Notatki - Testowanie - Zasady

Spis treści

Zasady pisania dobrych testów jednostkowych

Poniżej chciałbym umieścić klika reguł, których powinniśmy się trzymać pisząc testy jednostkowe. Oczywiście jest to kwestia filozoficzna, więc prezentuję tutaj moje zdanie.

Nie używaj infrastruktury

Jeżeli będziemy pisali testy jednostkowe i natkniemy się na klasę, która korzysta z bazy danych to zaślepmy ją. W sensie zastąpmy ją zaślepką. Testy jednostkowe mają testować jednostkę, a nie jednostkę i bazę danych. O zaślepaniu (**mockowaniu**) dowiemy się później.

Dodam również, że testy jednostkowe mają być szybkie, a jeżeli zaczniemy uwzględniać bazę danych w testach, to ani nie będą one jednostkowe, ani nie będą szybkie bo dojdzie nam cała komunikacja z bazą danych. Nie będzie to widoczne przy jednym teście, ale przy kilkuset już tak.

Testy jednostkowe powinny być wyizolowane

Trochę wiąże się to z poprzednim punktem. Jednostkowy, oznacza, że testujemy tę konkretną klasę. Czyli jeżeli nasza klasa odwołuje się do innych klas, to te inne klasy powinny zostać zaślepione (**mock**). Oczywiście w praktyce często jest tak, że piszemy testy jednostkowe, które wykorzystują kilka klas ☺ - czyli w sumie są to takie testy integracyjne, ale nazywane są nadal jednostkowymi. Wiadomo, teoria i praktyka.

Testy jednostkowe powinny być małe

Test powinien testować tylko jedną funkcję. Wpływa to później na strukturę klasy testowej, bo jeżeli dobrze opiszemy testy, to będzie wiadomo co konkretnie jest testowane, która metoda lub wycinek kodu. Nie będziemy mieli jednej metody testującej pół naszej aplikacji. Oczywiście cały czas mówię o testach jednostkowych. W przypadku testów integracyjnych, czasem jest naszym założeniem, żeby przetestować pół naszej aplikacji, by zobaczyć jak klasy/moduły integrują się ze sobą.

Testujmy nie tylko przypadki pozytywne

To jest pułapka w którą często wpadają programiści. Jeżeli będziemy testować tylko przypadki pozytywne (innymi słowy nie będziemy uwzględniać żadnych **corner case'ów**), może się okazać, że przy wprowadzaniu zmian, nasza ścieżka pozytywna będzie nadal super działać, ale kod popsuje nam sytuacje brzegowe. Takie które zdarzają się rzadko i są trudne do znalezienia. Czyli nie piszmy testów tylko wprowadzając dane pozytywne i oczekując, że zostanie wszystko dobrze policzone. Stwórzmy też test, który przyjmuje dane dziwne/złe/niewłaściwe i określmy jak wtedy aplikacja ma się zachować.

Testy powinny być niezależne od siebie

Dobrą praktyką jest pisanie testów w taki sposób żeby były one całkowicie niezależne od siebie. Jeżeli przestawimy losowo kolejność wykonania testów, mają one nadal działać prawidłowo. Jeżeli zaczniemy uzależniać testy od siebie (tzn, test kolejny oczekuje, że poprzedni coś zmieni), może to doprowadzić do bardzo dziwnych i nieprzewidywalnych błędów.

Testy powinny być deterministyczne

Zagadnienie to nie zostało omówione na nagraniu, natomiast odniesienie do tego zagadnienia znajdziesz w nagraniach w części praktycznej.

Bardzo dużym problemem w praktyce okazuje się sytuacja gdy testy zachowują się w sposób niedeterministyczny. Czyli jeżeli uruchomimy test i na zmianę otrzymujemy wynik `passed` i `failed`. Takie błędy są często bardzo trudne w diagnozie, więc musimy zawsze się upewnić, czy testy zachowują się w sposób deterministyczny, czyli zawsze tak samo. Inaczej takie zjawisko określa się stwierdzeniem **migotanie testów**. Jeżeli test na przemian przechodzi bądź nie, określa się to stwierdzeniem, że **test migocze**.