Testowanie i Jakość Oprogramowania

Akademia Tarnowska

Katedra Informatyki

Tomasz Gądek

Wykład 10: Wprowadzenie do testowania, piramida testów

Testowanie oprogramowania

Testowanie oprogramowania jest to sprawdzenie poprawności i zgodności implementacji oprogramowania z założeniami zapisanymi w formie dokumentacji lub też z założeniami zlecającego wytworzenie oprogramowania.

Dlaczego aplikacje nie są testowane?

Aplikacje nie jest testowana, ponieważ...

zajmuje to zbyt wiele czasu.

Aplikacje nie jest testowana, ponieważ...

nie wiem jak kod powinien się zachowywać, więc go nie testuje.

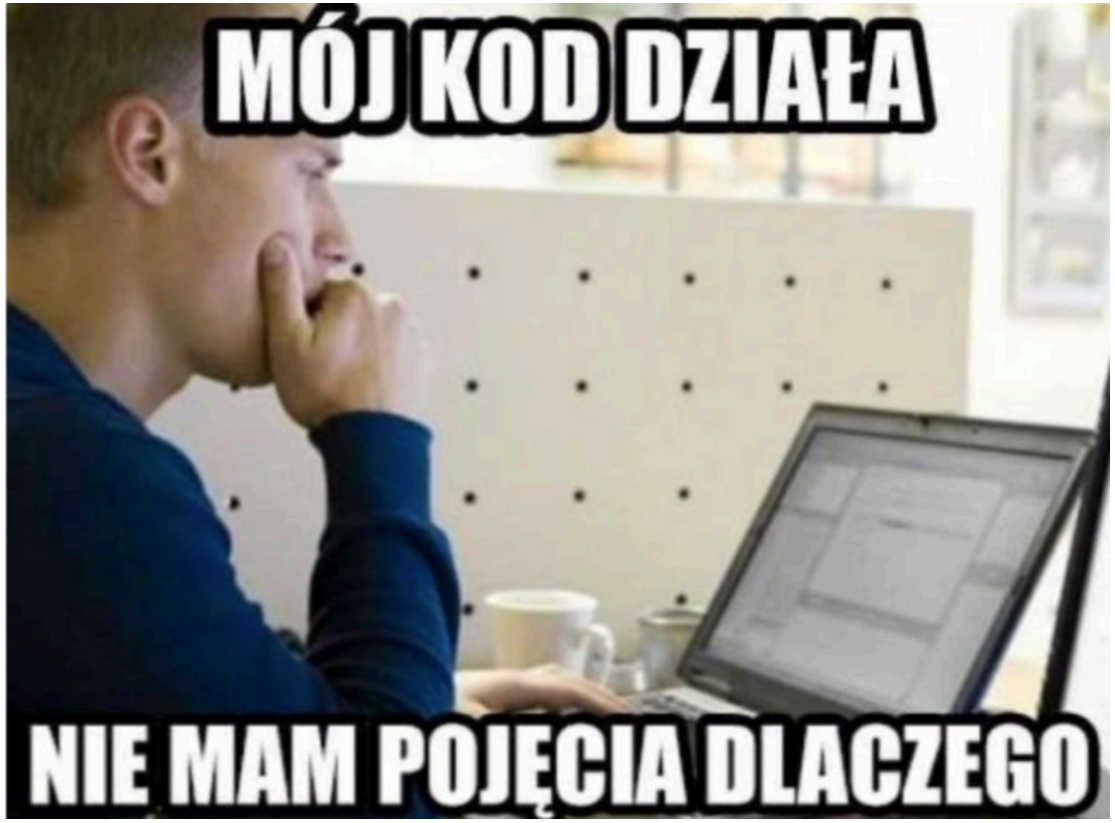
Aplikacje nie jest testowana, ponieważ...

to jest nudne.

Aplikacje nie jest testowana, ponieważ...

