

Program szkolenia:

Techniki efektywnego testowania kodu dla programistów Java (Spock/JUnit 5)

Informacje:

| | |
|----------------------|--|
| Nazwa: | Techniki efektywnego testowania kodu dla programistów Java (Spock /JUnit 5) |
| Kod: | craft-test-effective |
| Kategoria: | Testowanie automatyczne |
| Odbiorcy: | developerzy |
| Czas trwania: | 3-5 dni |
| Forma: | 50% wykłady / 50% warsztaty |

Podczas szkolenia uczestnicy poznają techniki programowania i testowania ułatwiające codzienną pracę z kodem.

Podczas warsztatów praktycznych uczestnicy posiadą umiejętność pisania i utrzymywania testów oraz pracy techniką Test-Driven Development oraz nowoczesnych narzędzi (Spock Framework lub JUnit 5).

Przedstawiona propozycja programu przy szerokim potraktowaniu wszystkich zagadnień przekracza zaproponowane ramy czasowe. Wcześniej przeprowadzone szkolenia potwierdzają, iż różne elementy szkolenia mogą być mniej lub bardziej istotne dla konkretnej grupy uczestników w zależności od specyfiki pracy, doświadczenia, czy posiadanej już znajomości rozwiązań. Dlatego też po zaakceptowaniu oferty konieczne będzie ustalenie nacisku, jaki ma być położony na poszczególne części szkolenia.

Materiały wstępne

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

Zalety szkolenia:

- Tworzenie skutecznych i łatwych w utrzymaniu testów automatycznych
- Najlepsze wzorce i praktyki (w tym TDD)
- Sensowne testowanie integracyjne (ze Spring Framework)
- Testowanie na poziomie API mikroserwisów (Consumer Driven Contract - CDC)
- Techniki pracy z kodem legacy (opcjonalnie)
- Praca z nowoczesnymi narzędziami do testowania - do wyboru Spock Framework lub JUnit 5

Szczegółowy program:

1. Jednostkowe testowanie kodu

- 1.1. Dlaczego warto automatycznie testować kod
- 1.2. Dobre testy jednostkowe - FIRST
- 1.3. Struktura testów jednostkowych (given-when-then/arrange-act-assert)
- 1.4. Nazewnictwo testów (metod testowych)
- 1.5. Konstrukcje wspierające testowalność kodu (m.in. dziedziczenie -> kompozycja, małe klasy, wstrzykiwanie zależności)
- 1.6. Antywzorce dla testowalnego kodu (m.in. singletony, elementy statyczne, pola i metody final)
- 1.7. Zadania frameworku testowego
- 1.8. Formułowanie naturalnie wyglądających asercji (AssertJ)
- 1.9. Ćwiczenia praktyczne (w trakcie - przeplatane z teorią)

2. Spock Framework - pisanie testów szybciej, zwięźlej i czytelniej (alternatywnie JUnit 5)

- 2.1. Nowa jakość w testowaniu kodu - dlaczego warto poznać Spocka
- 2.2. Konstrukcje i techniki upraszczające kod testowy (Groovy w pigułce na potrzeby pisania testów)
- 2.3. Podział testu na bloki/sekcje
- 2.4. Testy parametryzowane
- 2.5. Testowanie wyjątków
- 2.6. Warunkowe wykonywanie testów
- 2.7. Inicjowanie i sprzątanie w testach
- 2.8. Porównanie trio JUnit/Mockito/AssertJ ze Spockiem
- 2.9. Ćwiczenia praktyczne (w trakcie - przeplatane z teorią)

