1. Weryfikacja wymagań, wykrycie defektów i awarii, sprawdzenie czy przedmiot testów jest kompletny
2. Testowanie ujawnia awarie będące wynikiem defektów a debugowania to znajdowanie tych defektów i ich usuwanie
3. Ze względu na typ, poziom i sposób przeprowadzania testu
4. Sposób przeprowadzania testu: testy automatyczne i manualne

Poziom: testy jednostkowe, testy integracyjne, testy E2E, Testy akceptacyjne

Typ: strukturalne, funkcjonalne, regresywne, potwierdzające, retesty, niefunkcjonalne, obciążeniowe, wydajnościowe

|  |  |  |
| --- | --- | --- |
| Typ | Poziom | Sposób przeprowadzania testu |
| strukturalne | jednostkowe | automatyczne |
| funkcjonalne | integracyjne | manualne |
| regresywne | E2E |  |
| potwierdzające | akceptacyjne |  |
| obciążeniowe |  |  |
| wydajnościowe |  |  |

1. Testy czarnoskrzynkowe, testy białoskrzynkowe, testy oparte na doświadczeniu
2. Test driven Development. Nie można zacząć pisać kodu produkcyjnego, zanim nie stworzymy testu jednostkowego, Pojedynczy test jednostkowy powinien sprawdzać co najwyżej jeden case programu.
3. Wytworzenie testu jednostkowego (w tzw. fazie czerwonej) -- Implementacja w kodzie danego feature'a (faza zielona)-- Refaktor kodu (faza niebieska).
4. FIRST

F – fast - szybkie

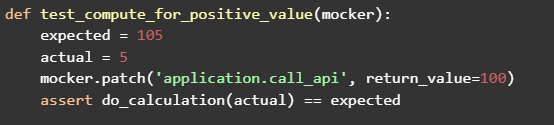
I – independent – niezależne

R – repeatable – powtarzalne

S – Self-Validating – samokontrolujące

T – timely – o czasie

1. Mock to obiekt, którego uzywa się zamiast rzeczywistej implementacji w trakcie testów jednostkowych. Mockowanie to tworzenie makiety danej funkcjonalności. Wykorzystuje się je aby nie było konieczne wykonywanie całej operacji
2. pytest-mock
3. d



1. import Klasa, get\_data

def test\_mockin\_method(mocker):

expected = ‘data loaded’

def mock\_load\_data(self):

return ‘data loaded’

mocker.patch(‘Klasa.load\_data’, mock\_load\_data)

actual = get\_data()

assert expected == actual