# Property based testing v praxi

Marek Třešňák

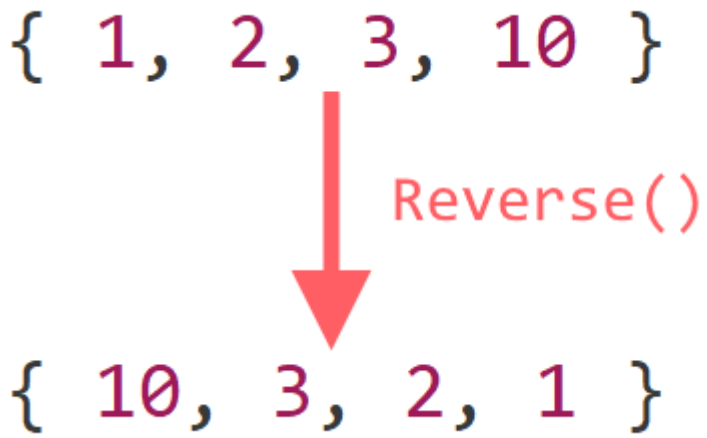
# Automatické testování kódu

- Testy jdou pouštět rychle.
- Chrání nás proti chybám, které už jsme dřív udělali.
- Nezapomenou projít některé scénáře.
- Slouží jako dokumentace.

# Example based testing

1. Vymyslíme si konkrétní vstupní hodnoty.
2. Nadefinujeme očekáváné výstupní hodnoty.
3. Spustíme testovaný kód a porovnáme hodnoty.

# Příklad - List Reverse



# Test s kolekcí čísel

```
[Test]
public void ReversalOfListOfIntsWorks()
{
    var list = new List<int> { 1, 2, 3, 10 };
    var reversedList = Reverse(list);
    CollectionAssert.AreEqual(
        expected: new List<int> { 10, 3, 2, 1 },
        actual: reversedList
    );
}
```

# + Test s prázdnou kolekcí

```
[Test]
public void ReversalOfListOfIntsWorks()
{
    var list = new List<int> { 1, 2, 3, 10 };
    var reversedList = Reverse(list);
    CollectionAssert.AreEqual(
        expected: new List<int> { 10, 3, 2, 1 },
        actual: reversedList
    );
}

[Test]
public void ReversalOfEmptyListOfIntsWorks()
{
    var list = new List<int>();
    var reversedList = Reverse(list);
    CollectionAssert.AreEqual(
        expected: new List<int>(),
        actual: reversedList
    );
}
```

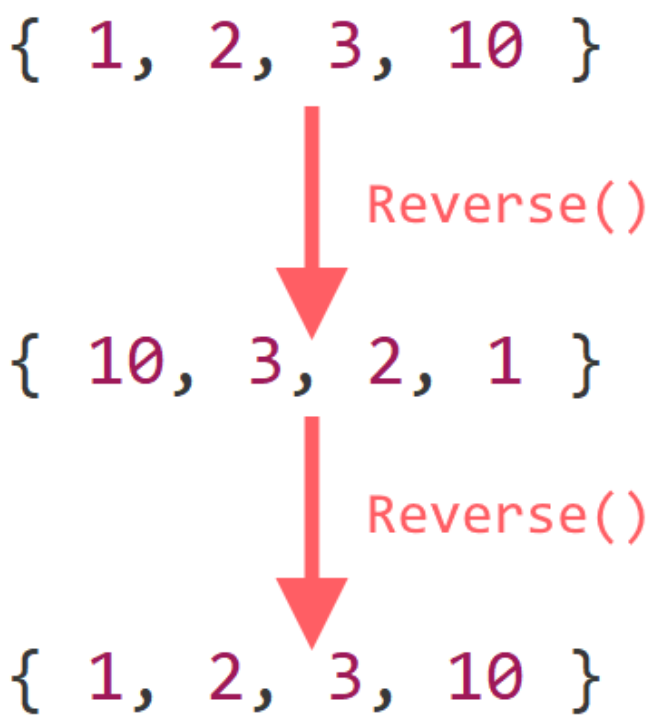
# Refactoring

```
[Test]
public void ReversalOfListOfIntsWorks()
{
    TestListReversal(
        list: new List<int> { 1, 2, 3, 10 },
        expected: new List<int> { 10, 3, 2, 1 }
    );
}

[Test]
public void ReversalOfEmptyListOfIntsWorks()
{
    TestListReversal(
        list: new List<int>(),
        expected: new List<int>()
    );
}

2 usages
private void TestListReversal<T>(List<T> list, List<T> expected)
{
    var reversedList = Reverse(list);
    CollectionAssert.AreEqual(
        expected: expected,
        actual: reversedList
    );
}
```

# Property based testing

1. Nadefinujeme množinu vstupních hodnot.
2. Nadefinujeme očekáváné vlastnosti výstupu.
3. Spustíme test s náhodně generovanými hodnotami a ověřujeme vlastnosti.

