# ZESPÓŁ SZKÓŁ ELEKTRYCZNO-MECHANICZNYCH

## Pracownia testowania i dokumentowania aplikacji

#### Testowanie aplikacji

**Autor:** Mateusz Kmak

**Klasa:** 5P

**Prowadzący:** mgr inż. Kamil Wojnarowski

###### Nowy Sącz 2024r.

Spis treści

**[1. Testy jednostkowe 5](#_Toc24771)**

**[1.1 Implementacja programu 6](#_Toc8847)**

**[1.2 Implementacja testów 6](#_Toc23234)**

**[1.3 Git 10](#_Toc7125)**

**[2. Testy integracyjne 11](#_Toc13378)**

**[2.1 Implementacja programu 11](#_Toc19966)**

**[2.2 Implementacja testów 12](#_Toc13811)**

**[2.3 Git 14](#_Toc1029)**

**[3. Testy systemowe 15](#_Toc22445)**

# Wstęp

Testowanie oprogramowania jest jednym z kluczowych elementów procesu tworzenia aplikacji, mającym na celu zapewnienie jej poprawnego, stabilnego i bezpiecznego działania. Jego głównym celem jest wykrycie błędów oraz upewnienie się, że oprogramowanie spełnia założenia projektowe i wymagania użytkowników. Oprogramowanie bez gruntownego testowania może być podatne na awarie, co skutkuje obniżeniem jego wartości w oczach użytkownika.

W zależności od rodzaju aplikacji, testowanie może obejmować różne poziomy. Najczęściej stosowane są testy: jednostkowe, integracyjne, systemowe oraz akceptacyjne.

* Testy jednostkowe służą do weryfikacji pojedynczych komponentów oprogramowania, takich jak funkcje lub klasy. Dzięki nim możliwe jest szybkie wykrycie błędów w zie na najniższym poziomie.
* Testy integracyjne sprawdzają, czy różne moduły aplikacji współpracują ze sobą zgodnie z założeniami. Na tym etapie weryfikowane są interakcje pomiędzy poszczególnymi częściami systemu, co pozwala na zidentyfikowanie problemów związanych z komunikacją między nimi.
* Testy systemowe mają na celu ocenę działania całości aplikacji. W ich ramach sprawdzane są funkcjonalności z perspektywy końcowego użytkownika, a także zgodność oprogramowania z wymaganiami niefunkcjonalnymi, takimi jak wydajność, bezpieczeństwo czy skalowalność. Testy te odbywają się w środowisku jak najbardziej zbliżonym do produkcyjnego, co pozwala na symulację rzeczywistych warunków pracy aplikacji.
* Testy akceptacyjne mają na celu potwierdzenie, że aplikacja spełnia wszystkie założenia biznesowe i wymagania stawiane przez klienta. Testy te są często przeprowadzane przez zespół klientów lub użytkowników końcowych, którzy oceniają, czy oprogramowanie spełnia ich oczekiwania. To one decydują o ostatecznej akceptacji aplikacji przed jej wdrożeniem.

Warto zaznaczyć, że testowanie nie jest jednorazowym procesem. Nowe funkcje, poprawki i zmiany w oprogramowaniu wymagają cyklicznego powtarzania testów, aby upewnić się, że nie wprowadziły one nowych problemów. Współczesne techniki automatyzacji testów pozwalają na regularne i szybkie sprawdzanie aplikacji, co zwiększa efektywność całego procesu i pozwala na bieżąco monitorować jakość oprogramowania.

# Testy jednostkowe

Testy jednostkowe to jedna z podstawowych technik stosowanych w procesie testowania oprogramowania. Skupiają się na weryfikacji działania najmniejszych fragmentów kodu. Zazwyczaj są to pojedyncze klasy, metody lub funkcje. Każdy z tych testów ma za zadanie sprawdzić, czy dany element działa zgodnie z założeniami w pełnej izolacji od pozostałych części programu. To kluczowy etap, bo błędy w małych komponentach mogą powodować większe problemy na poziomie całego projektu.

