

# Po co nam testy?

---

TESTOWANIE KODU W PRAKTYCE

V 1.0

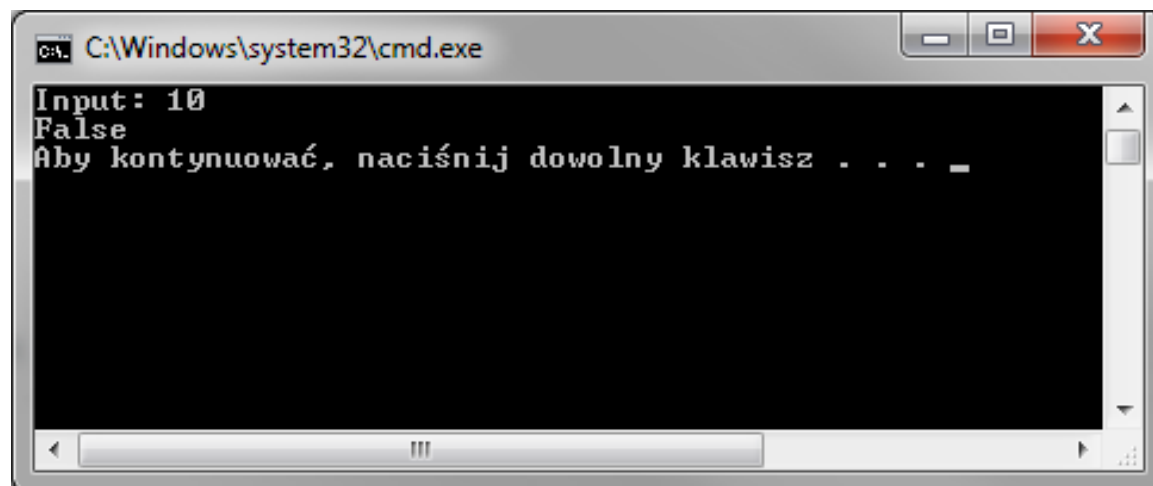
PIOTR CHRZCZONOWICZ

# Dlaczego nie piszemy testów

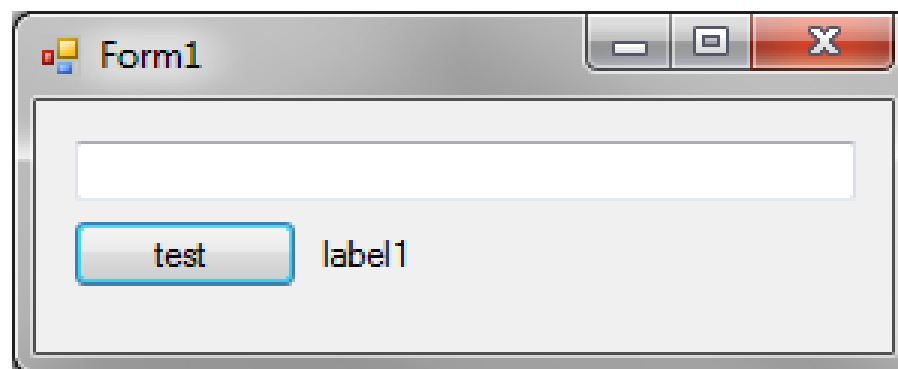
---

- Testy są trudne do napisania
- Testy są trudne do utrzymania
- Zajmują czas
- Manager nie widzi potrzeby
- itd...

