

Program szkolenia:

Wzorce projektowe i efektywne techniki Object Oriented Design dla projektantów systemów

Informacje ogólne

Nazwa:	Wzorce projektowe i efektywne techniki Object Oriented Design dla projektantów systemów
Kod:	Patterns Sys
Kategoria:	Inżynieria oprogramowania
Grupa docelowa:	Projektanci i architekci
Czas trwania:	3-4 dni
Forma:	50% wykłady / 50% warsztaty

Szkolenie prezentuje wybrane Wzorce Projektowe w praktycznym i niepodręcznikowym ujęciu osadzonym w kontekście projektowania bibliotek, frameworków, platform i systemów. Podczas szkolenia prezentowane są przykłady praktycznego zastosowania zaczerpnięte z rzeczywistych systemów klas: ERP, narzędzia wizualne, systemy rozproszone, serwery.

Podczas szkolenia uczestnicy nabędą zintegrowaną wiedzę na temat zdobyczy nowoczesnej inżynierii oprogramowania pozwalającą im na tworzenie zaawansowanych systemów. Podczas warsztatów praktycznych łączymy wzorce projektowe i architektoniczne aby stworzyć giętkie i otwarte na rozbudowę rozwiązania cechujące się wysokim poziomem testowalności. Omawiane zagadnienia leżą u podstaw nowoczesnych frameworków i technologii – co zwiększa poziom ich zrozumienia i pozwala na świadome korzystanie. Przedstawiamy techniki łączenie wzorców w struktury wyższego rzędu.

Szkolenie przeznaczone dla projektantów i architektów pragnących poszerzyć swe kompetencje w zakresie profesjonalnych technik inżynierii oprogramowania zwiększających jakość kodu i projektu.

Zalety szkolenia:

- » Skupienie na kontekście projektowania aplikacji i systemów
- » Wybór jedynie użytecznych wzorców oraz technik
- » Realne przykłady

Program szkolenia:

1. Techniki Object Oriented Design

- 1.1. Analiza Paradygmatu Object oriented i jego poprawna interpretacja
- 1.2. Ukierunkowanie myślenia w stylu OO
- 1.3. Najlepsze praktyki i pułapki
- 1.4. GRASP - General Responsibility Assignment Software Patterns
- 1.5. SOLID - Single Responsibility Principle (SRP), the Open/Closed Principle (OCP), the Liskov Substitution Principle (LSP), the Dependency Inversion Principle (DIP), and the Interface Segregation Principle (ISP)

2. Antywzorce i typowe pułapki

- 2.1. Code smell – wykrywanie ok 20 zapachów kodu
- 2.2. Przegląd typowych błędów i pułapek
- 2.3. Techniki refaktoryzacji

3. Wzorce projektowe - praktyczne, nieksiążkowe przykłady oparte o rzeczywiste problemy w kontekście aplikacji Enterprise

- 3.1. Command – hermetyzacja usług, autoryzacja dostępu
- 3.2. Decorator – reużywalność logiki
 - 3.2.1. Składanie złożonej logiki biznesowej o przyrostowym charakterze
 - 3.2.2. Wrapper – odmiana wzorca użyteczna w modelowaniu zorientowanym na znaczenie
 - 3.2.2.1. Opakowanie typów podstawowych wygodnymi obiektami
 - 3.2.2.2. Alternatywa dla Utils
 - 3.2.2.3. Archetyp Money
 - 3.2.3. Połączenie Dekoratora ze Strategią w celu zbudowania odmian algorytmów przyrostowych
- 3.3. Strategy – hermetyzacja logiki biznesowej
 - 3.3.1. Wybór odmiany algorytmu bez ingerencji w core biznesowy