**Kiedy wykonujemy testy jednostkowe?**

Najczęściej wykorzystujemy je w trakcie tworzenia nowych funkcji lub modyfikacji istniejącego kodu. W idealnym scenariuszu, testy jednostkowe tworzy się równocześnie z kodem co pozwala na bieżąco sprawdzać, czy kod działa poprawnie.Regularne wykonywanie tych testów pozwala na szybkie wykrycie sytuacji, w której nowa zmiana wprowadza błędy w funkcjach, które wcześniej działały poprawnie.

**Przykład**

Załóżmy, że tworzony jest system do zarządzania zamówieniami. Jednym z jego elementów jest funkcja, która oblicza łączny koszt zamówienia na podstawie listy produktów i ich ilości. Test jednostkowy dla tej funkcji mógłby polegać na tym, aby sprawdzić, czy dla pewnej listy produktów, wynik zwracany przez funkcję jest poprawny. Innym testem mogłaby być sytuacja, w której lista produktów jest pusta i sprawdzenie czy program poprawnie zwróci wartość 0.

**Narzędzie do testowania xUnit**

xUnit jest to framework, który umożliwia pisanie testów w prosty i czytelny sposób. Każdy test w jest oznaczony atrybutem [Fact], co jednoznacznie wskazuje, że dana metoda jest testem. xUnit oferuje szeroki zestaw narzędzi, które pozwalają na weryfikację wyników, porównywanie oczekiwanych wartości z rzeczywistymi, a także obsługę wyjątków. Jedną z największych zalet jest integracja z popularnymi środowiskami programistycznymi, takimi jak Visual Studio. Dzięki temu testy mogą być uruchamiane automatycznie podczas kompilacji, a wszelkie błędy są natychmiast wykrywane. xUnit wspiera również bardziej zaawansowane scenariusze testowe, takie jak testy parametryzowane, które pozwalają na przetestowanie tej samej funkcji z różnymi danymi wejściowymi.

## Implementacja programu

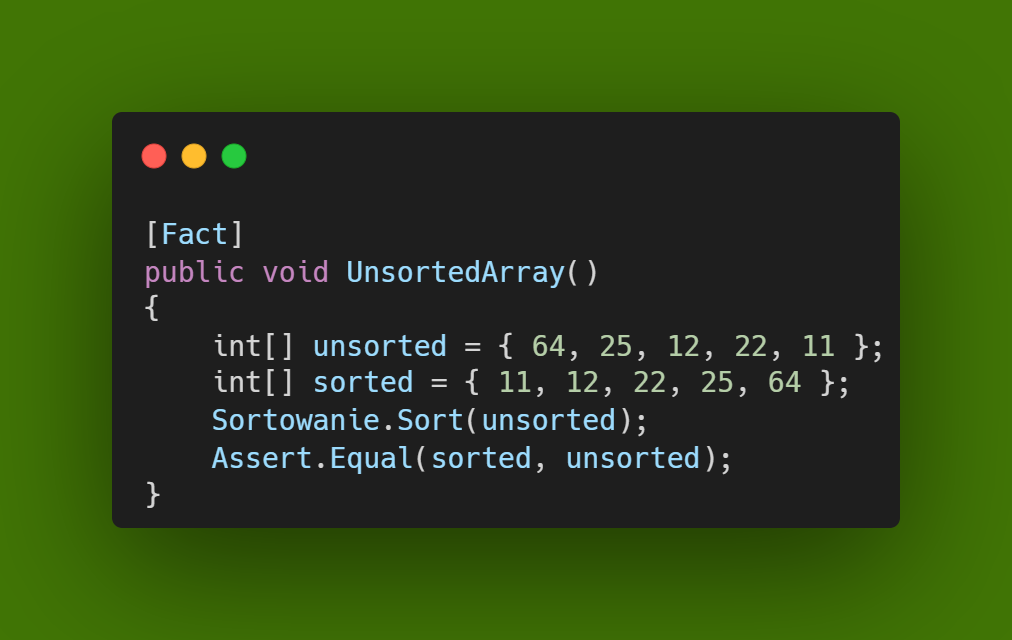
Program implementuje algorytm sortowania, który sortuje przekazaną tablicę liczb całkowitych w porządku rosnącym. Algorytm działa w sposób iteracyjny, porównując kolejne elementy tablicy i zamieniając miejscami najmniejszy znaleziony element z aktualnym elementem na danej pozycji.



