

## Program szkolenia:

# Strategie automatyzacji testowania i utrzymywania wysokiej jakości

### Informacje:

<b>Nazwa:</b>	<b>Strategie automatyzacji testowania i utrzymywania wysokiej jakości</b>
<b>Kod:</b>	<b>QA-qa</b>
<b>Kategoria:</b>	Testowanie dla QA
<b>Grupa docelowa:</b>	developerzy testerzy
<b>Czas trwania:</b>	2 dni
<b>Forma:</b>	50% wykłady / 50% warsztaty

Podczas szkolenia uczestnicy zdobędą wiedzę na temat testowania oraz automatyzacji podczas wszystkich etapów procesu wytwarzania oprogramowania. Omówione zostaną konkretne techniki testowania "white-" i "black-box".

Podczas warsztatów praktycznych uczestnicy posiadą umiejętności pisania testów, przypadków testowych oraz zarządzania nimi. Uczestnicy przetestują istniejący system pod kątem wydajności i bezpieczeństwa.

### Materiały wstępne

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

### Zalety szkolenia:

- Narzędzia automatyzacji
- Najlepsze wzorce i praktyki
- Aspekty Behavior Driven Development
- Dwóch trenerów prowadzących szkolenie

## Szczegółowy program:

### 1. Podstawy testowania

- 1.1. Sposoby testowania systemów
- 1.2. Zagadnienie testowalności
- 1.3. Rodzaje testów i przykłady ich wykorzystania
- 1.4. Automatyzacja
- 1.5. Koszty różnych strategii testowania

### 2. Techniki testowania "black-box"

- 2.1. Testy funkcjonalne
  - 2.1.1. Projektowanie przypadków testowych
  - 2.1.2. Automatyczne testowanie "end-to-end" przy użyciu narzędzia Selenium
  - 2.1.3. Zarządzanie przypadkami testowymi
- 2.2. Testy wydajnościowe
  - 2.2.1. Profilowanie w celu wykrycia problemów wydajnościowych
  - 2.2.2. Testowanie obciążeniowe i wydajnościowe przy użyciu Jmeter
- 2.3. Testowanie bezpieczeństwa
  - 2.3.1. Testowanie penetracyjne
  - 2.3.2. Wykrywanie najczęstszych problemów związanych z bezpieczeństwem

### 3. Techniki testowania "white-box"

- 3.1. Testowanie jednostkowe
  - 3.1.1. Szablony testów w JUnit / TestNG (do wyboru)
  - 3.1.2. Tworzenie własnych asercji
  - 3.1.3. Podstawy technik mockowania (na podstawie Mockito)
- 3.2. Testowanie integracyjne

3.2.1. Konfiguracja środowiska programistycznego

3.2.2. Testowanie dostępu do danych

3.2.3. Testowanie aplikacji w architekturze 3-warstwowej

3.2.4. Utrzymanie złożonych testów

#### **4. Analiza statyczna kodu**

4.1. Interpretowanie metryk

4.2. Wykrywanie punktów krytycznych systemu

4.3. Wpływ metryk na testowanie

4.4. Przegląd kodu i identyfikowanie "zapachów"

#### **5. Automatyzacja procesu wytwarzania oprogramowania**

5.1. Automatyzacja budowania projektu przy użyciu Maven / Ant / Gradle (do wyboru)

5.2. Wykorzystanie serwera Continuous Integration

5.3. Optymalizacja złożonego procesu budowy