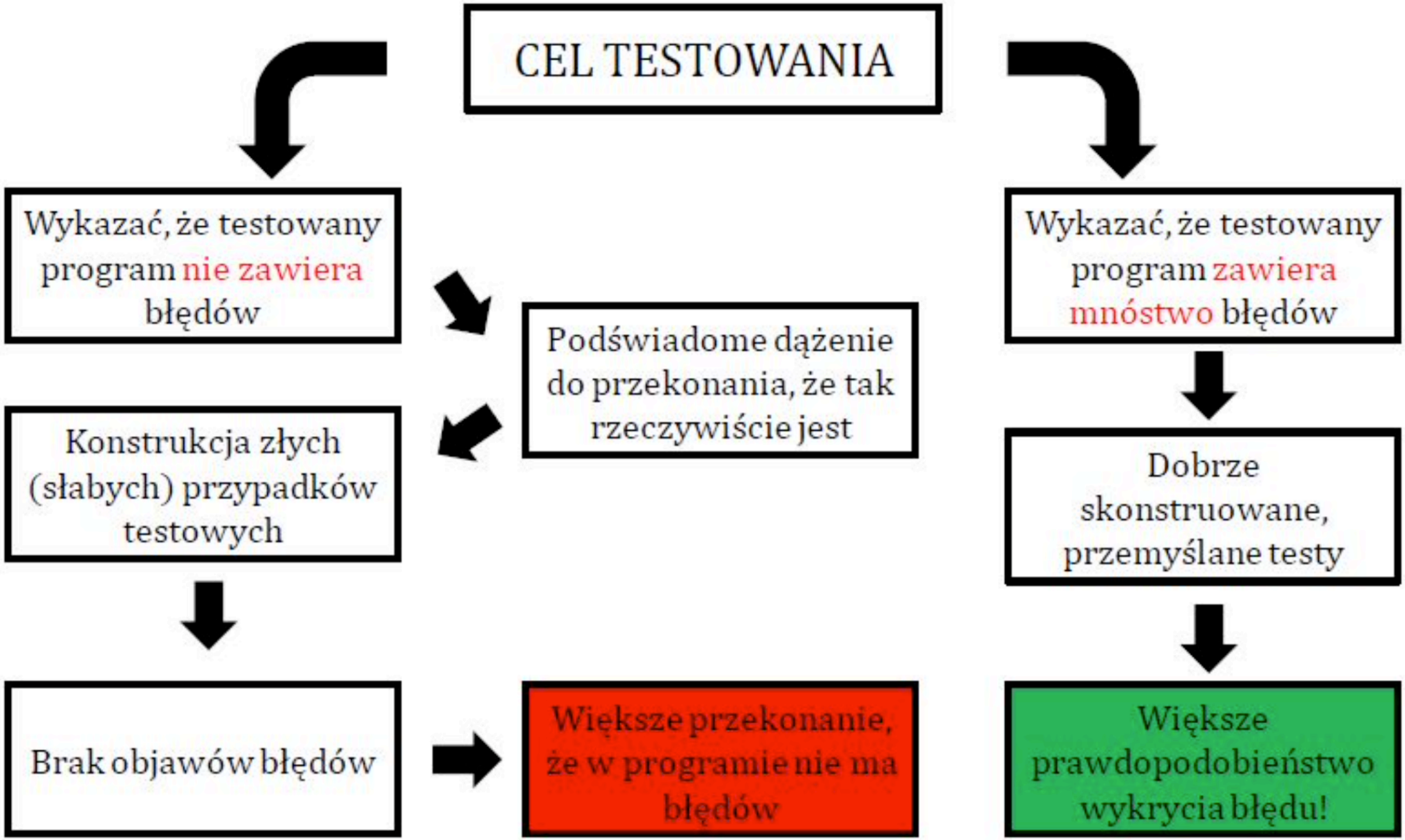
działa (???).

Konsekwencje...



Jaki jest cel testowania

Celem testowania jest **znajdowanie błędów**.


CEL TESTOWANIA
Wykazać, że testowany program nie zawiera błędów
Konstrukcja złych (słabych) przypadków testowych
Brak objawów błędów
Podświadome dążenie do przekonania, że tak rzeczywiście jest
Większe przekonanie, że w programie nie ma błędów
Wykazać, że testowany program zawiera mnóstwo błędów
Dobrze skonstruowane, przemyślane testy
Większe prawdopodobieństwo wykrycia błędu!

Kiedy mamy do czynienia z błędem?

> Gdy program **nie robi czegoś co zostało wymienione w jego specyfikacji**.