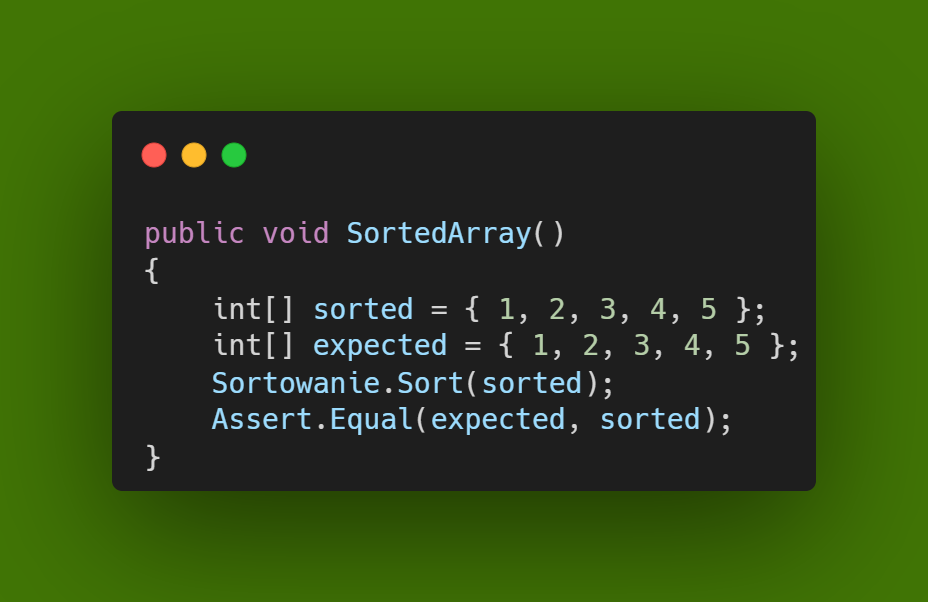
## Implementacja testów

Do testowania algorytmu sortowania zaimplementowanego w klasie Sortowanie, zastosowano framework xUnit, który umożliwia przeprowadzanie testów jednostkowych. Poniżej przedstawiono kilka przypadków testowych, które sprawdzają działanie funkcji sortującej dla różnych zestawów danych.

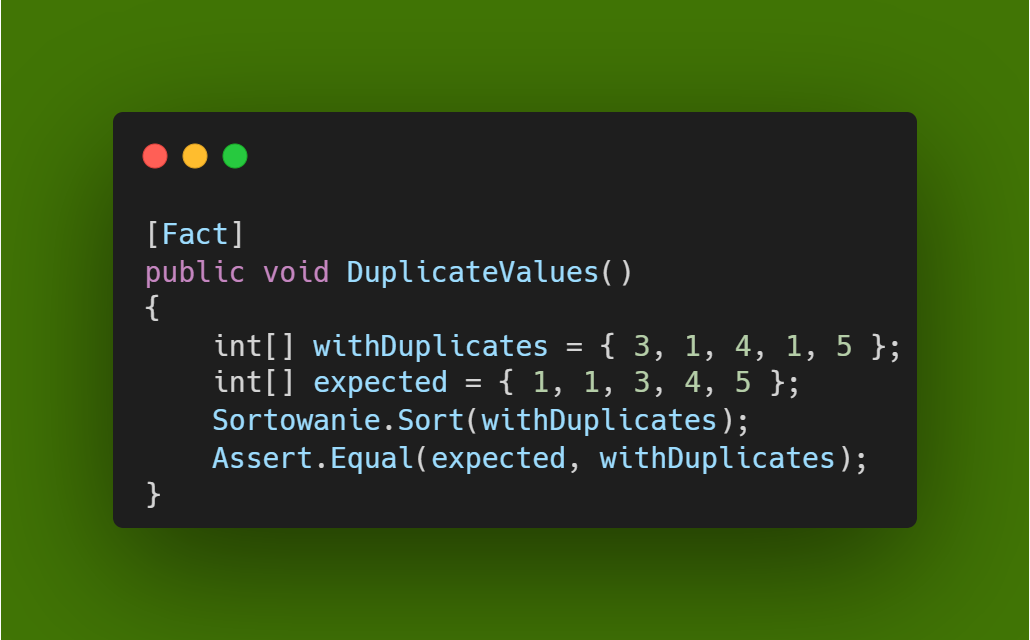
**Test dla nieposortowanej tablicy**  
Ten test sprawdza działanie algorytmu na typowym przykładzie – tablicy liczb, które są w losowej kolejności. Oczekiwane jest, że po wywołaniu funkcji sortującej tablica zostanie uporządkowana w sposób rosnący.



