WI

Programowanie komponentowe

Podstawy

Piotr Błaszyński

Wydział Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego

8 marca 2016

Jak było - Paradygmat obiektowy

- Abstrakcja
- Enkapsulacja (Hermetyzacja)
- Polimorfizm
- Dziedziczenie

Ważnym elementem zrozumienia programowania obiektowego jest akceptacja faktu istnienia różnych technik (wbrew często występującej postawie: korzystam z X bo Y się nie nadaje do jednego z zastosowań (lub przeciwnie)).

Jak było źle - typowe użycie obiektowości

- ścisłe powiązania,
- wspólna funkcjonalność w klasach bazowych,
- różnice pomiędzy klasami implementowane przy użyciu polimorfizmu.

Jak było źle - Problemy

- skomplikowane projekty,
- za duże klasy bazowe,
- skomplikowane klasy bazowe,
- wielodziedziczenie,
- mnogość interfejsów,
- duplikacja kodu,
- wysoka specjalizacja — problem z ponownym wykorzystaniem,
- drobnoziarnistość,
- ścisłe powiązanie z programem (konieczność kompilacji i łączenia w jednym miejscu).

Komponenty jako ewolucja OOP

- predefiniowane usługi,
- zdalna komunikacja,
- transakcyjność,
- bezpieczeństwo,
- trwałość danych,
- samotestowanie,
- zazwyczaj w postaci skompilowanej,
- pełna hermetyzacja,
- niezaleznosc od jezyka aplikacji,
- mozliwosc powtórnego uzycia,
- gruboziarnistosc.

Zalety komponentów

- separacja usług systemowych od funkcjonalnosci,
- redukcja złożonosci budowy aplikacji,
- mniejsze koszty budowy i utrzymania,
- budowa z „klocków”.

Komponent - definicja

- jest podstawową jednostką oprogramowania z interfejsami opisanymi w sposób deklaratywny,
- często również z podanymi wprost zaleznościami,
- może być integrowany w całości albo wcale,
- moze byc skonfigurowany i wdrożony niezależnie od programisty, który go napisał.

Komponenty - wymagania

- możliwość ponownego użycia (reusing),
- bezobsługowość,
- specyfikacja zależności,
- wyspecyfikowana funkcjonalność,
- użycie wyłącznie na podstawie specyfikacji,
- kompatybilność z innymi komponentami,
- łatwa integracja z różnymi systemami,

Komponenty - właściwości

- brak samokontroli,
- Komponenty nie są uruchamiane samodzielnie,
- komponentami zarządza kontener (w naszym przypadku aplikacja host),
  - tworzy je,
  - rozwiązuje zależności,
  - czuwa nad ich cyklem zycia.

Komponenty - zalety

- zwiekszona niezawodnosc,
- zmniejszone zagrozenie procesu,
- efektywne wykorzystanie specjalistów,
- zgodnosc ze standardami,
- przyspieszone tworzenie.

Komponenty - wady

- zwiększone koszty pielegnacji (brak zródeł),
- brak lub słabe wspomaganie narzedziowe,
- przepisywanie (syndrom NIH),
- konieczność prowadzenie biblioteki komponentów,
- trudności w znajdowaniu i przystosowywaniu komponentów.

Podsumowanie części teoretycznej.

- Komponenty to dalszy, naturalny etap rozwoju OOP.
- Komponenty są gotowym i w miarę tanim w użyciu prefabrykatem.
- Dają zwiększenie niezawodności i skalowalności systemów.
- Wciąż rozwijający sie rynek.

.NET - użycie komponentów przez refleksję

- Assembly - w dużym uproszczeniu - odpowiednik jednego pliku .dll - komponent,
- Możliwość pobrania informacji (o klasach i typach w Assembly) na podstawie pliku .dll,
- Assembly assemblyInfo = Assembly.LoadFrom(@"plik.dll"); ,
- assemblyInfo.GetTypes() ,
- .GetMethods() ,
- .Name ,
- .Invoke() - wywołanie,
- możliwość weryfikacji również listy parametrów.