Dokumentacja: Użytkownik z uprawnieniami administratora **ma** mieć dostęp do zasobu **A**.

Program: Użytkownik z uprawnieniami administratora **nie ma** dostępu do zasobu **A**.

Kiedy mamy do czynienia z błędem?

> Gdy program **wykonuje coś czego według specyfikacji nie powinien robić**.

Dokumentacja: Użytkownik **niezalogowany** nie może dodać nowego produktu do listy.

Program: **Każdy** może dodać nowy produkt do listy.

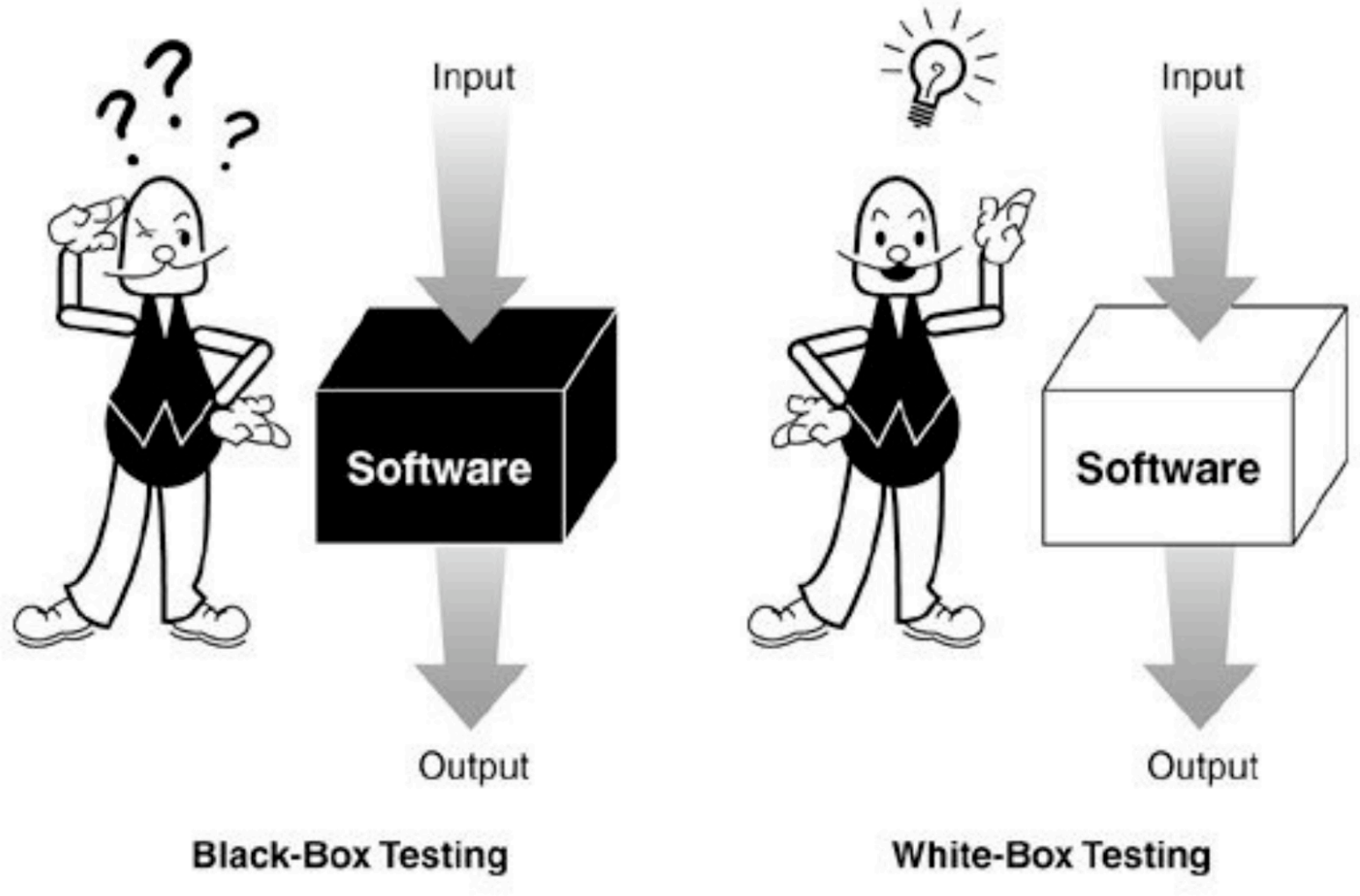
Kiedy mamy do czynienia z błędem?

