Kod szkolenia: **REFAKT/ADV**

Tytuł szkolenia: **Rozwój oprogramowania z wykorzystaniem Refaktoryzacji**

Dni: 2

Opis:

Adresaci szkolenia:

Szkolenie skierowane jest do programistów i architektów, którzy pragną poznać techniki rozpoznawania, przeprowadzania oraz planowania złożonych jak i ryzykowanych refaktoryzacji.

Cel szkolenia:

Cel szkolenia obejmuje poznanie i opanowanie następujących zagadnień/umiejętności:

- Pisanie wysokiej jakości testów.
- Znajomość technik rozpoznania problemów, które występują w kodzie.
- Umiejętność doboru odpowiedniej strategii refaktoryzacji do problemu.
- Umiejętność refaktoryzacji kodu nieprzetestowanego

Wymagania:

Uczestnik szkolenia powinien posiadać podstawowe doświadczenie w programowaniu obiektowym (preferowanym językiem jest Java), testowaniu oraz refaktoryzacji.

Parametry szkolenia

2 * *8 godzin (2 *7* godzin netto) wykładów i warsztatów w proporcji: 80% warsztaty, dyskusje; 20% wykłady.

Program szkolenia:

1. Wprowadzenie
    - Rozwój, a tworzenie aplikacji
    - Legacy Code, a Technical Debt
2. Prewencja
    - Fast feedback
    - Podejmowanie decyzji i odsuwanie ich w czasie
    - Akceptacja jako sposób na radzenie sobie z problemami, których nie rozwiążesz

- Czy można uniknąć degradacji jakości kodu?
- Projektowanie aplikacji jako sposób na kontrolę degradacji jakości
- Prewencja ważniejsza niż leczenie

3. Testowanie, a bezpieczeństwo
   - Piramida testów
     - Unit czy Component Test?
     - Testy integracyje
     - Component czy System Test?
   - Test Double Patterns
     - Rodzaje
     - Boundary Objects
     - Niebezpieczeństwa Test Double
   - Jak mierzenie pokrycia kodu może pomóc, a jak zaszkodzić?
4. Refaktoryzacja
   - Kiedy refaktoryzacja ma sens?
   - Niebezpieczeństwa refaktoryzacji
   - Refaktoryzacja, a testowanie
     - Czy testy są niezbędne?
     - Kiedy posiadanie testów szkodzi?
     - Jak poradzić sobie bez testów?
   - Jak rozpocząć refaktoryzację?
     - Poznaj swoje IDE
     - Wizualizacja problemów
     - Implementation-Driven Tests
   - Małe refaktoryzacje obarczone niewielkim ryzykiem
     - Jak nie zmarnować czasu przeznaczone na zrozumienie kodu?
     - Ekstrakcja
     - Zadbanie o widoczność
     - Wyrzuć to, czego nie potrzebujesz
   - Poprawa jakości poprzez zmianę designu
     - Jak testować Singletony?
     - Interfejsy z jedną implementacją – warto czy nie warto?
     - Dlaczego nie łączyć klasy abstrakcyjnej i interfejsu?
     - Refaktoryzacja klas abstrakcyjnych
     - Usuwanie zbędnych interfejsów
     - Dlaczego warto posiadać klasy odpowiedzialne za tworzenie obiektów?
     - Granica pomiędzy kodem produkcyjnym, a testami.
     - Exceptions Driven Flow
   - Refaktoryzacje do wzorców
     - Czy zawsze warto?
     - Kiedy warto pozbyć się wzorców?
     - Command, Strategy czy State?
     - Strategy czy Template Method?
     - Chain of Responsibility czy Decorator?
     - Adapter and Boundary Object
     - Fabryka jako sposób kontroli spójności i enkapsulacji

        - Facade jako sposób kontroli spójności i enkapsulacji
        - Facade, a Anti-Corruption Layer
        - Builder
        - Observer
        - Mediator
        - Visitor
    - Jak odzyskać wiedzę domenową?
        - Adapter i Bounded Context
        - Warunki, czyli jak zrozumieć pytanie?
        - Korzyści z wprowadzenia Value Objects
        - Kolekcje czy własne obiekty?
        - Jak odnaleźć utracone informacje i je odzyskać?
5. Dodawanie nowej funkcjonalności
    - Eliminacja zdegradowanego kodu, a Strangler pattern
    - Jak się przygotować?
    - Proof of Concept jako sposób na rozpoznanie problemu
    - Minimalizowanie ryzyka wynikającego z potencjalnych błędów
        - Ryzyka wynikające z Feature Branch
        - Feature toggles
        - Jak planować migracje?
        - Jak testować kod bez testów?
6. Podsumowanie