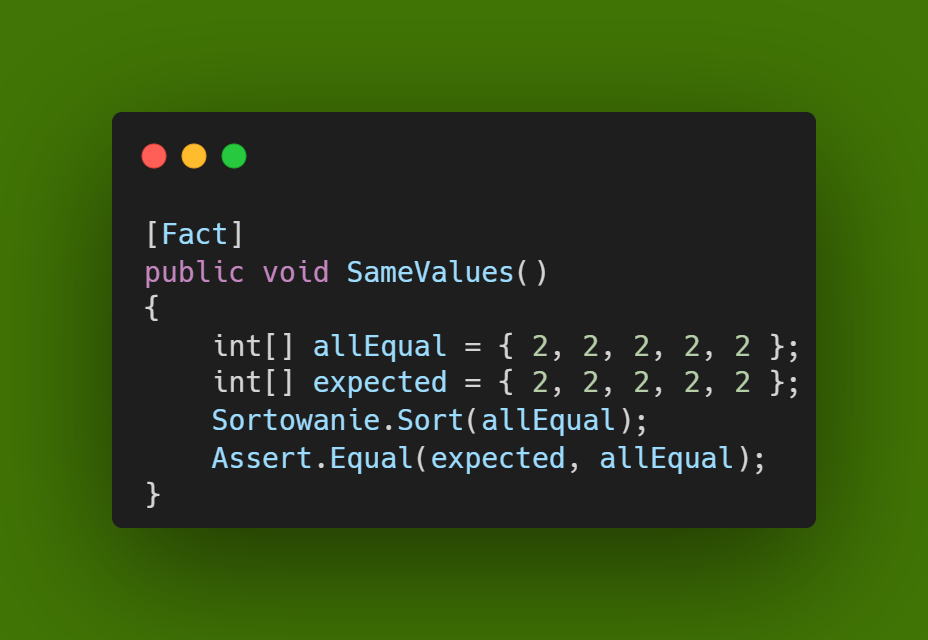
**Test dla już posortowanej tablicy**Ten test weryfikuje, czy algorytm poprawnie rozpoznaje i pozostawia bez zmian tablicę, która jest już posortowana.