> Gdy program **robi coś o czym specyfikacja nie wspomina**.

Kiedy mamy do czynienia z błędem?

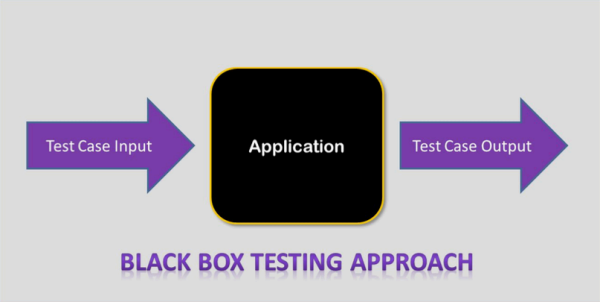
> Gdy program **jest trudny do zrozumienia, powolny lub skomplikowany**.

Testy białej i czarnej skrzynki



Testy czarnej skrzynki

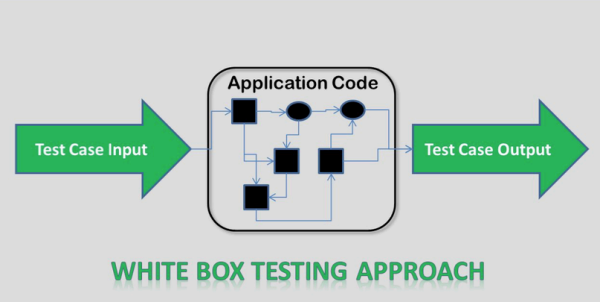
> Są stosowane gdy wewnętrzna budowa / implementacja oprogramowania nie są znane testerowi.



- Testy wykonywane z punktu widzenia użytkownika pomagające odnaleźć rozbieżności w specyfikacji.
- Testowanie może się odbywać niezależnie od programistów tworzących kod testowanego programu co zwiększa obiektywizm.
- Możliwość szybkiego tworzenia przypadków testowych.

