BO·TT·EGA
IT minds

Program szkolenia:

Wzorce projektowe i architektoniczne oraz efektywne techniki Object Oriented Design dla projektantów systemów

Informacje:

Nazwa:	**Wzorce projektowe i architektoniczne oraz efektywne techniki Object Oriented Design dla projektantów systemów**
Kod:	**craft-patterns-Patterns Sys**
Kategoria:	Wzorce projektowe
Odbiorcy:	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.

Program jest ogólną ramą merytoryczną. Jego realizacja wykracza poza 3 dni. W trakcie analizy przedszkoleniowej wybieramy wzorce, które będą przydatne dla zespołu i skupiamy się tylko na nich w trakcie szkolenie. W ciągu 3 dni zwykle jesteśmy w stanie zaadresować ok 70% wymienionych wzorców.

Zalety szkolenia:

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

BOTTEGA IT minds

Szczegółowy program:

- **1. Wzorce architektury systemowej**
  - 1.1. Dostępu do danych
  - 1.2. Warstwy
    - 1.2.1. Podejścia do warstw
    - 1.2.2. Relaxed Layers
  - 1.3. Micro Kernel
  - 1.4. Command and Command Handler
  - 1.5. Ports and Adapters
  - 1.6. Restful
    - 1.6.1. 4 poziomy dojrzałości
    - 1.6.2. REST jako protokół aplikacyjny a nie transportowy
  - 1.7. Event Driven
    - 1.7.1. Zależności czasowe pomiędzy zdarzeniami
    - 1.7.2. Sagi – orkiestracja zdarzeń
- **2. Wzorce architektury aplikacji**
  - 2.1. Podział na logikę aplikacji i logikę domenową
  - 2.2. Logika aplikacji
    - 2.2.1. Modelowanie Use Case/User Story
    - 2.2.2. Stanowo czy bezstanowo
    - 2.2.3. Wykorzystanie Aspect Oriented Programming
  - 2.3. Logika domenowa
    - 2.3.1. Techniki Domain Driven Design - Building Blocks
      - 2.3.1.1. Agregaty

BO·TT·EGA IT minds

2.3.1.2. Encje

2.3.1.3. Value Objects

2.3.1.4. Serwisy Domenowe

2.3.1.5. Polityki

2.3.1.6. Wykorzystanie Dependency Injection

2.3.1.7. Specyfikacje

2.3.1.8. Fabryki

2.3.1.9. Repozytoria

2.3.2. Modelowanie niezmienników

2.3.3. Poziomomy modelu

2.3.3.1. Capacity

2.3.3.2. Operations

2.3.3.3. Policy – dostrajanie modelu

2.3.3.4. Decission Support

3. Paradygmat Inversion of Control – sprawdzona koncepcja budowy frameworków i systemów

3.1. Dependency Injection – podstawa współczesnych frameworków - dokładne omówienie z zadaniami

3.1.1. Wsparcie dla testability

3.1.2. Praktyczne techniki wykorzystania w celu osiągnięcia giętkości designu

3.1.3. Wstrzykiwanie poprzez kontenery

3.1.4. Wstrzykiwanie w Fabrykach

3.2. Systemy sterowane zdarzeniami – omówienie koncepcji i problemów

3.2.1. Architektura pluginowa

3.2.2. Separacja modułów

3.2.3. Zwiększanie responsywności systemu

BO·TT·EGA IT minds

3.2.4. Skalowanie

4. Aspect Oriented Programming – omówienie koncepcji i przegląd podejść

4.1. Praktyczne zastosowania

4.2. Przykłady (Spring)

5. Techniki Object Oriented Design.

5.1. Analiza Paradygmatu Object oriented i jego poprawna interpretacja.

5.2. Ukierunkowanie myślenia w stylu OO.

5.3. Najlepsze praktyki i pułapki.

5.3.1. Wady naiwnego modelowania rzeczowniki-czasowniki

5.3.2. Modelowanie hermetycznych agregatów - model behawioralny

5.3.3. Modelowanie niezmienników.

5.3.4. Modelowanie czasu jako jednego z głównych czynników złożoności esencjonalnej

5.4. GRASP - General Responsibility Assignment Software Patterns.

5.5. 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).

