ZESPÓŁ SZKÓŁ ELEKTRYCZNO- MECHANICZNYCH

Pracownia testowania i dokumentowania aplikacji

Testowanie aplikacji

Autor: Mateusz Bernacki  
Klasa: 5p  
Prowadzący: mgr inż. Kamil Wojnarowski

NOWY SĄCZ 2024r.

Spis treści

Wstęp 4

Testy jednostkowe 5

Testy integracyjne 7

Testy systemowe i akceptacyjne 10

Testy systemowe

Wstęp

Testowanie oprogramowania to kluczowy element procesu tworzenia aplikacji, mający na celu weryfikację poprawności działania kodu oraz wykrywanie błędów. Dzięki testom możemy upewnić się, że aplikacja spełnia oczekiwania użytkowników i działa zgodnie   
z założeniami.

Podstawowe zasady testowania obejmują:

* *Izolację testowanych komponentów,*
* *Powtarzalność testów,*
* *Niezależność od kolejności ich wykonywania,*
* *Pełne pokrycie kluczowych funkcji aplikacji.*

Testy można podzielić na różne typy:

* *Testy jednostkowe – sprawdzają pojedyncze elementy aplikacji,*
* *Testy integracyjne – weryfikują współdziałanie różnych modułów,*
* *Testy systemowe – badają całość działania aplikacji.*
* *Testy akceptacyjne - potwierdzają wykonanie aplikacji w pożądanej jakości*

W C# jednym z popularniejszych narzędzi do realizacji testów jednostkowych jest xUnit. Testy jednostkowe (unit tests) sprawdzają pojedyncze elementy aplikacji, takie jak metody czy funkcje, w izolacji od reszty systemu. xUnit wyróżnia się:

* *Prostą składnią i łatwą konfiguracją,*
* *Elastycznością w definiowaniu testów,*
* *Wsparciem dla współczesnych metodologii testowania.*

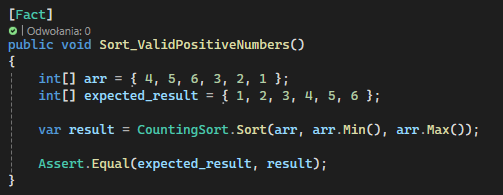
xUnit wspiera automatyzację testów, co jest szczególnie przydatne w procesie *CI/CD* (Continuous Integration/Continuous Deployment). Umożliwia szybkie wykrywanie błędów i zapewnia, że zmiany w kodzie nie wprowadzają nowych problemów.

Testy jednostkowe

Test pozytywnych cyfr

Opis: Sprawdza, czy algorytm poprawnie sortuje tablicę zawierającą losowe pozytywne liczby dodatnie.

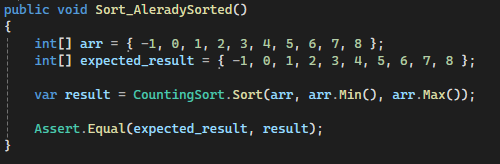
Wejście: Tablica {4,5,6,3,2,1}.

Oczekiwany wynik: {1,2,3,4,5,6}.

Test sortowania już posortowanej tablicy:

Opis: Sprawdza, czy algorytm działa poprawnie na tablicy, która już jest posortowana.

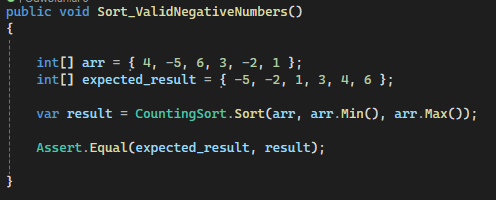
Wejście: Tablica {-1, 0, 1, 2, 3, 4, 5, 6, 7, 8}.

Oczekiwany wynik: {-1, 0, 1, 2, 3, 4, 5, 6, 7, 8}.

Test negatywnych cyfr:

Opis: Sprawdza, czy algorytm poprawnie sortuje tablicę zawierającą losowe negatywne liczby dodatnie.

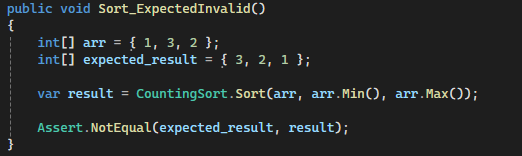
Wejście: Tablica {4,-5,6,3,-2,1}.