# Reverse(Reverse(list)) = list



# Test s “manuálně” generovanými hodnotami

```
[Test]
public void DoubleReversalOfListReturnsOriginalList()
{
    var random = new Random();
    for (var i = 0; i < 100; i++)
    {
        var list =
            Enumerable.Range(0, random.Next(1000))
            .Select(_ => random.Next())
            .ToList();
        var reversedList = Reverse(list);
        var doubleReverseList = Reverse(reversedList);
        CollectionAssert.AreEqual(
            expected: list,
            actual: doubleReverseList
        );
    }
}
```

# Test s “manuálně” generovanými hodnotami

```
DoubleReversalOfListReturnsOriginalList (.NETCoreApp,Version=v3.1) [34 ms] System.ArgumentOutOfRangeException : Index was out of range. Must be non-negative and less than the size of the collection. (Parameter 'index')
PropertyBasedTesting.ListReverse.DoubleReversalOfListReturnsOriginalList

System.ArgumentOutOfRangeException : Index was out of range. Must be non-negative and less
 than the size of the collection. (Parameter 'index')
   at System.Collections.Generic.List`1.get_Item(Int32 index)
   at PropertyBasedTesting.ListReverse.Reverse[T](List`1 input) in
 C:\Users\marek\OneDrive\mews\prezentace\property-based-testing\samples
 \PropertyBasedTesting\PropertyBasedTesting\ListReverse.cs:line 18
   at PropertyBasedTesting.ListReverse.DoubleReversalOfListReturnsOriginalList() in
 C:\Users\marek\OneDrive\mews\prezentace\property-based-testing\samples
 \PropertyBasedTesting\PropertyBasedTesting\ListReverse.cs:line 71
```

# FsCheck

# Test s použitím FsChecku

```
[Test]
public void DoubleReversalOfListReturnsOriginalList()
{
    Prop.ForAll<List<int>>(list =>
    {
    });
}
```

# Test s použitím FsChecku

```
[Test]
public void DoubleReversalOfListReturnsOriginalList()
{
    Prop.ForAll<List<int>>(list =>
    {
    });
}
```

```
Output (.NETCoreApp,Version=v3.1) [173 ms]
97:
seq [0; 0; 0; 0; ...]
98:
seq [3; 0; -3; -5; ...]
99:
seq [0; 1; 34; 0; ...]
```

# Test s použitím FsChecku

```
[Test]
public void DoubleReversalOfListReturnsOriginalList()
{
    Prop.ForAll<List<int>>(list =>
    {
        var reversedList = Reverse(list);
        var doubleReversedList = Reverse(reversedList);
        return list.SequenceEqual(doubleReversedList);
    });
}
```

# Test s použitím FsChecku

```
[Test]
public void DoubleReversalOfListReturnsOriginalList()
{
    Prop.ForAll<List<int>>(list =>
    {
        var reversedList = Reverse(list);
        var doubleReversedList = Reverse(reversedList);
        return list.SequenceEqual(doubleReversedList);
    }).QuickCheckThrowOnFailure();
}
```

