Notatki - Testowanie - Intro

Spis treści

Czym jest testowanie? Po co się testuje?

Testy to nic innego jak oprogramowanie (można to rozumieć jako program testujący nasz program), który wywołuje jakiś fragment programu i oczekuje, że zachowa się on w określony sposób, przy podaniu określonych parametrów wejściowych. Testy istnieją po to, aby móc automatycznie zapewnić nas, że napisany przez nas program zachowuje się zgodnie z oczekiwaniami.

Testy są zazwyczaj wykonywane na etapie budowania paczki z aplikacją. Dzięki temu możemy dosyć szybko dowiedzieć się, że albo napisany przez nas kod nie spełnia oczekiwań, albo napisaliśmy kod, który popsuł wcześniej działającą funkcjonalność. W takim przypadku istniejące już testy zaczną nam sygnalizować, że coś przestało działać. W praktyce jest to częsta sytuacja i automatyczna możliwość weryfikacji czy nic nie popsuliśmy jest bardzo przydatna.

Pokrycie kodu testami oznacza, że kod naszej aplikacji został sprawdzony przez testy, które zostały wykonane. Jeżeli powiemy, że chcemy aby nasza aplikacja miała pokrycie (**code coverage**) na poziomie 80%, oznacza to, że 80% linijek napisanego przez nas kodu zostało uruchomionych/wykonanych podczas uruchamiania testów naszej aplikacji.

W praktyce wysokie pokrycie kodu testami jest bardzo przydatne, bo dzięki temu nie boimy się wprowadzać zmian w aplikacji. Nie boimy się bo wiemy, że coś nad nami czuwa i automatycznie sprawdzi czy nic nie popsuliśmy swoimi zmianami 😉. Oczywiście jest to sytuacja idealna i w praktyce nie zawsze spotkamy wysokie pokrycie kodu testami.

Panuje przekonanie, że testy powinny być pisane dla krytycznych i złożonych funkcjonalności. Piszę o tym dlatego, że przyjmuje się, że można zignorować pisanie testów dla getterów i setterów, a przecież nadal są to linijki kodu. Dlatego wspomniałem wcześniej o pokryciu kodu testami na poziomie 80% (który w praktyce i tak jest bardzo wysokim wynikiem). Oznacza to, że zakładamy jakiś bufor na fragmenty kodu, które nie są wywoływane w żadnym z testów.

Jeżeli podchodzimy do aplikacji, która nie ma napisanych testów, dobrze jest zacząć od krytycznych funkcjonalności, czyli takich, które generują dla nas "największy biznes" lub są krytyczne z punktu widzenia użytkownika, np. proces zakupu produktu. Przykładowo dlatego, że np. więcej osób kupuje w sklepie niż reklamuje zakupiony towar.

Rodzaje testów

Jeżeli będziemy rozmawiać o testach, w pewnym momencie zaczniemy też wyróżniać rodzaje testów. W tym materiale skupimy się tylko na najprostszych, natomiast rodzajów testów jest dużo. Wymieńmy

parę z nich:

- testy **wydajnościowe** (performance tests) - ich podstawowym celem jest zapewnienie, że testowany kod będzie działał wystarczająco szybko nawet jak będziemy go uruchamiać pod bardzo dużym obciążeniem, np. aplikacja zachowa się inaczej gdy będzie z niej korzystało 1_000 a 1_000_000 użytkowników.
- testy **penetracyjne** (penetration tests) - celowe przeprowadzenie cyberataku na naszą aplikację celem sprawdzenia stopnia zabezpieczeń aplikacji.
- testy **jednostkowe** (unit tests) - testy pisane przez developerów, które wykonują określony fragment kodu z określonymi parametrami wejściowymi i sprawdzają czy zachowanie końcowe jest zgodne z oczekiwanym. Testy jednostkowe mają to do siebie, że sprawdzają bardzo małe fragmenty kodu, pojedyncze klasy lub metody. Jeżeli dany fragment kodu wykonuje operację, która jest zależna od interakcji z systemem zewnętrznym (np. baza danych, albo inna aplikacja), wtedy taki system jest zastępowany zaślepką (**mock** - powiemy o tym później). Zaślepka taka może nam odpowiedzieć w określony sposób, pozytywnie lub negatywnie, konfigurujemy to sami. Robimy to w tym celu, żeby osiągnąć swojego rodzaju izolację. Kod, który testujemy ma być niezależny od niczego z zewnątrz. Jeżeli natomiast nasze testy uwzględniają interakcję z systemami zewnętrznymi zaczynamy wtedy mówić o testach integracyjnych.
- testy **integracyjne** (integration tests) - mają sprawdzać zachowanie komponentów, na styku których następuje integracja. Testujemy wtedy komponenty, które integrują się ze sobą i są testowane w grupie. Z racji szerokości tej definicji, w życiu codziennym najprawdopodobniej spotkasz się ze stwierdzeniem "testy integracyjne" w poniższych sytuacjach:
  - testy, w których testujemy jednocześnie kilka/kilkanaście klas. Jeżeli przyjmiemy, że testy jednostkowe testują tylko pojedynczą metodę w klasie, to w przypadku, gdy testujemy kod, który integruje ze sobą kilka/kilkanaście naszych klas lub modułów, to mówimy o testach integracyjnych,
  - testy, w których zaślepiamy komunikację z systemami zewnętrznymi, ale nadal testujemy kilka/kilkanaście naszych klas lub modułów. Testy integracyjne mają sprawdzić integrację między komponentami, więc sprawdzamy integrację między naszymi komponentami, ale zaślepiamy systemy zewnętrzne,
  - testy, w których faktycznie komunikujemy się z systemami zewnętrznymi, np. z bazą danych, albo innymi systemami. Może być to np. system, w którym składamy zamówienia. Wtedy jedynie zostaje kwestia utrzymania porządku z danymi w takim systemie, np. po wykonaniu takiego testu możemy wtedy takie dane czyścić.

Rodzajów testów jest więcej, ale na ten moment chcę się skupić tylko na **testach jednostkowych**. Wspominam też cały czas o tych zaślepkach, ale w ramach tego warsztatu nie będziemy poruszać tej tematyki. Przejdziemy do niej w dalszych częściach materiału.