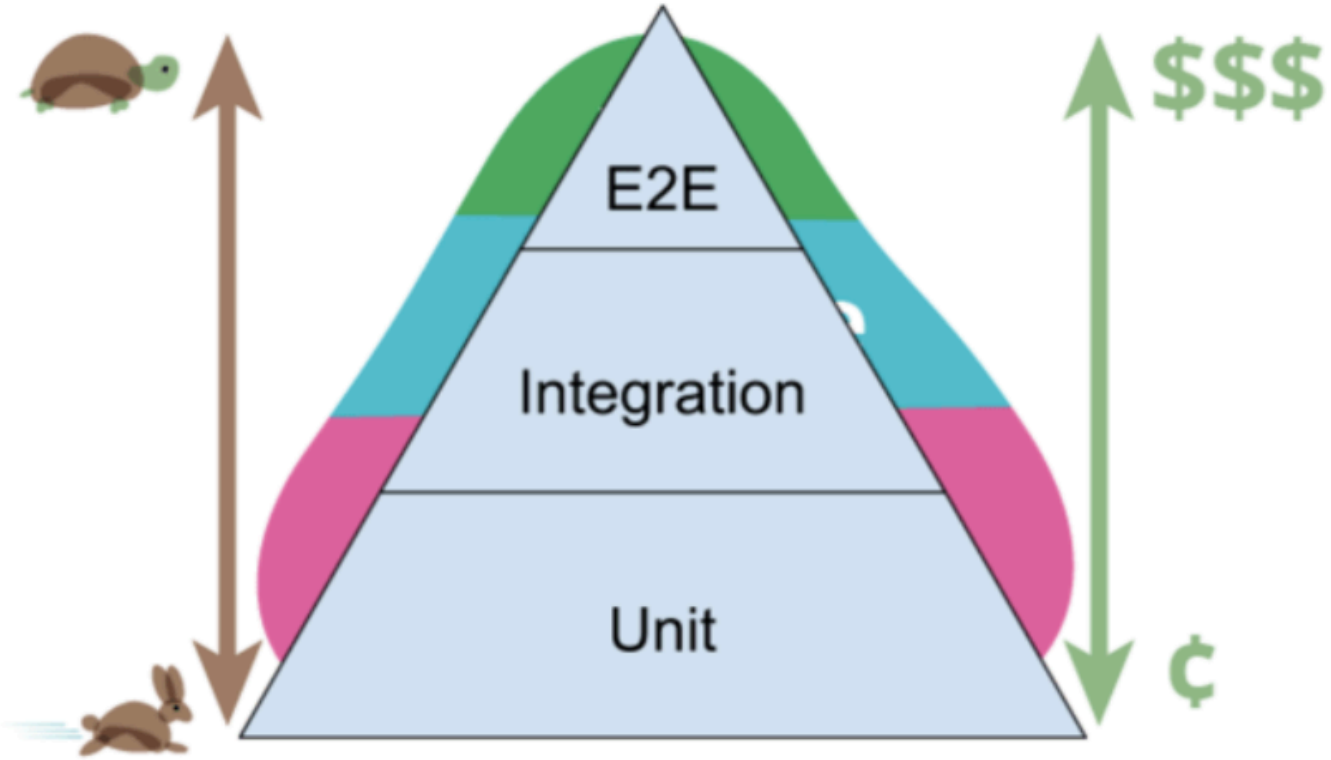
Testy białej skrzynki

> Tester musi posiadać wiedzę na temat działania kodu testowanego oprogramowania.



- Badanie jest tym bardziej dokładne im pokrycie testami kodu jest większe.
- W prosty sposób można zidentyfikować brakujące przypadki testowe.

Piramida testów



Główne rodzaje testów

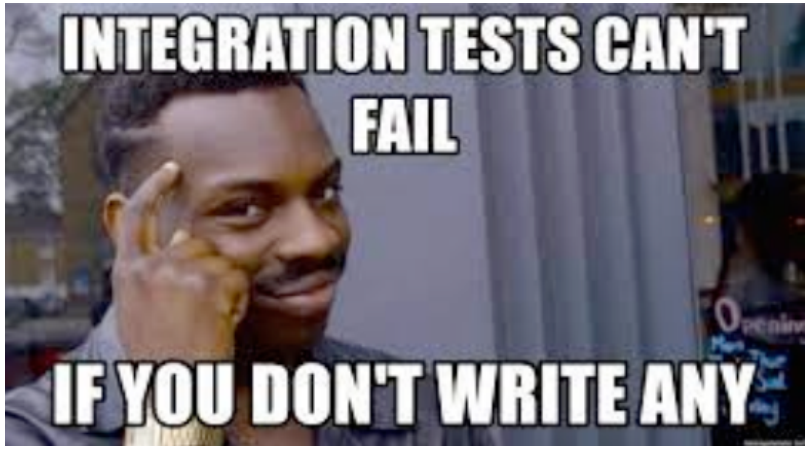
- unit tests (testy jednostkowe)
  - Narzędzia: **unittest**.
- integration tests (testy integracyjne)
  - Narzędzia: **unitest** / **pytest**.
- end-to-end tests / e2e (testy od końca do końca)
  - Narzędzia: **Selenium**.

Testy jednostkowe

Skupiają się na weryfikowaniu poprawności pojedynczych komponentów lub modułów systemu, zwanych jednostkami, niezależnie od reszty systemu.

Testy integracyjne

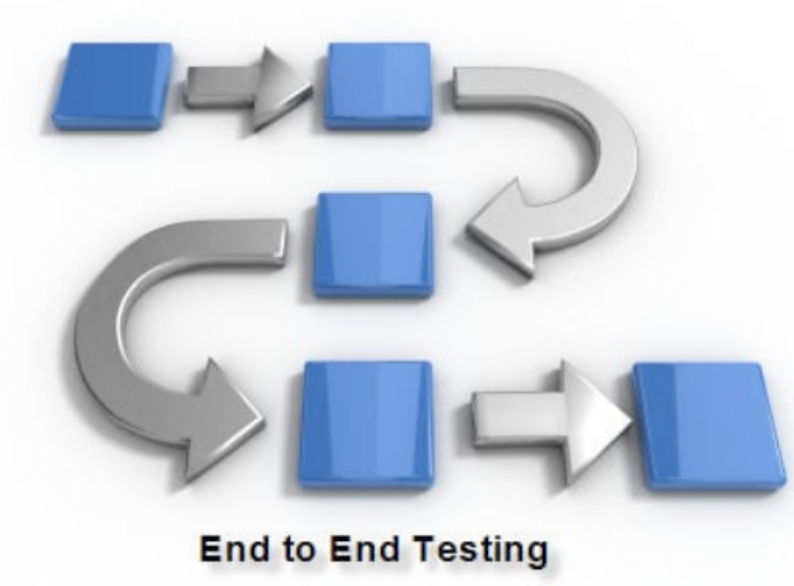
Weryfikują, czy poszczególne komponenty działają poprawnie razem jako zintegrowana całość. Testujemy ich integrację, bez wnikania czy te moduły są prawidłowe, gdyż to robią testy jednostkowe.



