

Program szkolenia:

Testowanie aplikacji mobilnych na platformie Android - architektura, wzorce, praktyki i narzędzia

Informacje:

Nazwa:	Testowanie aplikacji mobilnych na platformie Android - architektura, wzorce, praktyki i narzędzia
Kod:	craft-test-test
Kategoria:	Testowanie automatyczne
Grupa docelowa:	developerzy
Czas trwania:	4 dni
Forma:	40% wykłady / 60% warsztaty

Szkolenie skupia się na tworzenia testowalnego kodu oraz testów automatycznych aplikacji na platformie Android.

Program kładzie szczególny nacisk na architekturę aplikacji, która jest podatna na testowanie.

W programie znajduje się również wstęp do testowania w ogólności: strategię testowania, techniki i wzorce.

Zalety szkolenia:

- Rzetelne podstawy testowania
- Architektura i wzorce projektowe
- Strategia racjonalnych testów (w tym kiedy testowanie nie ma sensu)

Szczegółowy program:

1. Podstawy testowania

1.1. Rodzaje testów i przykłady ich wykorzystania

1.1.1. Zakres testów

1.1.1.1. Jednostkowe

1.1.1.2. Integracyjne

1.1.1.3. End 2 End

1.1.2. Rola testów

1.1.2.1. Akceptacyjne

1.1.2.2. Regresyjne

1.1.3. Skupienie testów

1.1.3.1. Testy funkcjonalne

1.1.3.2. Testy bezpieczeństwa

1.1.3.3. Testy wydajności

1.2. Strategia budowania piramidy testów

2. Projektowanie przypadków testowych

2.1. Podejście do dokumentowania testów

2.1.1. User Story i Scenariusze akceptacyjne

2.1.2. Wykonywalne specyfikacje - techniki Behavior Driven Development

2.2. Testowanie przypadków granicznych

2.2.1. Nacisk na testy jednostkowe w celu osiągnięcia wysokiego pokrycia testami

2.2.2. Architektura wspierająca wysokie pokrycie testami

2.2.3. Rozwarstwienie logiki na Aplikacyjną i Domenową

2.2.4. Modelowanie logiki przy pomocy Building Blocks z Domain Driven Design

2.3. Powtarzalność testów, wyeliminowanie losowości z testów

2.4. Najlepsze praktyki tworzenia przypadków testowych

3. Strategia testowania architektury aplikacji

3.1. Rozwarstwienie logiki na aplikacyjną i domenową

3.2. Piramida testów - jak ją interpretować w kontekście warstw

3.2.1. Logika aplikacji - testy End 2 End

3.2.2. Logika domenowa - testy jednostkowe

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

4. Najlepsze techniki testowania - projektowanie testów, które można utrzymywać w przyszłości

4.1. Wprowadzenie

4.1.1. Czego nie testować

4.1.2. Struktury przypadków testowych

4.2. Organizacja kodu testowego

4.2.1. Klasa testowa per klasa produkcyjna

4.2.2. Klasa testowa per funkcjonalność

4.2.3. Klasa testowa per setup

4.2.4. Testy parametryzowane

4.3. Przygotowanie stanu (test fixture setup)

4.3.1. Testy wykorzystujące źródło danych (data-driven testing)

4.3.2. Użycie wzorca Asemblera (odmiana Builder Design Pattern)

4.3.3. Wzorce i szablony

4.4. Weryfikacja

4.4.1. Value Object, weryfikacja przez equals

4.4.2. Własne asercje

4.4.3. Weryfikacja przy użyciu specyfikacji (Matcher object)

4.4.4. Upraszczenie asercji z użyciem Assert Object

4.4.5. Poprawna weryfikacja przypadków negatywnych

4.4.6. Wzorce i szablony

4.5. Uprzątniecie po teście (fixture teardown)

4.5.1. Kiedy warto stosować

4.5.2. Manualne

4.5.3. Automatyczne

4.5.4. Wzorce i szablony

4.6. Antywzorce testowania (ponad 20 typowych błędów i pułapek)

