

Program szkolenia:

Zwinna współpraca programistów i testerów z wykorzystaniem BDD i Spec by Example (JBehave/Spock/SpecFlow)

Informacje:

Nazwa:	Zwinna współpraca programistów i testerów z wykorzystaniem BDD i Spec by Example (JBehave/Spock/SpecFlow)
Kod:	test-coop
Kategoria:	Testowanie i QA
Odbiorcy:	testerzy, developerzy
Czas trwania:	3 dni
Forma:	50% wykłady / 50% warsztaty

Szkolenie ma na celu wprowadzenie konkretnych praktyk umożliwiających płynną współpracę pomiędzy testerami a programistami. Przedstawione techniki mogą być stosowane również w celu umożliwienia współpracy na styku IT i biznesu.

Szkolenie rozpoczyna się od przyjęcia strategii testowania odpowiadając na pytanie: co testujemy (zakres) oraz w jakim celu (rola testu).
Rozpoczynamy od zapoznania z technikami Agile: Behavior Driven Development oraz Specification by Example raz z najlepszymi praktykami tworzenia wykonywalnych specyfikacji.

Drugiego dnia pracujemy w zespołach nad tworzeniem automatycznych testów akceptacyjnych do projektu szkoleniowego. Każdy zespół składa się z testerów i programistów (grających role twórców projektu testowego).
Programiści pracują nad warstwą automatyzacji tworząc Feature Objecty, które zajmują się interakcją z testowanym systemem - zarówno poprzez UI (Selenium) jak i API (REST). Celem programistów jest stworzenie wygodnych "agentów", na bazie których testerzy będą budować wyższe warstwy testów. Testerzy pracują nad warstwą Flow i Spec by Example. Testerzy ćwiczą deklaratywne formułowanie historyjek oraz narzędzia: JBehave/SpecFlow. Narzędzia te pozwalają testerom **programować w języku naturalnym** parametryzowalne, wykonywalne specyfikacje.

Warsztaty są wzbogacone o mechanikę grywalizacji: zespoły rywalizują ze sobą po kątem ilości znalezionych błędów i pokrycia systemu testami.

Materiały wstępne

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

Zalety szkolenia:

- Współpraca programistów i testerów w całym procesie produkcji oprogramowania
- Narzędzia automatyzacji
- Zagadnienia architektury aplikacji wspierającej testowalność kodu
- Najlepsze wzorce i praktyki
- Aspekty Behavior Driven Development i Domain Driven Design

Szczegółowy program:

1. Podstawy testowania

1.1. Strategiczne podejście do testowania - dwuwymiarowe ujęcie: CO x PO CO

1.1.1. Zakres testów - co testujemy

1.1.1.1. Jednostkowe

1.1.1.2. Czym jest Unit w Unit testach?

1.1.1.3. Integracyjne

1.1.1.4. End 2 End komponentowe

1.1.1.5. End 2 End systemowe

1.1.2. Rola testów - po co testujemy

1.1.2.1. Perfekcja działania szczegółów

1.1.2.2. Akceptacyjne - spełnianie wymagań

1.1.2.3. Testy funkcjonalne

1.1.2.4. Testy bezpieczeństwa

1.1.2.5. Testy wydajności

1.1.2.6. Regresyjne - czy wciąż działa

1.2. Automatyzacja procesu testowania

1.3. Wybór strategii testowania w projekcie

1.4. Strategia budowania piramidy testów

1.4.1. Problem: im wyżej w warstwach tym więcej kombinacji możliwości i więcej zależności

1.4.2. Mapowanie piramidy na warstwy systemu

2. Architektura aplikacji otwartej na testowanie

2.1. Rozwarstwienie logiki na aplikacyjną i domenową

2.1.1. Wyłanianie Funktorów i Service Objectów

2.1.1.1. Uniknięcie zależności technicznych

2.1.1.2. Uniknięcie potrzeby Mockowania

2.2. Piramida testów - jak ją interpretować w kontekście warstw

2.2.1. Logika aplikacji - testy End 2 End

2.2.2. Logika domenowa - testy jednostkowe

2.3. Kiedy warto stosować zaślepki (Mock) a kiedy jest to zbędny koszt

3. Behaviour Driven Development

3.1. Zalety bliskiej współpracy z klientem

3.1.1. Rola dostawcy, rola klienta w testach akceptacyjnych

3.2. Tworzenie aplikacji podejściem BDD

3.3. Techniki tworzenia scenariuszy akceptacyjnych

3.3.1. Podejście deklaratywne zamiast imperatywnego

3.3.1.1. Unikanie pisania "skryptów klikania"

3.3.1.2. Raczej: "co ma być" niż "co trzeba zrobić"

3.3.2. Odporność scenariuszy na zmiany systemu

3.3.3. Wady kruchych scenariuszy

3.4. Podejście dwuwarstwowe

3.4.1. Warstwa Flow - User Story

3.4.2. Warstwa Automatykacji interakcji z systemem

3.5. Narzędzia i wzorce

3.5.1. JBehave/Spock/SpecFlow - najlepsze praktyki

3.5.1.1. Integracja z Selenium

3.5.2. Page Object - antywzorzec powodujący powstawanie kruchych testów

3.5.3. Feature Object - bezpieczna alternatywa dla Page Object

3.5.4. Technika ujednociania testów wykonywanych poprzez GUI i Servisy - Agenty

4. Specification by Example

4.1. Wzorce i techniki tworzenie wykonywalnych specyfikacji

4.1.1. Pułapki

4.1.2. Przykłady złych specyfikacji

4.2. Podejście trójwarstwowe

4.2.1. Warstwa Specyfikacji - cele biznesowe

4.2.2. Warstwa Flow - User Story

4.2.3. Warstwa Automatyzacji interakcji z systemem

4.3. Narzędzia automatyzacji

5. Architektura testów akceptacyjnych - podejście trójwarstwowe

5.1. Strategia: metafora "stref zgniotu" z motoryzacji

5.1.1. Tworzenie stabilnych testów, które nie rozpadają się podczas zmian systemu

5.1.2. Integracja BDD i Spec by Example

5.2. Warstwa automatyzacji - API dla testerów

5.2.1. Interakcja z UI poprzez abstrakcję odporną na zmiany ekranów

5.2.2. Wzorce

5.2.2.1. Agent

5.2.2.2. Feature Object

5.2.3. Narzędzia

5.2.3.1. Selenium

5.2.3.2. Remoting

5.3. Warstwa Flow

5.3.1. Scenariusze akceptacyjne BDD

5.3.2. Nastawienie na kroki biznesowe zamiast na interakcje z UI

5.3.3. Narzędzia

5.3.3.1. JBehave/Spock/SpecFlow

5.4. Warstwa Specyfikacji

5.4.1. Nastawienie na cele biznesowe zamiast flow

5.4.2. Narzędzia

5.4.2.1. JBehave/Spock/SpecFlow

5.5. Najlepsze praktyki

5.5.1. Given i Then - unikamy UI

5.5.1.1. Operowanie na API systemu

5.5.1.2. Backdoors na potrzeby testów

5.5.2. When - interakcja z UI

5.6. Archetypowe role aktorów