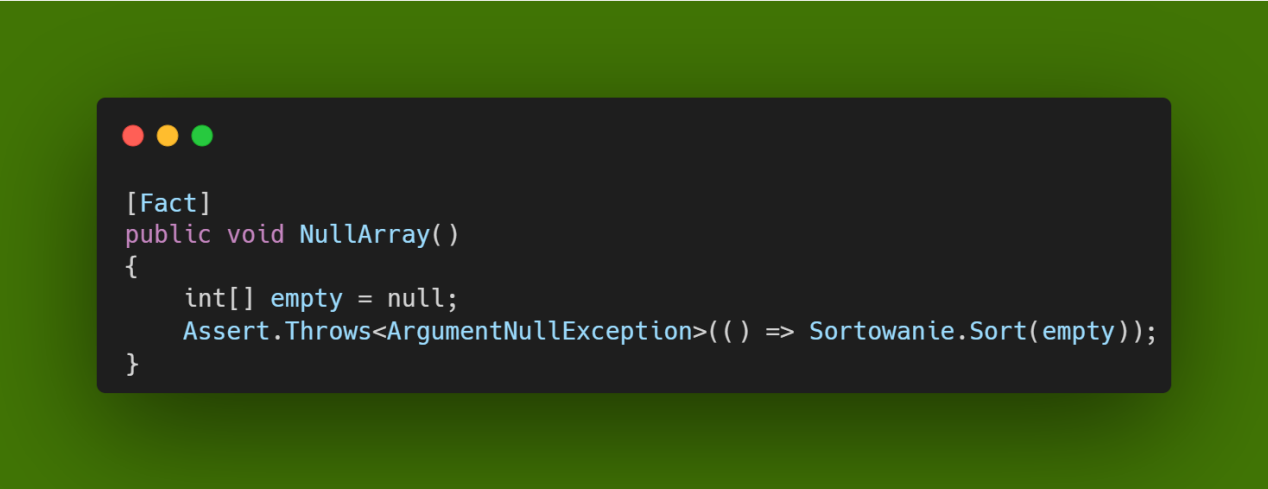
**Test dla tablicy z powtórzonymi wartościami**W przypadku tablicy z powtarzającymi się wartościami, algorytm powinien poprawnie posortować liczby, pozostawiając powtórzenia na swoich miejscach.



**Test dla tablicy z identycznymi wartościami**  
Test sprawdza przypadek, w którym wszystkie wartości w tablicy są identyczne. Algorytm nie powinien wprowadzać żadnych zmian, a wynikowa tablica powinna być taka sama jak na wejściu.

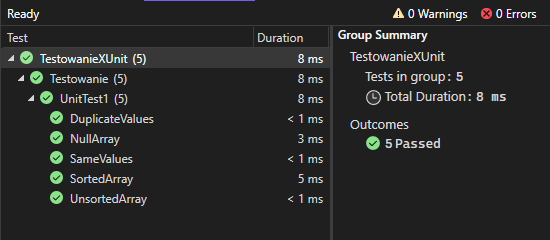


**Test dla tablicy null**  
W tym przypadku sprawdzamy, czy program poprawnie reaguje na niepoprawne dane wejściowe, czyli tablicę z wartością null. Zgodnie z założeniem funkcja powinna zgłosić wyjątek ArgumentNullException.



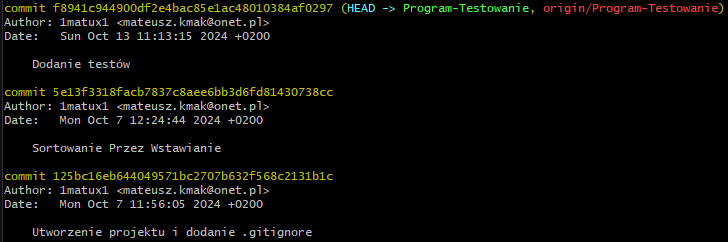
Test jest uznawany za negatywny, ponieważ sprawdza scenariusz, w którym spodziewamy się wystąpienia błędu, a nie poprawnego działania funkcji.

**Podsumowanie testów**



Wszystkie testy zostały pomyślnie uruchomione i zakończyły się sukcesem. Zarówno testy pozytywne, jak i negatywny przyniosły oczekiwane rezultaty. Funkcja sortująca poprawnie sortuje tablice w różnych przypadkach, a także zgłasza odpowiedni wyjątek w przypadku błędnych danych wejściowych. Dzięki tym testom można stwierdzić, że implementacja algorytmu sortowania jest stabilna i działa zgodnie z założeniami.

