

## Program szkolenia:

# Receptury testowania automatycznego - problemy, strategie, taktyki, techniki, narzędzia

## Informacje:

<b>Nazwa:</b>	<b>Receptury testowania automatycznego - problemy, strategie, taktyki, techniki, narzędzia</b>
<b>Kod:</b>	<b>craft-test-receptury</b>
<b>Kategoria:</b>	Testowanie automatyczne
<b>Grupa docelowa:</b>	developerzy
<b>Czas trwania:</b>	3 dni
<b>Forma:</b>	50% wykłady / 50% warsztaty

Szkolenie przeznaczone jest dla programistów, którzy posiadają podstawowe umiejętności testowania automatycznego i pragną poszerzyć je o szersze spojrzenie na problemy i strategie ich rozwiązywania.

## Uczestnicy szkolenia:

- zostaną zapoznani z technikami, które są użyteczne w kontekście ich pracy
- następnie nabędą umiejętność świadomego składania technik w taktyki
- w kolejnym etapie będą uczyć się klasyfikować problemy i dobierać do nich odpowiednie strategie (na które składają się taktyki i techniki poznane wcześniej)

## Praktyki i techniki działają na poziomach:

- wymagań,
- architektury,
- modelu domenowego,
- procesu implementacji,
- testów (end-to-end i jednostkowych).

## Materiały wstępne

Przed szkoleniem możesz zapoznać się z serią naszych artykułów: [Testowanie automatyczne](#).

## Zalety szkolenia:

- Zagadnienia architektury aplikacji wspierającej testowalność kodu

- Najlepsze wzorce i praktyki
- Aspekty Behavior Driven Development i Domain Driven Design

## Szczegółowy program:

### 1. Problemy

- 1.1. Eksplozja kombinatoryczna przypadków – w górnych warstwa multiplikują się możliwe stany obiektów domenowych
- 1.2. Koszt stworzenia i utrzymania testów – zaślepki (Fake/Mock/Stub), dane testowe np. w bazie danych
- 1.3. Nieaktualna dokumentacja (nikt jej nie czyta ani nie aktualizuje)
- 1.4. Problem z komunikacją - brak zrozumienia celów biznesowych, biznes nie rozumie systemu
- 1.5. Kosztowne w utrzymaniu skrypty do „wyklikania”
- 1.6. Dobór zakresu testów do rzeczywistych potrzeb (Unit, End-to-End-komponent, End-to-End-system, Integracyjne)
- 1.7. Rola testów: jakość dev. czy jakość biz.
- 1.8. Architektura wspierająca testability (zależności, rozwarstwienie i rygor warstw)
- 1.9. OO Design (SOLID, GRASP, RDD, DDD)
- 1.10. Smells: Delikatne testy (fragile), Nieczytelne testy, Wolne testy, Testy niedeterministyczne

### 2. Strategie

- 2.1. Mapowanie piramidy testów na warstwy aplikacji
- 2.2. Perfekcja domeny, ogólna zgodność wymagań
- 2.3. Obniżanie kosztów poprzez unikanie zbędnej pracy
- 2.4. Tworzenie wykonywalną dokumentację
- 2.5. Skupienie dokumentację wokół celów biznesowych i flow użytkownika
- 2.6. Operowanie słownictwem domenowym
- 2.7. Orientacja na Flow a nie na skrypt UI
- 2.8. Orientacja na Specyfikację a nie Flow
- 2.9. Ciągła refaktoryzacja

### 3. Taktyki

- 3.1. Warstwa Aplikacji – Testy End-to-end – Komponentowe
- 3.2. Warstwa Domenowa – Testy jednostkowe
- 3.3. Abstrakcje Warstwy Infrastruktury – Testy integracyjne
- 3.4. Unikaj pracy z serwerem
- 3.5. Dwuwarstwowe Historyjki akceptacyjne
- 3.6. Trójwarstwowe Wykonywalne Specyfikacje
- 3.7. Struktura Given-When-Then: W proponuje dev, GT definiuje ekspert
- 3.8. Testowanie przed refaktoryzacją (odpowiednie testy)

### 4. Techniki

- 4.1. Struktura Testów
- 4.2. Testowanie niezmienników Agregatów
- 4.3. Assembler Agregatów
- 4.4. Assert Object
- 4.5. Testowanie Serwisów Domenowych w stylu funkcyjnym
- 4.6. Jak tworzyć zaślepki: Command-Mock, Query-Stub
- 4.7. Testowanie Serwisów Domenowych w stylu funkcyjnym
- 4.8. Agenty abstrahujące od protokołu komunikacyjnego
- 4.9. Struktura Given-When-Then: W przez UI, GT niżej
- 4.10. Niektóre specyfikacje testuj jednostkowo w warstwie domeny
- 4.11. Techniki świadomej refaktoryzacji

### 5. Narzędzia

- 5.1. Narzędzia testowania jednostkowego (frameworki, asercje, matchery, mockowanie)
- 5.2. Narzędzia testowania end-to-end
- 5.3. Narzędzia automatyzacji UI

## 5.4. Narzędzia mapowania wykonywalnych specyfikacji na kroki