3. Test Driven Development

- 3.1. Historia TDD

3.2. Podstawowe założenia

3.3. Cykl red-green-refactor

3.4. Przejrzysta struktura testu

3.5. Wybór kolejnych funkcji do zaimplementowania

3.6. Sprawne uruchamianie testów z IDE (przydatne wtyczki, skróty klawiaturowe)

3.7. Ćwiczenia praktyczne z TDD w formie Kata/Coding Dojo

3.8. Korzyści ze stosowania TDD

4. Bezpieczny refaktoring

4.1. Przydatne przekształcenia kodu (refaktoring)

4.2. Automatyczny refaktoring w IDE z użyciem skrótów klawiaturowych (Idea lub Eclipse)

4.3. Ćwiczenia praktyczne

5. Separacja od obiektów współpracujących

5.1. Potrzeba, kiedy i dlaczego warto

5.2. Testowe zastępniki obiektów współpracujących (ang. test doubles)

5.3. Mockowanie ze Spock Framework

5.4. Mockowanie z Mockito (mocki w Spocku mają swoje ograniczenia)

5.5. Ćwiczenia praktyczne (w trakcie - przeplatane z teorią)

6. Testowanie integracyjne aplikacji opartej o Spring Framework

6.1. Dlaczego tylko testy jednostkowe nie wystarczą

6.2. Wsparcie w JUnit/JUnit 5/Spock/TestNG

6.3. Konfiguracja i zarządzanie kontekstem Springa

6.4. Wstrzykiwanie zależności

6.5. Odcinanie zależności w kontekście Springa (wstrzykiwanie zależności testowych, w tym mocków)

6.6. Ręczne tworzenie kontekstu w teście (do testowania specyficznych przypadków)

6.7. Cache'owanie kontekstu między testami (i problemy z tym związane)

6.8. Wydzielanie wspólnej konfiguracji dla wielu testów

6.9. Ćwiczenia praktyczne (w trakcie - przeplatane z teorią)

7. Testowanie aplikacji Spring MVC

7.1. Testowanie kontrolerów w Spring MVC bez kontenera serwletów

7.2. Pełne testy integracyjne z kontenerem serwletów

7.3. Mockowanie wywołań do zewnętrznych serwisów (WireMock)

7.4. Testowanie asynchronicznych wywołań HTTP

7.5. Ćwiczenia praktyczne (w trakcie - przeplatane z teorią)

8. Testowanie z bazą danych

8.1. Rozwiązywanie problemów z zależnościami między testami

8.2. Zarządzanie transakcjami bazodanowymi w testach

8.3. Utrzymywanie zestawu danych testowych

8.4. Testowanie z bazą danych w pamięci

8.5. Kiedy warto mockować warstwę dostępu do bazy danych

8.6. Ćwiczenia praktyczne (w trakcie - przeplatane z teorią)

9. Wybrane tematy z testowania akceptacyjnego

9.1. Wstęp do BDD (Behavior-Driven Development)

9.2. Kryteria akceptacyjne (w Cucumber lub JBehave)

9.3. Techniczne aspekty testowania GUI

9.3.1. Web Driver/Selenium 2 + GEB

9.3.2. Testowanie asynchronicznych elementów strony

9.3.3. Łatwiejsze utrzymywanie testów GUI z Page Object Pattern

9.4. Architektura microserwisów i wprowadzenie do Consumer Driven Contract (Spring Cloud Contract)

9.5. Ćwiczenia praktyczne (w trakcie - przeplatane z teorią)

10. (Opcjonalnie) Praca z odziedziczonym kodem (legacy code)

10.1. Czym jest, dlaczego powstaje i jak temu przeciwdziałać

10.2. Techniki rozbijania zależności

10.3. Techniki testowania

10.4. Techniki refaktoryzacji

10.5. Przydatne narzędzia

10.6. Ćwiczenia praktyczne

11. Zaawansowane zagadnienia

11.1. Techniki testowania asynchronicznych wywołań (Awaitility, Spock)

11.2. Badanie jakości testów z testowaniem mutacyjnym (PIT)

11.3. Techniki przyśpieszania testów

11.4. Uproszczenie niektórych aspektów testowania z Java 8

11.5. Zaawansowany Spock (lub JUnit 5)

11.5.1. Tworzenie własnych rozszerzeń