## Git



# Testy integracyjne

Testy integracyjne skupiają się na sprawdzaniu współdziałania poszczególnych modułów aplikacji, w przeciwieństwie do testów jednostkowych, które badają pojedyncze, izolowane fragmenty kodu. Głównym celem testów integracyjnych jest upewnienie się, że moduły poprawnie współpracują, a dane są prawidłowo przekazywane między komponentami. W przypadku systemów bardziej złożonych, testy integracyjne pomagają wykryć błędy wynikające z nieprawidłowej komunikacji między modułami, np. przy przekazywaniu danych między warstwą logiki biznesowej a warstwą dostępu do danych.

## Implementacja programu

Na potrzeby testów integracyjnych został rozbudowany istniejący program   
o dodatkowy moduł, który symuluje interakcję między komponentem sortującym,   
a systemem odpowiedzialnym za przetwarzanie wyników sortowania.



## Implementacja testów

**Test dla tablicy o nieparzystej liczbie elementów**Test sprawdza poprawność wyznaczania mediany dla tablicy z nieparzystą liczbą elementów. Oczekuje się, że funkcja zwróci środkową wartość.



**Test dla tablicy o parzystej liczbie elementów**Ten test weryfikuje obliczenie mediany dla tablicy o parzystej liczbie elementów, gdzie oczekiwane jest wyliczenie średniej dwóch środkowych wartości.



**Test dla tablicy z identycznymi wartościami**Sprawdza działanie funkcji Mediana dla tablicy, w której wszystkie elementy są takie same. W takim przypadku wynik powinien być równy tej wartości.



**Test dla wartości maksymalnej i minimalnej**Test ma na celu sprawdzenie, czy algorytm poprawnie identyfikuje wartości minimalną i maksymalną w tablicy.



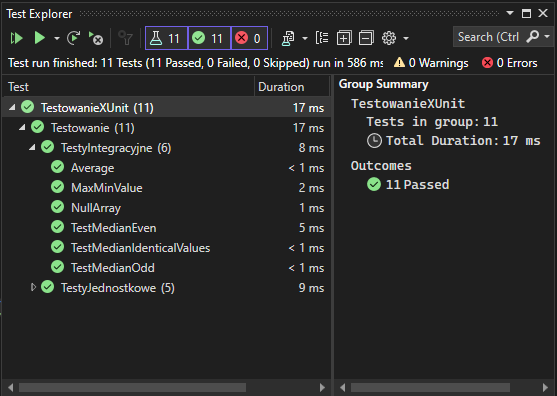
**Test dla średniej wartości**Test weryfikuje poprawność wyznaczania średniej wartości dla tablicy liczb całkowitych. Funkcja powinna zwrócić średnią arytmetyczną wszystkich elementów.



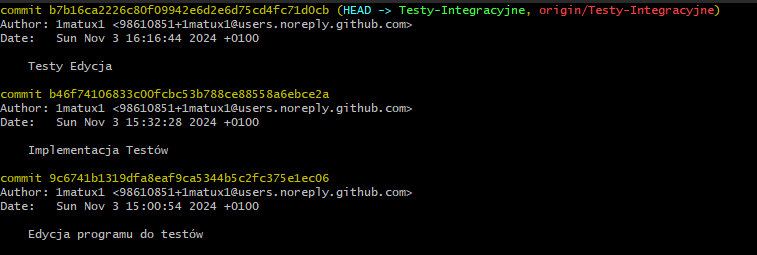
**Test dla tablicy null**Sprawdza, czy program poprawnie reaguje na próbę stworzenia obiektu z tablicą null, zgłaszając odpowiedni wyjątek ArgumentNullException.



**Podsumowanie testów**



## Git



# Testy systemowe

Testy systemowe to jeden z kluczowych etapów w procesie weryfikacji oprogramowania. To dzięki nim możemy upewnić się, że cała aplikacja funkcjonuje jako spójny system i spełnia zarówno założenia techniczne, jak i oczekiwania użytkowników. W odróżnieniu od testów jednostkowych, które badają poszczególne fragmenty kodu, czy testów integracyjnych, skupiających się na współdziałaniu modułów, testy systemowe obejmują działanie całego programu w warunkach zbliżonych do rzeczywistych.

