BO·TT·EGA
IT minds

Program szkolenia:

Testowanie z Pytest

Informacje:

Nazwa:	**Testowanie z Pytest**
Kod:	**craft-test-pytest**
Kategoria:	Testowanie automatyczne
Odbiorcy:	testerzy, architekci, developerzy
Czas trwania:	3 dni
Forma:	20% wykłady, 80% warsztaty

Przewodnik po technikach testowania wspomagających pracę programisty i testera automatyzującego. Szkolenie łączy poznawanie dobrych praktyk testowania z opanowywaniem tajników pytesta. Program i ćwiczenia oparte o studium przypadku.

Dzień 1: pisanie scenariuszy akceptacyjnych, podstawy testów akceptacyjnych, jednostkowych i zaślepianie

Dzień 2: “Dobre praktyki” (weryfikacja, testowanie interfejsu zamiast implementacji, nazewnictwo testów)

Dzień 3: Opanowywanie narzędzia pytest oraz rozwiązywanie problemów z testami

Zalety szkolenia:

- Wzorce i antywzorce
- Strategia testowania
- Testowalny kod

BOTTEGA IT minds

Szczegółowy program:

1. Piszemy testy z myślą o celu

- 1.1. Wzorce
  - 1.1.1. Sterowanie implementacji testami
  - 1.1.2. Ochrona przed regresją
  - 1.1.3. Stabilne dostarczanie oprogramowania
  - 1.1.4. Wspomaganie programisty, nie spowalnianie
- 1.2. Antywzorce
  - 1.2.1. Pisanie testów tylko po to, by spełnić cel pokrycia testsami
  - 1.2.2. Pisanie testów, które sprawdzają implementację zamiast zachowania
  - 1.2.3. Pisanie testów zawsze na końcu pracy nad zadaniem

2. Struktura testu

- 2.1. Pisanie testów tak, by mogły być uruchomione przez pytest
- 2.2. Kroki
  - 2.2.1. Arrange
  - 2.2.2. Act
  - 2.2.3. Assert
  - 2.2.4. Kiedy łamiemy zasady?

3. Testy jednostkowe

- 3.1. Czym jest jednostka?
- 3.2. Techniki zaślepiania
  - 3.2.1. Dummy
  - 3.2.2. Fake
  - 3.2.3. Stub

BOTTEGA IT minds

- 3.2.4. Mock
- 3.2.5. Dedykowane biblioteki dla określonych przypadków (np. responses, vcr.py, moto)
- 3.3. Co i kiedy zaślepiać?
- 3.4. Mock z biblioteki standardowej - jak go poprawnie wykorzystywać?
- 3.5. Inne narzędzia do pisania obiektow - dublerów - mockito

4. Testy akceptacyjne - sprawdzające spełnienie wymagań biznesowych

- 4.1. Pisanie scenariuszy akceptacyjnych
- 4.2. Wybór poziomu testowania akceptacyjnego - wady i zalety
  - 4.2.1. UI
  - 4.2.2. API
  - 4.2.3. Service layer

5. Dobre praktyki testowania

- 5.1. Sprawdzanie obserwowalnych wyników działania
  - 5.1.1. Rodzaje obserwowalnych wyników
    - 5.1.1.1. Zwracany rezultat
    - 5.1.1.2. Rzucany wyjątek lub jego brak
    - 5.1.1.3. Stan
    - 5.1.1.4. Interakcje
  - 5.1.2. Zastosowanie w różnych rodzajach testów
    - 5.1.2.1. testy jednostkowe
    - 5.1.2.2. testy akceptacyjne
- 5.2. Testy implementacji kontra testy interfejsu
  - 5.2.1. Zapach testów - powielona implementacja w teście i testowanym kodzie
  - 5.2.2. Weryfikujemy zachowania wybranej jednostki
- 5.3. Nazewnictwo testów

