BOTTEGA IT minds

Program szkolenia:

Całościowe podejście do testowania automatycznego dla programistów Java /C#/PHP (TDD, BDD, Spec. by Example, wzorce, narzędzia)

Informacje:

Nazwa:	**Całościowe podejście do testowania automatycznego dla programistów Java/C#/PHP (TDD, BDD, Spec. by Example, wzorce, narzędzia)**
Kod:	**NET-craft-auto**
Kategoria:	Craftsmanship dla programistów .NET
Grupa docelowa:	developerzy
Czas trwania:	3 dni
Forma:	50% wykłady / 50% warsztaty

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

Podczas warsztatów praktycznych uczestnicy posiądą umiejętność pisania i utrzymywania testów oraz pracy techniką Test-Driven Development.

Program szkolenia integruje popularne narzędzia (do wyboru w zależności od języka) w celu wsparcia procesu polegającego na tworzeniu wykonywalnych specyfikacji.
Przez wykonywalne specyfikacje rozumiemy dosłownie tekst Scenariusza Akceptacyjnego (wchodzącego w skład User Story), który można uruchomić aby upewnić się czy jest aktualnie spełniany.
Uruchomienie scenariusza może nastąpić na dowolnym poziomie architektury systemu: poprzez GUI lub warstwę API.

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 Spec by Example

BO·TT·EGA IT minds

Szczegółowy program:

1. Podstawy testowania

- 1.1. Strategia racjonalnego testowania: CO x PO CO x JAK
  - 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. Wybór strategii testowania w projekcie
- 1.3. 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. 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. Architektura aplikacji otwartej na testowanie

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 - ogólne 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

4.3.3.1. Object Mother

4.3.3.2. Assembler (aka Builder) - sensowne i mniej sensowne podejścia

4.4. Weryfikacja

4.4.1. Value Object, weryfikacja przez equals

BOTTEGA IT minds

4.4.2. Własne asercje

4.4.3. Weryfikacja przy użyciu specyfikacji (Matcher object)

4.4.4. Upraszczanie asercji z użyciem Assert Object

4.4.5. Poprawna weryfikacja przypadków negatywnych

4.4.6. Wzorce i szablony

4.4.6.1. Assert Object

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

4.8. Techniki organizacji kodu testowego - standardy, wzorce i najlepsze praktyki

4.8.1. Wsparcie refaktoryzacji ze strony IDE

4.8.2. Techniki refaktoryzacji kodu testowego

4.8.3. Efektywne sposoby utrzymania dużej liczby testów

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

5.4. 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. Co sprawia że czas poświęcony na napisanie testu zwróci się

5.6. Testability - podatność kodu na testy

5.6.1. Jak pisać kod, który daje się testować

5.6.2. Najlepsze praktyki: SOLID, GRASP

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

5.6.4. Pułapki i typowe błędy

5.6.5. Code smell - "zapachy" nietestowalnego kodu

6. Test Driven Development

6.1. Cykl czerwony-zielony-refaktoring

6.2. Wady i zalety TDD

6.3. Kiedy warto, dla kogo jest TDD

6.4. Ewolucyjny sposób rozwój kodu

6.5. Podstawowe techniki refactoringu

7. Testowanie akceptacyjne

7.1. Technika User Story

7.2. Tworzenie testów akceptacyjnych na podstawie User Story

7.3. Zyski i koszty różnych technik testowania akceptacyjnego

7.4. Efektywne narzędzia wspierające:

7.4.1. Testowanie poprzez warstwę GUI

7.4.2. Testowanie poprzez warstwę serwisów

7.4.3. Przygotowywanie stanu początkowego

7.5. Podejścia

7.5.1. Testowanie przez GUI

7.5.1.1. Selenium

7.5.2. Testowanie przez warstwę serwisów (API systemu)

7.5.2.1. Zdalne wywołanie obiektów (Spring Remoting/Remote EJB/Seam - do wyboru)

7.5.3. Unifikacja podejść dzięki Agentom BDD

8. Wstęp do Behaviour Driven Development

8.1. Zalety bliskiej współpracy z klientem

8.1.1. Rola dostawcy, rola klienta w testach akcpetacyjneych

8.2. Tworzenie aplikacji podejściem BDD

8.3. Podejście dwuwarstwowe

8.3.1. Warstwa Flow - User Story

8.3.2. Warstwa Automatyzacji interakcji z systemem

8.4. Narzędzia i wzorce

8.4.1. JBehave/Spock/SpecFlow - najlepsze praktyki

8.4.1.1. Integracja z Selenieum

8.4.2. Page Object - modelowanie użytkownika

8.5. Technika ujednolicania testów wykonywanych poprzez GUI i Servisy - Agenty

9. Wstęp do Specification by Example

9.1. Wzorce i techniki tworzenie wykonywalnych specyfikacji

9.2. Podejście trójwarstwowe

9.2.1. Warstwa Specyfikacji - cele biznesowe

BOTTEGA IT minds

9.2.2. Warstwa Flow - User Story

9.2.3. Warstwa Automatyzacji interakcji z systemem

9.3. Narzędzia automatyzacji

10. Kompleksowy proces - podsumowanie szkolenia

10.1. Korzystanie z user story, BDD i TDD w codziennej pracy

10.2. Automatyzacja - korzystanie z serwera Continous Integration

10.3. Narzędzia (do wyboru)

10.3.1. Java: JUnit/TestNG, JBehave/Spock, Selenium

10.3.2. .NET: nUnit, NDbUnit, nBehave, nSubstitute, nCover, Selenium

10.3.3. PHP: Behat, phpunit, mockery, prophecy

10.4. Wykorzystanie najlepszych praktyk, wskazanie typowych błędów i pułapek