4.7. Wykrywanie "Zapachów" złego kodu testowego

4.7.1. Delikatne testy (fragile)

4.7.2. Nieczytelne testy

4.7.3. Wolne testy

4.7.4. Testy niedeterministyczne

5. Testowanie jednostkowe

5.1. Szablony testów w xUnit

5.2. Tworzenie własnych asercji

5.3. Techniki: Mock, Stub, Fake

5.3.1. Dobór technik do potrzeb - czym się kierować

5.3.2. Przykłady implementacji w Mockito

5.4. Mockowanie

5.4.1. Zalety testowania w izolacji

5.4.2. Nagrywanie zachowania

5.4.3. Weryfikacja wywołań

5.4.4. Antywzorce testów wykorzystujące mockowanie

5.5. Testability - podatność kodu na testy

5.5.1. Jak pisać kod, który daje się testować

5.5.2. Najlepsze praktyki: SOLID, GRASP

5.5.3. Wybrane wzorce projektowe, które zwiększają testability: Factory, Strategy, Value Object

5.5.4. Pułapki i typowe błędy

5.5.5. Code smell - "zapachy" nietestowalnego kodu

6. Testowanie akceptacyjne

6.1. Technika User Story

6.2. Tworzenie testów akceptacyjnych na podstawie User Story

7. Behaviour Driven Development

7.1. Zalety bliskiej współpracy z klientem

7.1.1. Rola dostawcy, rola klienta w testach akceptacyjnych

7.2. Tworzenie aplikacji podejściem BDD

7.3. Podejście dwuwarstwowe

7.3.1. Warstwa Flow - User Story

7.3.2. Warstwa Automatykacji interakcji z systemem

7.4. Narzędzia i wzorce

7.4.1. JBehave - najlepsze praktyki

7.4.2. Page Object - modelowanie użytkownika

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

8. Specification by Example

8.1. Wzorce i techniki tworzenie wykonywalnych specyfikacji

8.2. Podejście trójwarstwowe

8.2.1. Warstwa Specyfikacji - cele biznesowe

8.2.2. Warstwa Flow - User Story

8.2.3. Warstwa Automatykacji interakcji z systemem

8.3. Narzędzia automatyzacji

9. Architektura testowalnej aplikacji

9.1. Building Blocks

9.1.1. Komponenty aplikacji

9.1.1.1. Podatność na testowanie

9.1.1.2. Rodzaje testów per rodzaj komponentu

9.1.2. Architektura i cykl życia aplikacji

9.1.3. Zarządzanie stanem aplikacji

9.1.3.1. Zaśleпки stanu

9.2. Użyteczne wzorce

9.2.1. Wzorzec Mediator

9.2.2. Layers

9.2.3. Ports and Adapters

9.3. Tworzenie modularnego kodu

9.3.1. Techniki Inversion of Control

9.3.1.1. Dependency Injection

9.3.1.2. Wstrzykiwanie zaślepek na czas testów

9.3.1.3. Events

9.3.1.4. Separacja i decoupling testowanych modułów

9.3.2. Dynamiczne wstrzykiwanie zależności.

9.3.3. Luźne wiązanie komponentów z użyciem Event Bus.

9.3.4. Dobre praktyki i użyteczne wzorce projektowe na platformie Android.

10. Zagadnienia zaawansowane

10.1. Wprowadzenie do testowania z użyciem Android Testing Framework

10.2. Mockowanie w środowisku Androida

10.2.1. Narzędzia, pułapki, najlepsze praktyki

10.3. Tworzenie szybkich testów z wykorzystaniem biblioteki Robolectric

10.4. Wербalizacja asercji z użyciem FEST oraz rozszerzenia FEST - Android

10.4.1. Pisanie czytelnych, wykonywalnych specyfikacji

10.4.2. Odporność testów na zmiany

10.4.3. Myślenie deklaratywne zamiast imperatywnego

10.5. Testowanie z użyciem biblioteki RoboSpock

10.6. Tworzenie testów funkcjonalnych aplikacji

10.7. Przegląd narzędzi wspomagających testowanie

11. Zagadnienia dodatkowe

11.1. Optymalizacja aplikacji i poszukiwanie błędów z użyciem narzędzi deweloperskich.

11.2. Przegląd narzędzi deweloperskich dostarczonych z Android SDK.