

# WZORCE PROJEKTOWE I REFAKTORYZACJA DO WZORCÓW

KOD: WP

# Profil uczestnika

Programista:

- zna obiektowy język programowania;
- posiada doświadczenie w tworzeniu systemów informatycznych;
- chce efektywnie posługiwać się wzorcami projektowymi.

# Korzyści ze szkolenia

1. **Wzrasta bezpieczeństwo tworzonego oprogramowania** – gdy programiści używają wzorców projektowych, kod jest łatwiejszy do testowania.
2. **Skróca się czas poświęcony na dodawanie nowych funkcjonalności** – ponieważ osoby, które posługują wzorcami projektowymi, piszą kod otwarty nad rozbudowę i zamknięty na modyfikacje. Wzorce projektowe zapewniają łatwe rozbudowywanie systemu.
3. **Polepsza się jakość komunikacji pomiędzy programistami** – gdy programiści komunikują się poprzez kod używając wzorców projektowych, porozumiewają się ze sobą na wyższym poziomie abstrakcji.

# Parametry szkolenia

CZAS TRWANIA: 3 dni - 24 godziny.

FORMA ZAJĘĆ: Laboratorium Wzorców Projektowych - 60%, wykład – 40%.

WIELKOŚĆ GRUPY: do 10 osób.

JĘZYKI PROGRAMOWANIA: Java, C#.

# Szczegółowy program

Moduły szkoleniowe	Nabyte wiedza i umiejętności, poruszane zagadnienia
<b>Wzorce GoF</b>	<ul style="list-style-type: none"><li>• Pojęcie wzorca projektowego</li><li>• Wzorce kreacyjne - przegląd</li><li>• Simple Factory</li><li>• Factory Method</li><li>• Builder</li><li>• Abstract Factory</li><li>• Prototype</li><li>• Singleton</li><li>• Wzorce strukturalne - przegląd</li><li>• Adapter</li><li>• Decorator</li><li>• Facade</li><li>• Proxy</li><li>• Decorator</li><li>• Bridge</li><li>• Flyweight</li><li>• Strategie implementacji wzorców</li><li>• Różnice pomiędzy podobnymi wzorcami</li></ul>
<b>Wzorce GoF(2)</b>	<ul style="list-style-type: none"><li>• Wzorce behawioralne – przegląd</li><li>• Command</li><li>• Strategy</li><li>• Observer</li><li>• Chain of Responsibility</li><li>• Template Method</li><li>• Observer</li><li>• Mediator</li><li>• Visitor</li><li>• Iterator</li><li>• State</li><li>• Różne sposoby implementacji wzorców</li><li>• Różnice pomiędzy podobnymi wzorcami</li></ul>

	<ul style="list-style-type: none"><li>• Aspekty współpracy pomiędzy wzorcami</li><li>• Antywzorce projektowania obiektowego</li></ul>
<b>Architektura warstwowa i wzorce architektoniczne</b>	<ul style="list-style-type: none"><li>• Porównanie modeli warstwowych</li><li>• Model warstwowy wg M. Fowlera</li><li>• Odpowiedzialność modelu warstwowym</li><li>• Modelowanie dziedziny problemu</li><li>• Wzorce Domain Model</li><li>• Różne podejścia do warstwy danych</li><li>• Data Access Object</li><li>• Object-Relational Mapping</li><li>• Active Record</li><li>• Wzorce modelu zdarzeniowego</li><li>• Wzorce stanu tymczasowego</li><li>• Temporal Object</li><li>• Snapshot</li><li>• Aspekty współpracy pomiędzy wzorcami</li></ul>
<b>Refaktoryzacja do wzorców</b>	<ul style="list-style-type: none"><li>• Podstawy refaktoryzacji</li><li>• Kryteria jakości kodu</li><li>• Wybrane refaktoryzacje podstawowe</li><li>• Zmiana nazwy zmiennej, metody, klasy</li><li>• Wyodrębnienie metody</li><li>• Wyodrębnienie zmiennej</li><li>• Odzwierciedlenie kroków algorytmu</li><li>• Kierunek wprowadzania wzorców</li><li>• Sygnały sugerujące wprowadzanie wzorca</li><li>• Zamiana instrukcji warunkowych na wzorec Strategy</li><li>• Przesunięcie kodu kreacyjnego do wzorca z rodziny Factory</li><li>• Przesunięcie elementów dodatkowych do wzorca Decorator</li><li>• Zastąpienie warunkowych wykonań poprzez wzorec Command</li><li>• Zastąpienie bezpośredniej komunikacji wzorcem Observer</li><li>• Zasada Odpowiedzialności</li><li>• Pragmatyzm we wprowadzaniu wzorców projektowych</li></ul>