Czy na pewno nie piszemy testów?



```
C:\Windows\system32\cmd.exe
Input: 10
False
Aby kontynuować, naciśnij dowolny klawisz . . . _
```



Form1

test label1

# To po co nam te testy (jednostkowe)?

---

- Pozwalają zweryfikować czy kod działa zgodnie z założeniami
  - „Mój kod działa.”
- Pozwalają ustrzec się od regresów
- **Aby pisać lepszy kod i szybciej iść do domu**

# Demo

---

- Pierwszy test - klasy bez zależności
- Wzorzec AAA
- NUnit

# Dobry test jednostkowy

---

- Prosty (zrozumiały)
- Szybki
- Powtarzalny
- Dobrze nazwany

PrimeFinderTests (5)		
✓	Returns_False_For_Negative_One	10 ms
✓	Returns_False_For_One	< 1 ms
✓	Returns_False_For_Twelve	1 ms
✓	Returns_True_For_Three	< 1 ms

# Dobry test jednostkowy

---

- **Arrange, Act, Assert**
- **Zawsze zawiera asercje**
- (Zazwyczaj) jedna asercja w jednym teście
- **Pełna izolacja**

# Demo

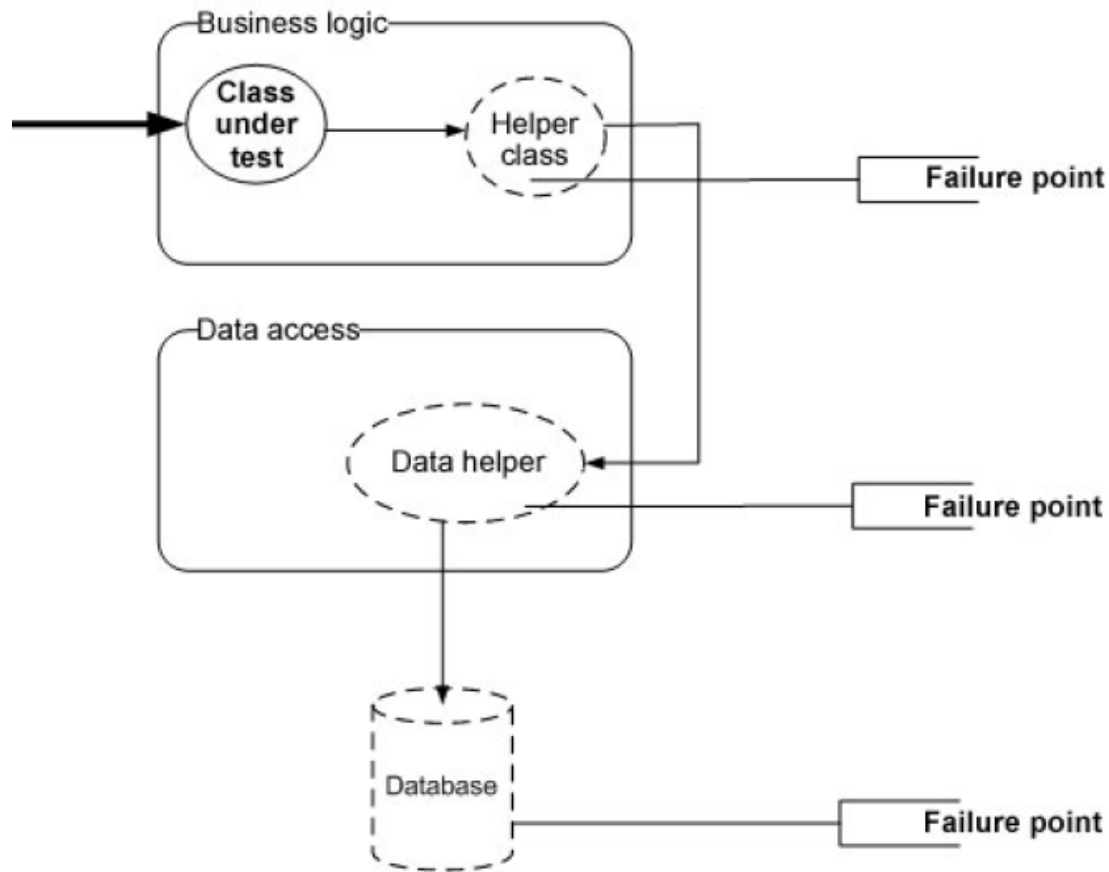
---

- Test klasy z zewnętrznymi zależnościami
- Przykład użycia NSubstitute



# Testy jednostkowe a integracyjne

---



# Demo

---

-Czemu Singleton jest zły

# A co z istniejącym kodem

---

- Łatwo jest testować nowe klasy
- „To nie zadziała w moim projekcie.”
- „Nie zamierzam przepisywać starego kodu, czy naprawdę nie ma sposobu na testowanie?”

