby klient** - serwer odpowiada za zarządzanie danymi; oprogramowanie klienta implementuje logikę programu użytkowego i interfejs użytkownika
- maszyna abstrakcyjna (**warstwowy**) - opisuje sprzęganie podsystemów; układa system w ciąg warstw, z których każda oferuje pewne usługi

Architektura klient-serwer



Modele sterowania

Sterowanie **scentralizowane** - jeden z podsystemów jest wybrany do roli sterownika i odpowiada za zarządzanie działaniem innych

- **model wywołanie-powrót** - sterowanie zaczyna się na wierzchołku hierarchii podprogramów i przez wywołania podprogramów przechodzi do niższych poziomów; jedynie systemy sekwencyjne
- **model menedżera** - stosuje się do systemów współbieżnych; jeden z komponentów systemu jest menedżerem i steruje rozpoczynaniem, zatrzymywaniem i koordynacją innych procesów systemu

Sterowanie **zdarzeniowe** - każdy podsystem musi reagować na zdarzenia zachodzące na zewnątrz

- **model rozgłaszania** - zdarzenie jest w zasadzie ogłoszeniem dla wszystkich podsystemów, każdy podsystem, który może obsłużyć to zdarzenie, reaguje na nie
- **modele z przerwaniami** - używane wyłącznie w systemach czasu rzeczywistego, gdzie zewnętrzne przerwania są wykrywane przez obsługę przerwań; następnie są one przekazywane do innego komponentu, który je przetworzy

W Unified Process architektura oprogramowania powinna obejmować:

- Ogólny plan struktury systemu - powinien umożliwiać zrozumienie struktury systemu, bez wchodzenia w szczegóły rozwiązania
- Kluczowe elementy strukturalne i ich interfejsy - z jakich elementów zbudowany system i w jaki sposób te elementy są połączone
- W jaki sposób elementy te współpracują (na najwyższym poziomie) - kiedy elementy wzajemnie na siebie oddziałują (co robią i dlaczego?)
- W jaki sposób elementy tworzą systemy i podsystemy - bardzo istotny element architektury; możliwie wczesne rozpoznanie kluczowych systemów i podsystemów pozwala organizować dalsze prace (projektowanie i implementację), wspiera dobre zrozumienie systemu i promuje ponowne wykorzystanie
- Styl architektoniczny

W ramach określonych w architekturze “przestrzeni do wypełniania” projektanci mają swobodę pracy

Dodatkowe elementy

Architektura oprogramowania może ponadto obejmować:

- W jaki sposób system będzie wykorzystywany
- Spodziewaną ostateczną funkcjonalność systemu
- Zagadnienia dotyczące osiągów aplikacji (ang. performance), które muszą być uwzględnione (może to wymagać bardziej szczegółowego opracowania sposobu implementacji architektury w celu spełnienia ograniczeń)
- Przygotowanie na przyszły rozwój
- Ekonomiczne i technologiczne ograniczenia oraz przyjęte kompromisy (architektura może rozważać różne rozwiązania problemu, umożliwiać przedstawienie różnych rozwiązań technologicznych i wybranie najodpowiedniejszego)

Architektura przypomina stację kosmiczną, w której rdzeń wyposażony jest we wszystkie kluczowe przewody i podłączenia umożliwiając dołączenie przyszłych modułów

Charakterystyka dobrej architektury

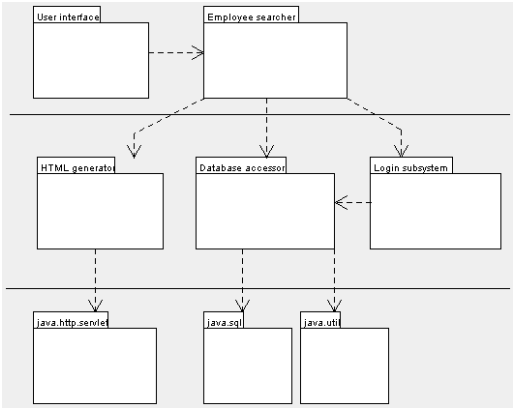
Łatwiej jest zdefiniować jakimi cechami powinna charakteryzować się architektura niż w rzeczywiści stworzyć dobrą architekturę