Oczekiwany wynik: {-5, -2, 1, 3, 4, 6}.

Test celowo zwracający fałsz:

Opis: Test został stworzony, aby nie przechodził.

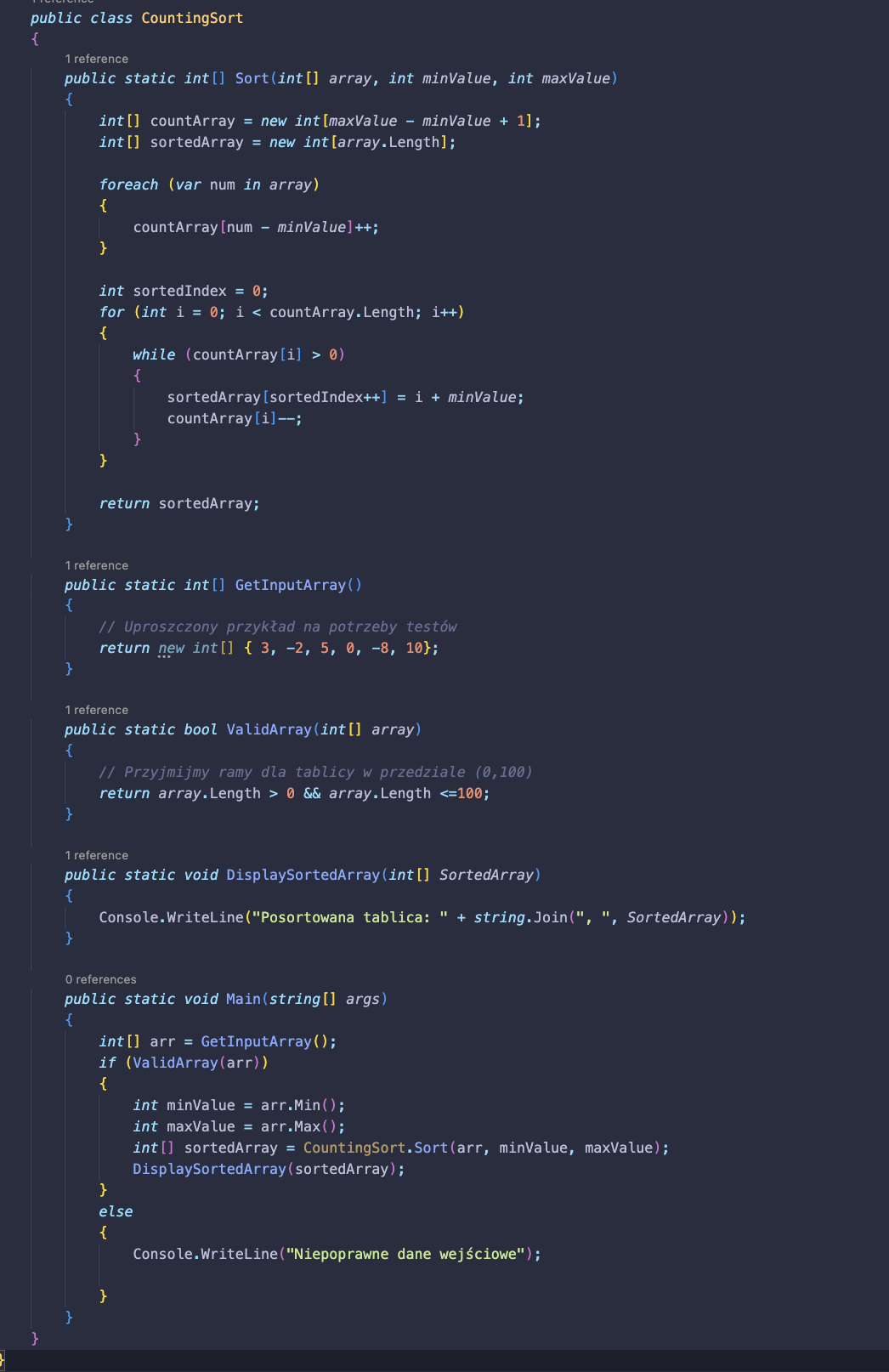
Wejście: Tablica {1,3,2}.

