BO·TT·EGA
IT minds

Program szkolenia:

Receptury - niezbędnik projektanta i architekta

Informacje:

Nazwa:	**Receptury - niezbędnik projektanta i architekta**
Kod:	**Craft-Receptury**
Kategoria:	Wzorce i Craftsmanship
Grupa docelowa:	developerzy architekci
Czas trwania:	3 dni
Forma:	50% wykłady / 50% warsztaty

Program szkolenia został zaprojektowany z myślą o typowych problemach projektowych i architektonicznych, które pojawiają się w niemal każdym systemie oraz sprawdzonych receptach na ich rozwiązanie.

Szkolenie prowadzone jest w podejściu zorientowanym na problem. Każdy moduł to opis problemu oraz alternatywnych sposobów jego rozwiązania wraz z omówieniem konsekwencji wynikających z danego podejścia.

Podejścia i techniki przedstawione podczas szkolenia są nieodzownym składnikiem "skrzynki z narzędziami" każdego projektanta i architekta.

Materiały wstępne

Przed szkoleniem możesz zapoznać się z serią naszych artykułów: Receptury projektowe – niezbędnik początkującego architekta.

Zalety szkolenia:

- Typowe problemy i ich rozwiązania
- Wykorzystanie sprawdzonych technik i wzorców
- Realne przykłady

BO·TT·EGA
IT minds

Szczegółowy program:

1. Zarządzanie złożonością modeli biznesowych

- 1.1. Rozwarstwienie logiki w celu uporządkowania odpowiedzialności
  - 1.1.1. Logika Aplikacji - model Use Case/User Story
  - 1.1.2. Logika Domenowa - model reguł biznesowych
- 1.2. 4 poziomy modelu w celu zarządzania zmianą logiki domenowej
  - 1.2.1. Decission Support - logika analityczna
  - 1.2.2. Policy - dostrojenie modelu
  - 1.2.3. Operations - model operacji
  - 1.2.4. Capability - model potencjału
- 1.3. Podejście archetypów modeli biznesowych
  - 1.3.1. Knowledge Level - modele ulegające zmianie
  - 1.3.2. Operation Level - modele nieulegające zmianie
- 1.4. Podział kodu na 3 grupy - Open Close Principle
  - 1.4.1. Logika Stała
    - 1.4.1.1. Wydzielenie logiki, która podlega relatywnie niewielu zmianom
  - 1.4.2. Rozbudowana
    - 1.4.2.1. Polityki
    - 1.4.2.2. Wzorzec Strategii
    - 1.4.2.3. Podejście funkcyjne
  - 1.4.3. Zmiana
    - 1.4.3.1. Fabryki - hermetyzacja reguł wyboru polityki/strategii/funkcji
    - 1.4.3.2. Parametryzacja przy pomocy kontenerów IoC

2. Architektury aplikacji otwarte na rozbudowę

BOTTEGA IT minds

2.1. Typowe problemy w kastomizowalnych systemach dostosowywanych per wdrożenie

2.1.1. Unikanie tworzenia branch w repozytorium kodu

2.1.2. Unikania wprowadzania logiki warunkowej

2.2. Alternatywa: podejście plugin-oriented

2.3. Wykorzystanie kontenerów IoC

2.3.1. Wstrzykiwanie zależności - pluginy typu "coś działa inaczej"

2.3.1.1. Wstrzykujemy nie tylko DAO

2.3.1.2. Zarządzenie odmiennością logiki

2.3.1.3. Wstrzykiwanie Strategy Design Pattern

2.3.2. Event Driven - pluginy typu "dodatkowa funkcjonalność"

2.3.2.1. Listenery jako pluginy

2.3.2.2. Listenery jako alternatywne klienty do API

2.4. Extension Object Pattern

3. Modularyzacja

3.1. Projektowanie API modułu

3.2. Hermetyzacja modułu w celu separacji zmian

3.3. Podejście Bounded Context

3.3.1. Granica modułu wyznaczana przez ograniczenia wiedzy

3.4. Model kanoniczny

3.5. Integracja modułów

3.5.1. Podejścia SOA

3.5.2. Podejście Event Driven

3.5.2.1. Listenery jako alternatywne klienty do API

3.5.3. Kiedy warto wprowadzić Data Transfer Object - a kiedy jest to zbędny narzut

3.6. Architektura Heksagonalna (Ports and Adapters)

BOTTEGA IT minds

4. DDD Lite - Building Blocks modelowanie złożonych domen

- 4.1. Techniki Domain Driven Design - Building Blocks
- 4.2. Agregaty
  - 4.2.1. Modelowanie niezmienników
  - 4.2.2. Modelowanie granicy Agregatu jako jednostki pracy
- 4.3. Encje
- 4.4. Value Objects
- 4.5. Serwisy Domenowe
- 4.6. Polityki
  - 4.6.1. Wykorzystanie Dependency Injection
- 4.7. Specyfikacje
  - 4.7.1. Modelowanie złożonych kryteriów
- 4.8. Fabryki
  - 4.8.1. Modelowanie złożonego procesu tworzenia obiektów
  - 4.8.2. Wartości początkowe, walidacja półproduktów, składanie obiektów, wstrzykiwanie zależności
- 4.9. Repozytoria
  - 4.9.1. Abstrakcja źródła danych

5. Testability - projektowanie kodu otwartego na testowanie

- 5.1. Typowe błędy projektowe
  - 5.1.1. Eksplozja kombinatoryczna przypadków wynikająca z braku rozwarstwienia logiki
  - 5.1.2. Konieczność testowania przez UI wynikająca z nieprzestrzegania warstw
  - 5.1.3. Konieczność testowania z bazą danych wynikająca z braku separcji odpowiedzialności
- 5.2. Techniki i taktyki
  - 5.2.1. Unikanie pracy z serwerem aplikacyjnym i bazodanowych

BO·TT·EGA
IT minds

5.2.2. Podejście funkcyjne - testowanie funkcji

5.2.2.1. Projektowanie Serwisów Domenowych

5.2.2.2. Projektowanie oparte o Strategy Design Pattern

5.2.3. Hermetyzacja zależności w Fabrykach

5.3. Modelowanie i testowanie niezmienników

5.4. Projektowanie klas testowych

5.4.1. Unikanie potrzeby tworzenia Mock/Stub/Fake

5.4.2. Wprowadzanie Object Mother

5.4.3. Wprowadzanie Assemblerów obiektów

5.4.4. Wprowadzenie własnych Assert Object

6. Modelowanie ról w systemie

6.1. Wady podejścia opartego o dziedziczenie

6.2. Role Object Pattern

6.2.1. Role zapewniające funkcjonalność

6.3. Party Archetype i Party Relationship Archetype

6.3.1. Model uczestników relacji

6.3.2. Integracja z Role Object Pattern

7. Konglomeraty wzorców - struktury rozwiązań typowych problemów

7.1. Fabryka Strategii - Separacja logiki stałej od zmiennej oraz zmiany od rozbudowy

7.1.1. Zapewnienie Open Close Principle

7.1.2. Fabryka Dekorowanych Strategii

7.1.3. Fabryka Strategii Szablonowych

7.1.4. Fabryka Strategii Łańuchowych

7.2. Command operujący na Micro Kernel - Separacja logiki API od aplikacji

7.2.1. Wsparcie dla operacji Undo

BO·TT·EGA
IT minds