BO·TT·EGA
IT minds

# Program szkolenia:

# Dobry start dla początkujących programistów - wprowadzenie do współczesnej inżynierii oprogramowania

## Informacje:

| | |
|---|---|
| **Nazwa:** | **Dobry start dla początkujących programistów - wprowadzenie do współczesnej inżynierii oprogramowania** |
| **Kod:** | **Craft-practices-start** |
| **Kategoria:** | Craftsmanship |
| **Odbiorcy:** | developerzy |
| **Czas trwania:** | 3 dni |
| **Forma:** | 50% wykłady / 50% warsztaty |

Program szkolenia został zaprojektowany z myślą o początkujących programistach lub doświadczonych programistach migrujących z języków nieobiektowych.

Celem szkolenia jest szybkie i efektywne wprowadzenie w świat programowania skupiając się na tych aspektach, które są szczególnie istotne w codziennej praktyce.

Rzetelne ogólne podstawy dają swobodę w uczeniu się szczególnych technologii i frameworków, ponieważ są uniwersalne.

## Zalety szkolenia:

- Tylko najważniejsze tematy, bez certyfikatowych kruczków
- Szersze spojrzenie pozwalające ocenić: co, kiedy i po co
- Podstawy, które pozwalają zrozumieć wynikające z nich zagadnienia zaawansowane

BO·TT·EGA
IT minds

# Szczegółowy program:

## 1. Techniki Object Oriented Design.

- 1.1. Analiza Paradygmatu Object oriented i jego poprawna interpretacja.
- 1.2. Ukierunkowanie myślenia w stylu OO.
- 1.3. Podstawowe Building Blocks
  - 1.3.1. Obiekty
  - 1.3.2. Struktury danych
  - 1.3.3. Procedury
  - 1.3.4. Funkcje
  - 1.3.5. Implementacja (i emulacja) w językach obiektowych
- 1.4. Najlepsze praktyki i pułapki.
  - 1.4.1. Wady naiwnego modelowania rzeczowniki-czasowniki
  - 1.4.2. Modelowanie hermetycznych agregatów - model behawioralny
  - 1.4.3. Modelowanie niezmienników.
  - 1.4.4. Modelowanie czasu jako jednego z głównych czynników złożoności esencjonalnej
- 1.5. Couping - 3 rodzaje
  - 1.5.1. Call
  - 1.5.2. Contain
  - 1.5.3. Create
- 1.6. GRASP - General Responsibility Assignment Software Patterns.
- 1.7. SOLID
  - 1.7.1. Single Responsibility Principle (SRP)
  - 1.7.2. Open/Closed Principle (OCP)
  - 1.7.3. Liskov Substitution Principle (LSP)

BO·TT·EGA IT minds

1.7.4. Dependency Inversion Principle (DIP)

1.7.5. Interface Segregation Principle (ISP).

1.8. Antywzorce i typowe pułapki

1.8.1. Błędy dziedziczenia

1.8.2. Typowe błędy w modelowaniu obiektowym

1.8.2.1. Superklasy

1.8.2.2. Wyciąganie "przed nawias"

1.8.2.3. Struktury danych zamiast agentów

1.8.3. Code smell – wykrywanie ok 20 zapachów kodu

1.8.4. Techniki refaktoryzacji

**2. Zasady czystego kodu**

2.1. Nazewnictwo

2.2. Przypisywanie odpowiedzialności

2.3. Obsługa błędów

2.4. Typowe błędy

2.5. Zarządzanie widocznością

2.6. Ukrywanie niewiedzy

2.7. Powtórzenia są złe o ile nie są dobre

**3. Wzorce implementacyjne**

3.1. Zasady pakietowania

3.1.1. Jedna klasa publiczna na pakiet - kiedy ma to sens

3.1.2. Moduły czy warstwy

3.2. Konwencje kodowania

3.3. Obsługa wyjątków

3.3.1. Checked, Unchecked albo status - czym się kierować

BOTTEGA IT minds

## 4. Podstawowe wzorce projektowe w praktycznych przykładach do codziennego wykorzystania

- 4.1. Strategy
  - 4.1.1. Podejście funkcyjne
  - 4.1.2. Open-closed Principle
- 4.2. Template Method
- 4.3. Factory Idiom
  - 4.3.1. Zarządzanie couplingiem
- 4.4. Chain of Responsibility i jego modyfikacje
  - 4.4.1. Zastosowanie do walidacji
- 4.5. State Machine

## 5. Podstawowe architektury aplikacyjne

- 5.1. Rozróżnienie architektury aplikacyjnej od systemowej
- 5.2. Design of Deisgn
- 5.3. Podejście warstwowe
  - 5.3.1. Różnice pomiędzy Layer a Tier
  - 5.3.2. Ogólny podział - 3I
  - 5.3.3. Interfejsy
  - 5.3.4. Interakcje
  - 5.3.5. Integracja
- 5.4. Warstwa prezentacji
  - 5.4.1. MVC, MVP, MVVM
  - 5.4.2. Front Controler i Page Controler
  - 5.4.3. DTO - zastosowanie
  - 5.4.4. Ekrany komponentowe

BO·TT·EGA IT minds

5.5. Podział na logikę aplikacji i logikę domenową

5.5.1. Logika aplikacji

5.5.1.1. Modelowanie Use Case/User Story

5.5.1.2. Stanowo czy bezstanowo

5.5.1.3. Problemy projektowania API modułu

5.5.2. Logika domenowa i Techniki Domain Driven Design - Building Blocks

5.5.2.1. Agregaty, Encje, Value Objects

5.5.2.2. Serwisy Domenowe, Polityki, Specyfikacje

5.5.2.3. Fabryki, Repozytoria

**6. Praktyczne wykorzystanie technik Inversion of Control do budowy frameworków i systemów – na przykładzie Spring/EJB/Seam (do wyboru)**

6.1. Dependency Injection – podstawowa technika IoC

6.1.1. Wykorzystanie zamiast wzorców fabrykujących

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

6.1.3. Otwartość na rozbudowę dzięki wzorcom Strategii

6.2. Systemy sterowane zdarzeniami – silniejsza technika IoC

6.2.1. Użycie do tworzenie rozszerzalnych architektur opartych o pluginy

6.3. Aspect Oriented Programming

6.3.1. Wstęp do AOP

6.3.2. Przykłady zastosowania AOP

**7. Testability – projektowanie architektur aplikacji zorientowanych na testy**

7.1. Strategiczne testowanie

7.1.1. Problem eksplozji kombinatorycznej przypadków testowych

7.1.2. Mapowanie warstw aplikacji na piramidę testów

7.2. Strategia

BO·TT·EGA IT minds

7.2.1. Warstwa domenowa - testowanie jednostkowe

7.2.2. Warstwa serwisów - testowanie end2end

7.3. Dążenie do uruchamiania logiki poza serwerem – zwiększanie produktywności, redukcja czasu używanego na redeploy

7.4. Zagadnienia podatności architektury na testy: problemy i pułapki

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

7.6. Narzędzia testowania jednostkowego i integracyjnego (xUnit, Mockito)

7.7. 3 poziomy testów

7.7.1. Specification by Example - cele biznesowe

7.7.2. Flow - User Story

7.7.3. Automatyzacja interakcji z UI - Agenty będące abstrakcją nad skryptem testowym