Oczekiwany wynik: {3, 2, 1} (celowo niepoprawny wynik). Test powinien zwrócić fałsz, ponieważ algorytm zawsze sortuje w porządku rosnącym, a oczekiwany wynik jest odwrotny.

Testy integracyjne

Testy te mają na celu wykrywanie błędów w interfejsach i interakcjach między modułami lub systemami. Polegają one na logicznym połączeniu modułów oprogramowania i ich testowaniu jako jednej grupy.

Testowanie integracyjne zapewnia prawidłowe działanie modułów i wykrywanie błędów związanych z interfejsem. Co więcej, testowanie integracyjne można rozpocząć już po udostępnieniu testowanych modułów. Przeprowadzenie testów nie wymaga bowiem uprzedniego zakończenia drugiego modułu.

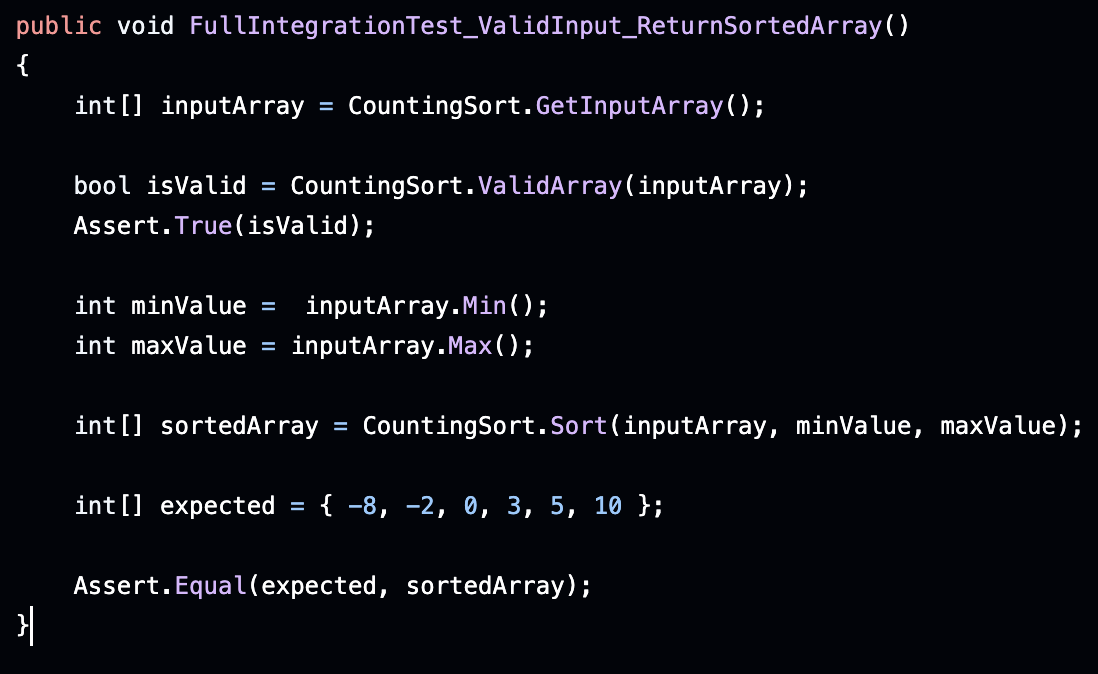
Opis algorytmu poszerzony o moduły symulacji input, walidacji danych i wyświetlania tablicy do użytkownika.

Pełny test integracyjny:

Opis: Sprawdza pełne działanie programu na poprawnych danych wejściowych.

Wejście: tablica {3, -2, 5, 0, -8, 10}.

Oczekiwany wynik: Tablica posortowana {-8, -2, 0, 3, 5, 10}.