# Kiedy nie pisać testów?

---

- Kiedy nie mamy jeszcze doświadczenia a istniejący kod:
  - Woła metody statyczne
  - Zależy od singletonów
  - Odpytuje bazę danych, zapisuje do pliku, etc.
  - Wymaga ogromnej ilości przygotowań (tworzenia instancji wielu obiektów, mocków, ...)
- Nowy kod testować zawsze\*

# Testowanie istniejącego kodu

---

- **Refaktoryzacja**
- Alternatywnie:
  - Testy integracyjne
  - *Typemock Isolator, Telerik JustMock (płatne)*
  - *Microsoft Fakes (VS  $\geq$  Premium)*
  - Prig

# Demo

---

- Shims (Microsoft Fakes)
- Dostęp do systemu plików

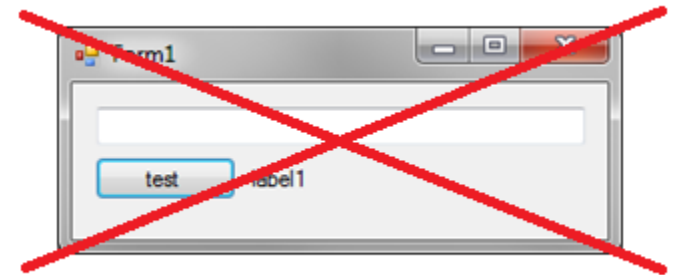
# Dobre praktyki

---

# Pisz testy

---

- Najlepszym sposobem na naukę testowania jest pisanie testów
- Warto korzystać z testów jednostkowych wszędzie gdzie się da:
  - Testowanie koncepcji
  - Szkice algorytmów
  - Sprawdzanie konkretnych przypadków





# Dobierz odpowiednie narzędzia

---

- Testowanie jest modne więc powstała masa narzędzi
- Warto wybrać te, z którymi pracuje się najwygodniej i poznać ich możliwości
- [Nuget.org](https://nuget.org)

# Nazywaj testy

---

- Nazwy powinny być zrozumiałe dla zespołu
- Warto przyjąć jakąś konwencję

```
[Test]
```

```
0 references
```

```
public void GetCustomerAddress_Returns_Null_When_Customer_Not_Selected_On_Invoice()  
{  
    //Arrange  
    //Act  
    //Assert  
}
```

# Nie testuj każdego przypadku

---

- Warto określić interesujące klasy parametrów i testować pojedyncze przypadki wchodzące w skład danej klasy
- Testy powinny sprawdzać wszystkie możliwe ścieżki, ale nie więcej

# Poznaj skróty klawiszowe

---

- Przydatne nie tylko przy testowaniu
- Im mniej czasu zajmuje uruchomienie testów, tym częściej mogą być uruchamiane

# Czy potrzebne nam testy?

---

---

## Literatura:

- Roy Osherove „The Art of Unit Testing. With Examples in .NET”
- Michael Feathers „Praca z zastanym kodem. Najlepsze techniki”
- Blogi praktyków (np. <http://www.daedtech.com/tag/unit-testing>)

## Materiały:

- Slajdy i kod dostępne po wypełnieniu ankiety:

<https://goo.gl/rB1abJ>



# Po co nam testy?

---

TESTOWANIE KODU W PRAKTYCE

PIOTR CHRZCZONOWICZ

PIOTR.CHRZCZONOWICZ@COMARCH.PL