W wielu przypadkach nie jest możliwe spełnienie wszystkich wymienionych poniżej cech tym niemniej należy do tego dążyć:

- **"gotowość na zmiany"** (ang. resilience) - zmiany funkcjonalności lub dodatkowe wymagania powinny mieć minimalny wpływ na architekturę => jasno zdefiniowana struktura z wyspecyfikowanymi interfejsami
- **prostota** - całościowe objęcie systemu => unikanie skomplikowanych rozwiązań
  - rozmiar architektury nie powinien przekraczać 10% projektu
- **przejrzystość prezentacji** - ponieważ jest to jeden z najistotniejszych elementów systemu, wykorzystywany zarówno w podczas projektowania jak i potencjalnych przyszłych iteracji, powinien być łatwo dostępny i unikać wszelkich dwuznaczności
  - nie powinien zakładać bieżącej znajomości projektu
- jasne **oddzielenie różnych aspektów** systemu
- zrównoważone **rozdzielenie zobowiązań**
- uwzględnia **ekonomiczne i technologiczne ograniczenia** - architektura może musieć uzasadniać dlaczego wybrano takie a nie inne rozwiązanie (np. wyjaśnienie ogólnych wyborów zespołowi); może mieć wpływ na elementy projektu (ograniczenia)

Architektura warstwowa

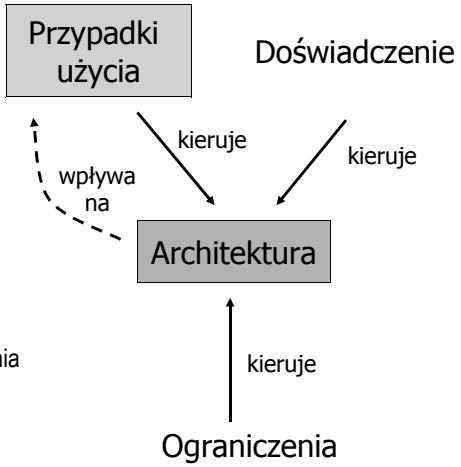
- Oparcie architektury o wyróżnione warstwy może być użyteczne w celu jej uproszczenia, aby ułatwić zrozumienie i rozwój złożonych systemów
- Zapewnia, że podsystemy znajdujące się na różnych warstwach są ze sobą jedynie luźno związane
- Podsystemy na na niższych poziomach nie muszą być świadome istnienia podsystemów na wyższych poziomach, które z nich korzystają
  - podsystemy mogą mieć dostęp jedynie do podsystemów z tej samej lub niższej warstwy



Przykład architektury z 3 warstwami: prosta aplikacja internetowa (ang. Web-based) wyszukująca pracowników

Architektura a przypadki użycia

- Istnieje związek pomiędzy funkcjonalnością systemu (reprezentowaną przez przypadki użycia) a architekturą oprogramowania



Opracowywanie architektury

Podstawowe kroki związane z tworzeniem architektury (kolejność może być zmieniana w pewnym zakresie i zwykle potrzebne są iteracje):

- Wyszukiwanie znaczących architektonicznie przypadków użycia
- Identyfikowanie systemów i podsystemów
- Rozpoznawanie klas architektury
  - jako część procesu zidentyfikowanie struktury klas i ich związków z podsystemami
- Sprawdzenie możliwości i potrzeb współbieżności oraz rozproszenia
- Zarządzanie magazynami danych systemu
- Rozpatrywanie dodatkowych aspektów architektonicznych
- Implementacja co najmniej jednego prototypu architektonicznego
- Przetestowanie na podstawie przypadków użycia
- Ocena architektury