BOTTEGA IT minds

- 5.3.1. Konwencje i dlaczego nie działają
- 5.3.2. Używanie zdań oznajmujących
- 5.3.3. Pozbywanie się szumu takiego jak “should”, “must”
- 5.3.4. Nazywanie testu jako okazja do przemyślenia czy to dobry test

6. Pokrycie kodu testami

- 6.1. Po co mierzyć pokrycie kodu testami?
- 6.2. Rodzaje pokrycia kodu testami
  - 6.2.1. Line coverage
  - 6.2.2. Branch coverage
  - 6.2.3. Conditional coverage
- 6.3. pytest-cov i konfiguracja
- 6.4. 100% pokrycia jako cel
  - 6.4.1. Jak dojść do 100% pokrycia testami i nie napracować się zbytnio?
- 6.5. Techniki testowania wspomagające pokrycie
  - 6.5.1. Testy mutacyjne
  - 6.5.2. Property-based testing

7. Strategia testowania

- 7.1. Piramida testów według Mike Cohn’a
  - 7.1.1. Testy jednostkowe
  - 7.1.2. Testy serwisów
  - 7.1.3. Testy end-to-end
  - 7.1.4. Ocena wzorcowej piramidy
- 7.2. Budowanie własnej strategii
  - 7.2.1. Gdy testy wyższego poziomu wystarczają
- 7.3. Dobieranie strategii a architektura projektu

BOTTEGA IT minds

- 7.3.1. Wygląd i możliwości pisania testów jest pochodną sposobu napisania kodu
- 7.3.2. Typowy CRUD w Django
- 7.3.3. czysta architektura / porty i adaptery

8. Testowalny kod

- 8.1. Stosowanie luźnego powiązania
- 8.2. Odwracanie zależności
- 8.3. Wydzielanie stabilnych interfejsów

9. Testy akceptacyjne - rozwinięcie

- 9.1. Własny DSL
  - 9.1.1. Przykład z giełdą i order bookiem
- 9.2. BDD z behave lub pytest-bdd
- 9.3. Taktyki zaślepiania w testach akceptacyjnych
  - 9.3.1. Znalezienie stabilnych punktów zaślepiania

10. Testy kontraktowe

- 10.1. Narzędzie do lepszej współpracy między zespołami budującymi rozproszony system
- 10.2. Consumer-Driven Contract Testing
- 10.3. Producer-Driven Contract Testing
- 10.4. Przegląd narzędzi dostępnych w Pythonie
  - 10.4.1. Pact

11. System fikstur w pytest

- 11.1. Przenoszenie części lub całości kroku arrange do fikstur
- 11.2. Sprzątanie w fiksturze po teście dzięki instrukcji yield
- 11.3. Zakresy fikstur
- 11.4. Parametryzowane fikstury

BOTTEGA IT minds

12. Taktyki przyspieszania testów

- 12.1. Metryki
  - 12.1.1. Czas od uruchomienia do przejścia pierwszego testu
  - 12.1.2. Czas od uruchomienia do zakończenia całego zestawu testów
  - 12.1.3. Czasy trwania poszczególnych testów
- 12.2. Namierzanie wolnych testów
  - 12.2.1. Szybka powtórka z teorii ograniczeń
  - 12.2.2. Szukamy wąskich gardeł i je optymalizujemy
- 12.3. Przyspieszanie wolnych testów
  - 12.3.1. Profilowanie
  - 12.3.2. Używanie stałych fikstur
- 12.4. Szybkie i proste tweaki

13. Taktyki poprawiania niestabilnych testów

- 13.1. Fikstura kontrolna
- 13.2. Naprawianie izolacji testów
  - 13.2.1. Transakcje
  - 13.2.2. Logiczna separacja
- 13.3. Testy niestabilne z definicji
  - 13.3.1. Odseparowanie od głównego zestawu przy pomocy markerów
  - 13.3.2. Wbudowujemy mechanizmy ponawiania w kod produkcyjny
  - 13.3.3. Ponawiamy przy pomocy pytest-rerun