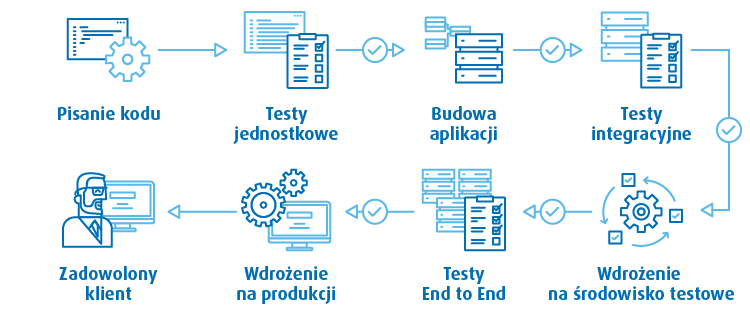
Biblioteki python do testów: Unitest, pytest, nose, doctest.

Metody testowania: manualne, automatyczne

## poziomy testów oprogramowania

* **Testy jednostkowe (unit testy)**

Wykonywane przez programistów / developerów, testowaniu pojedynczych fragmentów oprogramowania (klasa, metoda, funkcja). Szybkie, tanie, na odizolowanym fragmencie kodu

* **Testy integracyjne**

Wykonywane przez programistów / developerów po testach jednostkowych.

Sprawdzanie poprawność integracji pomiędzy danymi komponentami.

Wykrywają błędy w komunikacji między współpracującymi elementami systemu (np. aplikacja–db, klasa-klasa) Bardziej zaawansowane niż jednostkowe, bardziej czaso i pracochłonne

* **Testy systemowe (End-to-End)**

Wykonywane przez testerów w finalnej fazie projektu na zintegrowanym systemie. Sprawdzają działanie aplikacji jako całości. Powolne, trudno je w pełni zautomatyzować, wymagają częstych modyfikacji, są drogie i trudne w utrzymaniu.

* **Testy akceptacyjne**

Wykonywane przez klienta i użytk. w celu upewnienia się, że aplikacja spełnia oczekiwania.

**Np. UAT** – **User Acceptance Tests,** które wykrywają wszelkie niezgodności ze specyfikacją biznesową, oraz testy akceptacji operacyjnej **OAT (Operational Acceptance Tests)**sprawdzające na przykład zachowanie podczas awarii.

**typy testów oprogramowania**

* **Testy funkcjonalne (czarnoskrzynkowe)**

**Definiują one czynności wykonywane przez oprogramowanie. Testy funkcjonalne są zwykle opisane w specyfikacji wymagań jako przypadki użycia, a do ich wykonania nie jest konieczna znajomość wnętrza systemu. Definiuje się je jako zewnętrzne zachowanie systemu, traktując je jak czarną skrzynkę. Osoba przeprowadzająca testy jest w stanie przetestować aplikację względem wymagań funkcjonalnych zwykle bez posiadania wiedzy teoretycznej.**

* **Testy niefunkcjonalne (jakościowe – jak działa system)**

**Testowanie niefunkcjonalne pozwala m.in. sprawdzić:  **użyteczność** – czyli jak przydatna i intuicyjna jest aplikacja, czy jest estetyczna. , czy doświadczenia użytkownika są pozytywne, a często także – czy aplikacja jest dostosowana do potrzeb osób z niepełnosprawnościami,  **bezpieczeństwo**** – czy nikt nie wykradnie naszych danych z systemu oraz na ile jest on zabezpieczony przed atakami osób trzecich. Przykładem testy penetracyjne, które symulują rzeczywisty atak, a także pozwalają dokonać oceny ryzyka – analizuje się wówczas i identyfikuje największe zagrożenia bezpieczeństwa,  ****niezawodność**** – czy system jest w stanie działać poprawnie przez określony czas i w określonych warunkach. W tym przypadku następuje również weryfikacja tolerowania usterek przez oprogramowanie. Sprawdzamy zdolność systemu do kontynuowania działania podczas wystąpienia awarii oraz zdolność odtworzenia – odzyskania danych po awarii mierzone w czasie i stratach,  ****wydajność**** – jak dużo system wytrzyma. Czy będzie działał stabilnie w warunkach różnego obciążenia, jak szybko będzie reagować na polecenia oraz w jakim czasie będą wykonywały się obliczenia i jak szybka będzie komunikacja z innymi systemami. Testy wydajności opierają się zwykle na określeniu profili produkcyjnych na podstawie przewidywań lub doświadczeń z podobnymi systemami,  ****utrzymywalność****– mierzy stopień możliwych modyfikacji systemu w przyszłości, celem minimalizacji przestojów w działaniu aplikacji niezbędnych do jej utrzymania oraz minimalizacji kosztów utrzymania. Istotne będzie tutaj utrzymanie przejrzystości kodu oraz prowadzenie dokumentacji,  ****przenaszalność****– określa poziom skomplikowania przenoszenia aplikacji pomiędzy różnymi środowiskami. Bada zdolności adaptacyjne systemu – np. czy istnieje możliwość zainstalowania go na wszystkich systemach docelowych,  ****kompatybilność****– sprawdza możliwość współistnienia różnych programów na tym samym środowisku oraz możliwość działania przy różnych parametrach. Przykładowo, czy oprogramowanie działa na różnych urządzeniach, systemach czy przeglądarkach internetowych tak samo.

* **Testy strukturalne (białoskrzynkowe)**

**Mogą być wykonane na wszystkich poziomach testowych. **Najprościej rzecz ujmując – to testowanie kodu.** Wykonuje się je jako wsparcie pomiarów dokładności poprzez ocenę pokrycia kodu, która jest mierzona wykonaniem kodu przy pomocy szeregu testów, wyrażone procentowo dla pokrytych elementów.**

* **Testy związane ze zmianami (regresji, potwierdzające)**

**wykonane na wszystkich poziomach testowych.**

****Testy potwierdzające** – to ponowne testy po naprawie defektu, wyk. celem upewnienia się, że defekt został usunięty.**

**Testowanie regresywne** – ma na celu sprawdzenie, czy podczas naprawy lub zmian w aplikacji nie powstały inne defekty w danym module lub w niezmienionej części oprogramowania.

**Testy smoke (dymne) -** sprawdzają, czy aplikację da się uruchomić i czy jej podstawowe funkcje działają

**Testy sanity (kondycji) -** weryfikują nową funkcjonalność bez sprawdzania, czy reszta działa.

**Testy smoke i sanity przy zmianach czy naprawie oprogramowania pozwalają uniknąć marnowania czasu**