**Czym są tetsy systemowe?**

Podstawowym celem testów systemowych jest weryfikacja kompletnego oprogramowania w jego finalnej formie. Obejmują one wszystkie aspekty aplikacji , od interfejsu użytkownika, poprzez logikę, aż po komunikację z bazami danych i serwerami zewnętrznymi.

Testy te odpowiadają na pytanie: „Czy aplikacja działa dokładnie tak, jak powinna?” Oznacza to sprawdzenie zgodności działania programu z jego przeznaczeniem oraz wymogami użytkowników. Dzięki nim można ocenić, czy produkt gotowy jest do udostępnienia, czy może wymaga jeszcze poprawek.

**Jak wygląda proces testów systemowych?**

Przeprowadzenie testów systemowych to złożony proces, który można podzielić na kilka kluczowych etapów:

1. **Analiza wymagań**Każdy krok rozpoczyna się od zrozumienia wymagań wobec aplikacji. Stanowią one fundament do przygotowania scenariuszy testowych.
2. **Tworzenie scenariuszy testowych**  
   Scenariusze testowe dokładnie opisują kroki użytkownika i oczekiwane wyniki. Przykład: „Użytkownik wpisuje poprawne dane logowania i zostaje przekierowany na stronę główną”.
3. **Konfiguracja środowiska testowego**  
   Aplikacja uruchamiana jest w środowisku symulującym warunki rzeczywiste, np. na różnych systemach operacyjnych czy w różnych przeglądarkach.
4. **Wykonywanie testów**Testy systemowe można przeprowadzać ręcznie lub automatycznie. Testy manualne przeprowadzane są przez testerów, natomiast automatyczne wykonują specjalne narzędzia.
5. **Analiza wyników**  
   Wyniki testów są szczegółowo analizowane. W przypadku błędów określa się ich źródło, a następnie programiści przystępują do ich usunięcia.

**Kiedy warto korzystać z testów systemowych?**

* Testy systemowe stosuje się w późniejszych etapach cyklu tworzenia oprogramowania. Są szczególnie istotne:
* Przed wydaniem nowej wersji – weryfikując, czy wszystkie funkcje działają poprawnie.
* Po dodaniu nowych modułów – sprawdzając, czy ich integracja nie wpłynęła negatywnie na całość aplikacji.
* Przy zmianach w środowisku pracy – np. przy adaptacji aplikacji na nową platformę.

**Dlaczego testy systemowe są tak istotne?**

Testy systemowe przynoszą wiele korzyści:

* Kompleksowość – testowana jest całość aplikacji, co pozwala na wykrycie błędów pominiętych we wcześniejszych etapach.
* Realistyczne warunki – testy te naśladują rzeczywiste środowisko użytkownika, co pomaga ocenić aplikację w kontekście jej docelowego użycia.
* Redukcja ryzyka – dzięki wczesnemu wykryciu błędów minimalizuje się koszty i ryzyko problemów po wdrożeniu.
* Zwiększenie jakości – weryfikacja oprogramowania pod kątem zgodności z wymaganiami przekłada się na większe zadowolenie użytkowników.

**Podsumowanie**

Testy systemowe to jeden z filarów procesu tworzenia niezawodnego oprogramowania. Pomagają sprawdzić, czy aplikacja spełnia wymagania funkcjonalne, działa stabilnie i jest gotowa do użytku w realnym świecie. Chociaż przeprowadzenie testów systemowych wymaga czasu i zasobów, ich rola w zapewnieniu jakości i stabilności oprogramowania jest nieoceniona. To inwestycja, która zwraca się w postaci zadowolenia użytkowników i redukcji kosztów związanych z naprawą błędów po wdrożeniu.