BO·TT·EGA
IT minds

Program szkolenia:

Tworzenie czytelnego, testowalnego i bardziej niezawodnego kodu

Informacje:

Nazwa:	**Tworzenie czytelnego, testowalnego i bardziej niezawodnego kodu**
Kod:	**Craft-practices-code**
Kategoria:	Craftsmanship
Grupa docelowa:	developerzy
Czas trwania:	3-5 dni
Forma:	50% wykłady / 50% warsztaty

Szkolenie przeznaczone dla programistów i projektantów pragnących poszerzyć swe kompetencje w zakresie profesjonalnych technik zwiększających jakość kodu i projektu. Zdobyta wiedza przekłada się w praktyczny sposób na produktywność mierzoną w szerszej perspektywie czasu.

Zalety szkolenia:

- Sprawdzone techniki SOLID, GRASP i DDD
- Osadzenie technik w architekturze aplikacji i systemu
- Realne przykłady

BO·TT·EGA
IT minds

Szczegółowy program:

1. Część 1. Czytelny kod i testowanie jednostkowe

2. Czytelniejszy i łatwiejszy w utrzymaniu kod

- 2.1. Agile vs. Software Craftsmanship
- 2.2. tworzenie kodu vs. tworzenie dobrego kodu
- 2.3. złożoność oprogramowania
- 2.4. nazewnictwo
- 2.5. prostota konstrukcji (m.in. małe klasy i metody, zasada pojedynczej odpowiedzialności)
- 2.6. obsługa błędów
- 2.7. komentarze
- 2.8. pomocne zasady projektowe (m.in. SOLID, KISS, DRY)
- 2.9. ćwiczenie praktyczne ze zwiększania czytelności kodu

3. Jednostkowe testowanie kodu

- 3.1. dlaczego warto automatycznie testować kod
- 3.2. dobre testy jednostkowe - FIRST
- 3.3. struktura testów jednostkowych (given-when-then/arrange-act-assert)
- 3.4. nazywanie testów (metod testowych)
- 3.5. konstrukcje wspierające testowalność kodu (m.in. dziedziczenie -> kompozycja, małe klasy, wstrzykiwanie zależności)
- 3.6. antywzorce dla testowalnego kodu (m.in. singletony, elementy statyczne, pola i metody final)
- 3.7. krótkie wprowadzenie frameworka do tworzenia testów (TestNG lub JUnit)
- 3.8. formułowanie naturalnie wyglądających asercji (AssertJ)
- 3.9. różne podejścia do testowania wyjątków (try..catch, ExpectedExceptions, catch-exception)
- 3.10. kata na prostym przykładzie - jest kod i chcemy go przetestować

BO·TT·EGA
IT minds

4. Test Driven Development

- 4.1. teoretyczny wstęp do TDD (o testowalności samej w sobie jest już wcześniej)
- 4.2. Arrange-Act-Assert/Given-When-Then
- 4.3. wybór kolejnych funkcji do zaimplementowania
- 4.4. sprawne uruchamianie testów z IDE (przydatne wtyczki, skróty klawiaturowe)
- 4.5. programowanie w parach
- 4.6. kata w TDD

5. Separacja od obiektów współpracujących

- 5.1. dublerzy testowi (test doubles) i mockowanie - krótkie wprowadzenie + dlaczego warto i kiedy
- 5.2. krótki workshop z Mockito (całościowo lub tylko bardziej zaawansowane techniki)
- 5.3. ćwiczenia praktyczne z mockowania

6. Bezpieczny refaktoring

- 6.1. przydatne przekształcenia kodu (refaktoring)
- 6.2. automatyczny refaktoring w IDE z użyciem skrótów klawiaturowych (Idea lub Eclipse)
- 6.3. ćwiczenia praktyczne

7. Część 2. Nie tylko testy jednostkowe (do wyboru w zależności od dostępnego czasu i potrzeb)

8. Testowanie integracyjne

- 8.1. dlaczego tylko testy jednostkowe nie wystarczą
- 8.2. używanie kontekstu IoC (Spring) w testach
- 8.3. zarządzanie bazą danych w testach
- 8.4. transakcyjność w testach
- 8.5. separowanie zależności (mockowanie) w kontekście IoC
- 8.6. testowanie kontrolerów w Spring MVC bez kontenera serwletów

9. Testowanie akceptacyjne (tylko wstęp - z tego tematu całe szkolenie można zrobić)

BOTTEGA IT minds

- 9.1. BDD (Behavior-Driven Development)
- 9.2. kryteria akceptacyjne w Cucumber lub JBehave
- 9.3. testowanie GUI (Web Driver/Selenium 2 + GEB)
- 9.4. utrzymywalność testów GUI z Page Object Pattern

10. Praca z odziedziczonym kodem (legacy code)

- 10.1. czym jest, dlaczego powstaje i jak temu przeciwdziałać
- 10.2. techniki łamania zależności, refaktoryzacji i testowania
- 10.3. przydatne narzędzia (PowerMock, PIT)
- 10.4. ćwiczenia praktyczne

11. Ciekawe rozszerzenia

- 11.1. Spock Framework - pisanie testów szybciej, zwięźlej i czytelniej
- 11.2. podział testów na grupy i ich selektywne uruchamianie
- 11.3. badanie jakości testów z testowaniem mutacyjnym (PIT)
- 11.4. testowanie asynchronicznych wywołań (Awaitility)