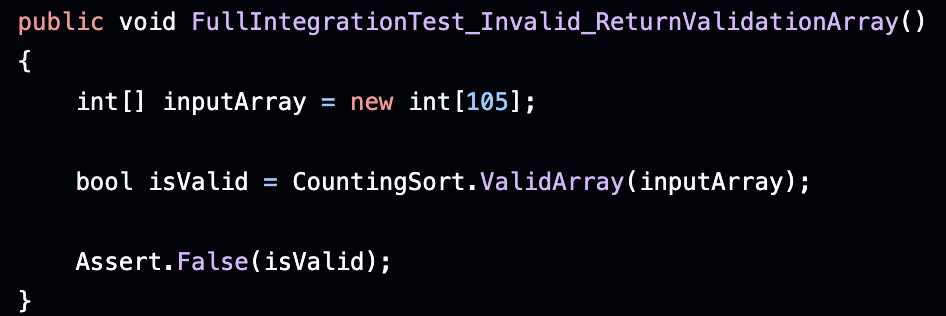


Test integracyjny niepoprawny:

Opis: Sprawdza, czy program poprawnie identyfikuje nieprawidłowe dane wejściowe (przekroczenie limitu liczby elementów).

Wejście: Tablica z 105 elementami.

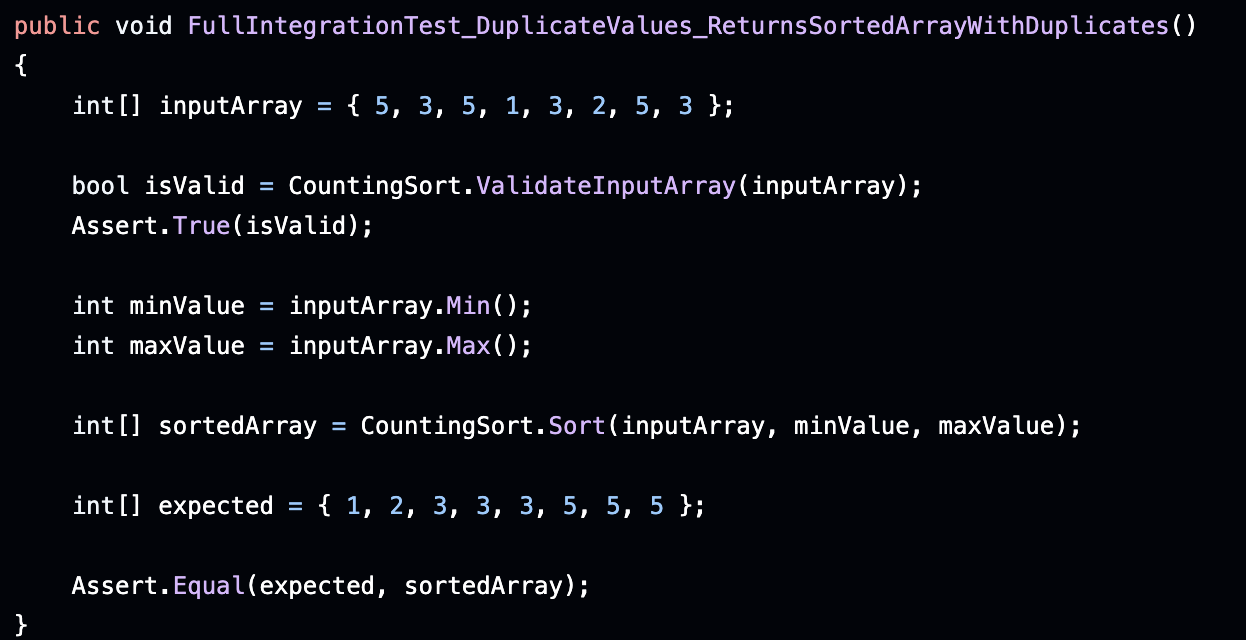
Oczekiwany wynik: Walidacja zwraca false