3.3.2. Integracja z mechanizmami wstrzykiwania zależności

3.3.3. Definiowanie konkretnej strategii biznesowej w kontenerze Injection of Control (XML lub metody fabrykujące)

3.4. Chain of Responsibility – dwie odmiany wzorca

3.4.1. Dobór logiki biznesowej do aktualnych warunków

3.4.2. Połączenie łańcucha ze Strategią w celu zbudowania odmian algorytmów warunkowych

3.5. Abstract Factory – tworzenie artefaktów domenowych

3.5.1. Spójny sposób na tworzenie rodzin obiektów biznesowych zależnych od konfiguracji wdrożeniowej systemu

3.5.2. Produkowanie Strategii

3.6. Builder – redukcja złożoności tworzenia struktur

3.6.1. Zunifikowane eksportowanie obiektów domenowych

3.6.2. Ukrywanie złożoności budowania zapytań

3.7. Template Method – antywzorzec w przykładach

3.7.1. Technika u Wspólniania logiki biznesowej

3.8. Przykłady antywzorca

3.9. Singleton – niebezpieczny wzorzec w przykładach

3.9.1. Szczegóły implementacji Singletonów tworzonych z opóźnieniem, odpornych na współbieżny dostęp

3.10. State – hermetyzacja procesu biznesowego

3.10.1. Implementacja maszyny stanów reprezentującej złożony cykl życia obiektu biznesowego

3.10.2. Maszyna Stanów jako Wrapper dodający nowe funkcjonalności

3.11. Observer – redukcja zależności, wysokowydajne systemy Event Driven i asynchroniczne przetwarzanie

3.12. Specification – hermetyzacja reguł biznesowych

3.12.1. Redukcja złożoności systemów zawierających złożoną logikę decyzyjną

3.12.2. Przypadek gdy istnieje wiele możliwych kryteriów logicznych

3.12.3. Jednak w danym kontekście (wdrożenie, klient) używanym tylko podzbiorem reguł

3.13. Facade – redukcja złożonej struktury pod wygodnym API

4. Użyteczne wzorce o zastosowaniu technicznym

4.1. Bridge – rozdzielenie interfejsu koncepcyjnego od jego implementacji

4.2. Flyweight, Prototype – wzorce optymalizacji

4.3. Memento – komunikacja pomiędzy kontekstami

4.4. Proxy – podstawowy wzorzec frameworków

4.5. Composite – powtarzalne struktury

4.6. Visitor – dynamiczne rozszerzanie API Klasy (emulacja Double Dispatch)

4.7. Iterator – standardowy wzorzec dla kolekcji

4.8. Interpreter – wygodny wzorzec dla tworzenia własnych DSL

4.9. Observer - systemy zorientowane na zdarzenia, Składowa MVC

5. Testability – wpływ użycia dobrych praktyk OOD i Wzorców na testowalność kodu

5.1. Zagadnienia podatności architektury na testy: problemy i pułapki

5.2. Techniki testowania jednostkowego: dummy, fake, stub, mock

5.3. Narzędzia testowania jednostkowego i integracyjnego

5.3.1. JUnit

5.3.2. Mockito

6. Wzorce Architektoniczne – projektowanie nowoczesnych architektur aplikacji Enterprise

6.1. Przegląd architektur: Layers, Microkernel, Pipes and Filters, Blackboard, Command-query Responsibility Segregation, Event oriented

6.2. Praktyczne wykorzystanie technik Inversion of Control do budowy frameworków i systemów – na przykładzie Spring lub Seam (do wyboru)

6.2.1. Dependency Injection

6.2.1.1. Wykorzystanie zamiast wzorców fabrykujących

6.2.1.2. Budowanie konkretnych Strategii, Dekoratorów itd. w zależności od stanu aplikacji (kontekst, konfiguracja)

6.2.2. Systemy sterowane zdarzeniami

6.2.2.1. Użycie do tworzenie rozszerzalnych architektur opartych o pluginy

6.2.2.2. Użycie do tworzenia skalowalnych systemów wysokiej wydajności

6.2.3. Aspect Oriented Programming