unit tests passing - no integration tests

Testy e2e

Koncentrują się na weryfikacji, czy cały system działa. Począwszy od wejścia danych, przez przetwarzanie i logikę biznesową, aż do wyjścia lub rezultatu końcowego. Skupiają się na rzeczywistej interakcji użytkownika z systemem.



Przykład

> Funkcja obliczająca cenę biletów dla grupy osób.

Oblicz cenę biletów dla wybranej ilości osób. Jeśli wśród nich są dzieci, należy obniżyć cenę ich biletów o **50%**.

Dane wejściowe:

- **tickets** - liczba biletów do zakupu.
- **children** - liczba dzieci.
- **regular_price** - cena regularna.

Implementacja (regularna cena)

> Obliczenie kosztu biletów po regularnej cenie.

Dane wejściowe: liczba biletów do zakupu: **5**; liczba dzieci: **0**; cena regularna: **6(zł)**.

app.py

```
def calculate_tickets_price(tickets, children, regular_price):
  total_price = tickets * regular_price
```

Implementacja (regularna cena)

Funkcja zwraca słownik z właściwością **total_price** z wartością wszystkich biletów.

app.py

```
def calculate_tickets_price(tickets, children, regular_price):
  total_price = tickets * regular_price

  return {"total_price": total_price, "discount": 0}
```

Implementacja (zniżka dla dzieci)

> Implementacja zniżki dla dzieci.

Dane wejściowe: liczba biletów do zakupu: **5**; liczba dzieci: **2**; cena regularna: **6(zł)**.

```
if(children > 0):
  discount = 0.5 * regular_price * children
  price_with_discount = total_price - discount

  return {"total_price": price_with_discount, "discount": discount}
```

Implementacja (zniżka dla dzieci)

```
def calculate_tickets_price(tickets, children, regular_price):

    total_price = tickets * regular_price

    if(children > 0):
        discount = 0.5 * regular_price * children
        price_with_discount = total_price - discount

        return {"total_price": price_with_discount, "discount": discount}

    return {"total_price": total_price, "discount": 0}
```

Jak sprawdzić, czy funkcja działa poprawnie?

Czy funkcja działa poprawnie?

Zakup biletów pełnych.

Dane wejściowe: liczba biletów do zakupu: **5**; liczba dzieci: **0**; cena regularna: **6(zł)**.

```
result = calculate_tickets_price(5, 0, 6)["total_price"]
print(f"Total: {result}")
```

```
Total: 30
```

Czy funkcja działa poprawnie?

Zakup biletów pełnych i ulgowych.

Dane wejściowe: liczba biletów do zakupu: **5**; liczba dzieci: **2**; cena regularna: **6(zł)**.

```
result = calculate_tickets_price(5, 2, 6)["total_price"]
print(f"Total: {result}")
```

```
Total: 24
```

Czy funkcja działa poprawnie?

Czy funkcja `calculate_tickets_price()` jest przetestowana wystarczająco dokładnie???

1. Zakup biletów bez zniżki.
2. Zakup biletów ze zniżką.

A co się stanie gdy...

```
result = calculate_tickets_price(0, 2, 6)["total_price"]
print(f"Total: {result}")
```

```
Total: ???
```

A co się stanie gdy...

```
result = calculate_tickets_price(0, 2, 6)["total_price"]
print(f"Total: {result}")
```

```
Total: -6.0
```

;)

TO BE CONTINUED...

Koniec