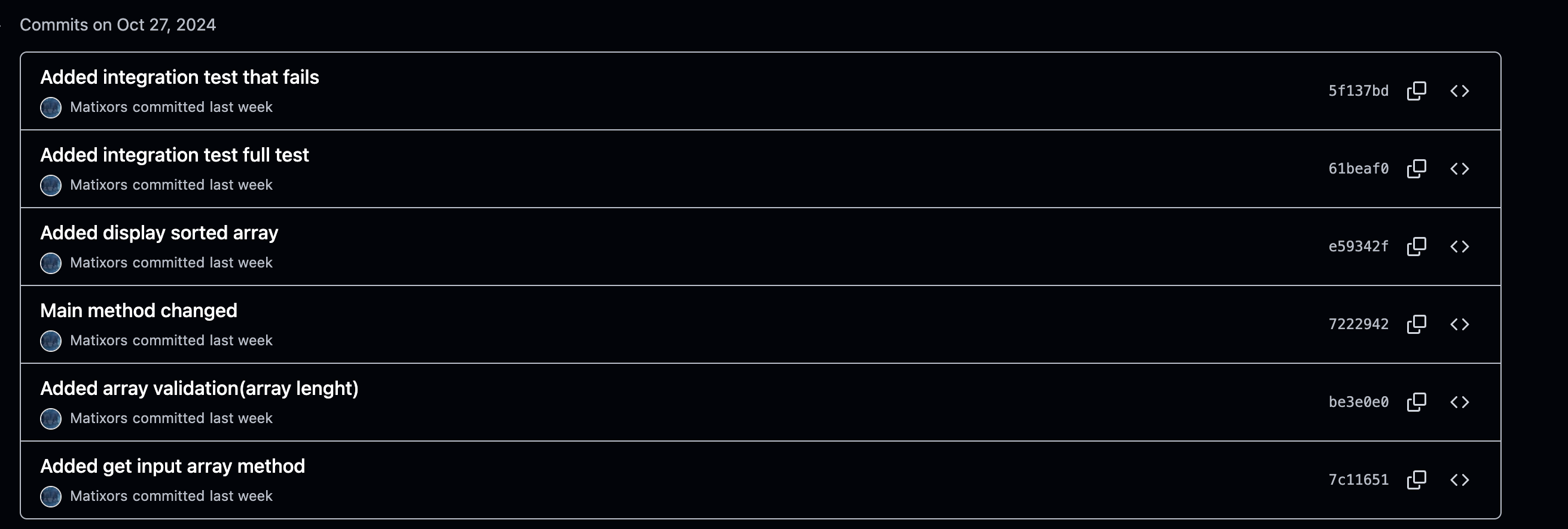


Test integracyjny z powtarzającymi się wartościami

Opis: Sprawdza, czy program poprawnie sortuje tablicę zawierającą powtarzające się wartości

Wejście: Tablica z powtarzającymi się wartościami.

Oczekiwany wynik: Tablica posortowana w z uwzględnieniem powtórzeń



Testy systemowe i akceptacyjne

Testy systemowe

Testy systemowe to etap testowania oprogramowania wykonywany na w pełni zintegrowanym i kompletnym systemie, którego celem jest weryfikacja zgodności systemu z określonymi wcześniej wymaganiami. Stanowią kompleksową ocenę działania aplikacji, dlatego niezwykle istotne jest, aby przeprowadzać je w środowisku jak najbardziej zbliżonym do produkcyjnego. Dzięki temu można wykryć potencjalne problemy środowiskowe, które mogłyby wpływać na komfort użytkowania systemu.

Na tym etapie testuje się zazwyczaj ukończony system, choć można je przeprowadzać wielokrotnie na różnych etapach cyklu życia oprogramowania. Weryfikowane są zarówno wymagania funkcjonalne, jak i niefunkcjonalne, takie jak wydajność, niezawodność czy skalowalność. Testy systemowe opierają się na technice czarnej skrzynki, a ich modelowy przebieg zakłada, że są realizowane przez niezależny zespół testerów. Typowe obiekty testowe obejmują:

* konfiguracje systemowe,
* dane konfiguracyjne,
* podręczniki użytkownika,
* systemowe dokumentacje operacyjne.

W ramach testów systemowych można wyróżnić wiele ich rodzajów, m.in.:

* testy regresji,
* testy użyteczności,
* testy wydajności,
* testy obciążenia,
* testy niezawodności,
* testy migracji,
* testy bezpieczeństwa,

W testach systemowych często stosuje się podejście end-to-end, gdzie badane są pełne scenariusze użytkownika, np.:

1. Założenie konta i logowanie.
2. Wykonanie kluczowych operacji, np. zakup produktu.
3. Wylogowanie z systemu.

Wyniki testów pomagają upewnić się, że system spełnia wymagania i działa zgodnie z oczekiwaniami, zanim zostanie przekazany do dalszych etapów oceny.