5.6. Antywzorce i typowe pułapki

5.6.1. Code smell – wykrywanie ok 20 zapachów kodu

5.6.2. Przegląd typowych błędów i pułapek

5.6.3. Techniki refaktoryzacji

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

6.1. Command – hermetyzacja usług, autoryzacja dostępu

6.2. Decorator – reużywalnośc logiki

6.2.1. Składanie złożonej logiki biznesowej o przyrostowym charakterze

6.2.2. Wrapper – odmiana wzorca użyteczna w modelowaniu zorientowanym na znaczenie

BO·TT·EGA IT minds

6.2.2.1. Opakowanie typów podstawowych wygodnymi obiektami

6.2.2.2. Alternatywa dla Utils

6.2.2.3. Archetyp Money

6.2.3. Połączenie Dekoratora ze Strategią w celu zbudowania odmian algorytmów przyrostowych

6.3. Strategy – hermetyzacja logiki biznesowej

6.3.1. Wybór odmiany algorytmu bez ingerencji w core biznesowy

6.3.2. Integracja z mechanizmami wstrzykiwania zależności

6.3.3. Definiowane konkretnej strategii biznesowej w kontenerze Incersion of Control (XML lub metody fabrykujące)

6.4. Chain of Responsibity – dwie odmiany wzorca

6.4.1. Dobór logiki biznesowej do aktualnych warunków

6.4.2. Połączenie Łańcucha ze Strategią w celu zbudowania odmian algorytmów warunkowych

6.5. Abstract Factory – tworzenie artefaktów domenowych

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

6.5.2. Produkowanie Strategii

6.6. Builder – redukcja złożoności tworzenia struktur

6.6.1. Zunifikowane eksportowanie obiektów domenowych

6.6.2. Ukrywanie złożoności budowania zapytań

6.7. Template Method – szablony procesów.

6.7.1. Technika uwspólniania logiki biznesowej.

6.7.2. Szablonowe Strategie

6.7.3. Przykłady antywzorca.

6.8. Singleton – niebezpieczny wzorzec w przykładach.

6.8.1. Szczegóły implementacji Singeltonów tworzonych z opóźnieniem, odpornych na współbieżny dostęp.

6.9. State – hermetyzacja procesu biznesowego.

6.9.1. Implementacja maszyny stanów reprezentującej złożony cykl życia obiektu biznesowego.

6.9.2. Maszyna Stanów jako Wrapper dodający nowe funkcjonalności.

6.9.3. Pułapki zbytniego uogólniania Maszyn Stanów

6.10. Observer – redukcja zależności

6.11. Event Broker - uogólnienie do poziomu Architektury Systemu

6.12. Wysokowydajne systemy Event Driven i asynchroniczne przetwarzanie.

6.13. Saga - orkiestracja wielu zdarzeń w czasie

6.14. Specification – hermetyzacja reguł biznesowych

6.14.1. Redukcja złożoności systemów zawierających złożoną logikę decyzyjną

6.14.2. Przypadek gdy istnieje wiele możliwych kryteriów logicznych

6.14.3. Jednak w danym kontekście (wdrożenie, klient) używanym tylko podzbioru reguł

6.15. Role Object

6.15.1. Modelowanie ról w systemie.

6.15.2. Alternatywa dla dziedziczenia.

6.15.3. Odmiana wzorca Bridge.

6.16. Facade – redukcja złożonej struktury pod wygodnym API

7. Użyteczne wzorce o zastosowaniu technicznym

7.1. Bridge – rozdzielenie interfejsu koncepcyjnego od jego implementacji

7.2. Flyweight, Prototype – wzorce optymalizacji

7.3. Memento – komunikacja pomiędzy kontekstami

7.4. Proxy – podstawowy wzorzec frameworków

7.5. Composite – powtarzalne struktury

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

BOTTEGA IT minds

7.7. Iterator – standardowy wzorzec dla kolekcji

7.8. Interpreter – wygodny wzorzec dla tworzenia własnych DSL

7.9. Observer - systemy zorientowane na zdarzenia, Składowa MVC

7.10. Extension Object - uogólnienie Role Object dla systemów otwartych na rozbudowę.

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

8.1. Zagadnienia podatności architektury na testy: problemy i pułapki

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

8.3. Narzędzia testowania jednostkowego i integracyjnego

8.3.1. JUnit

8.3.2. Mockito

9. Dokumentowanie architektury w podejściu 4C - Context, Containers, Components, Classes

9.1. Context

9.1.1. Kontekst w jakim będzie działał system

9.1.2. Główni aktorzy

9.1.3. Procesy na poziomie organizacji w które zaangażowany jest system

9.1.4. Rola innych systemów

9.2. Containers

9.2.1. Kontenery techniczne w jakich będzie rozmieszczony system

9.2.2. Wzorce integracji

9.2.3. Protokoły komunikacji

9.3. Components

9.3.1. Komponenty i Serwisy w ujęciu SOA

9.3.2. Dane kanoniczne wymieniane pomiędzy Serwisami

9.3.3. Projektowanie API

9.3.4. Zasada jednego źródła prawdy

9.4. Classes

9.4.1. Wstęp do technik i wzorców projektowania na poziomie klas