# Test s použitím FsChecku

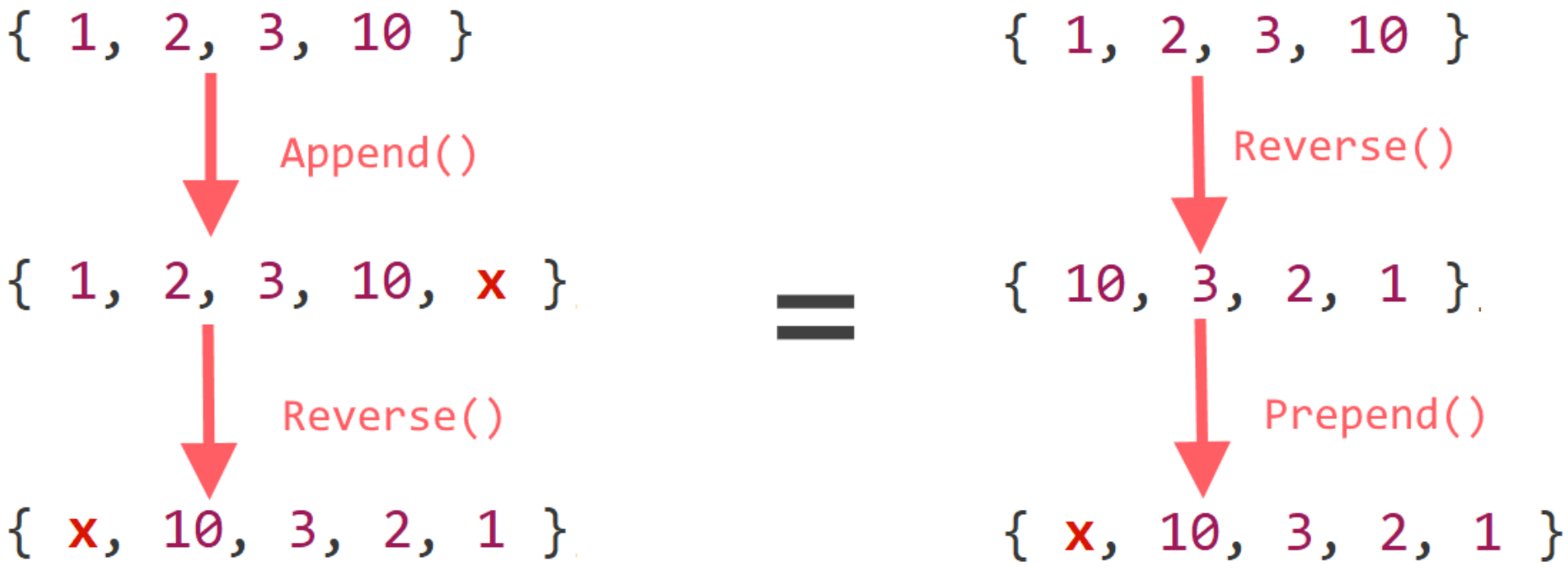
```
[Test]
public void DoubleReversalOfListReturnsOriginalList()
{
    Prop.ForAll<List<int>>(list =>
    {
        var reversedList = Reverse(list);
        var doubleReversedList = Reverse(reversedList);
        return list.SequenceEqual(doubleReversedList);
    }).QuickCheckThrowOnFailure();
}
```

# Implementace Reverse()

```
private List<T> Reverse<T>(List<T> input)
{
    return input;
}
```

✔ DoubleReversalOfListReturnsOriginalList [173 ms] Success

# Reverse(Append) = Prepend(Reverse)



# Několik generovaných hodnot

```
[Test]
public void AppendThenReverseIsEqualToReverseThenPrepend()
{
    Prop.ForAll(Arb.From<int>(), Arb.From<List<int>>(), (element, list) =>
    {
    });
}
```

# Test Reverse(Append) = Prepend(Reverse)

```
[Test]
public void AppendThenReverseIsEqualToReverseThenPrepend()
{
    Prop.ForAll(Arb.From<int>(), Arb.From<List<int>>(), (element, list) =>
    {
        var appendThenReverse = Reverse(Append(list, element));
        var reverseThenPrepend = Prepend(Reverse(list), element);
        return appendThenReverse.SequenceEqual(reverseThenPrepend);
    }).QuickCheckThrowOnFailure();
}
```

# Test Reverse(Append) = Prepend(Reverse)

```
[Test]
public void AppendThenReverseIsEqualToReverseThenPrepend()
{
    Prop.ForAll(Arb.From<int>(), Arb.From<List<int>>(), (element, list) =>
    {
        var appendThenReverse = Reverse(Append(list, element));
        var reverseThenPrepend = Prepend(Reverse(list), element);
        return appendThenReverse.SequenceEqual(reverseThenPrepend);
    }).QuickCheckThrowOnFailure();
}
```

⛔ AppendThenReverseIsEqualToReverseThenPrepend [203 ms] Failed: System.Exception : Falsifiable

# Správná implementace Reverse()

```
private List<T> Reverse<T>(List<T> input)
{
    var output = new List<T>(input);
    output.Reverse();
    return output;
}
```

✔ AppendThenReverseIsEqualToReverseThenPrepend [130 ms] Success

✔ DoubleReversalOfListReturnsOriginalList [71 ms] Success

# Test Reverse(Append) = Prepend(Reverse)

```
AppendThenReverseIsEqualToReverseThenPrepend (.NETCoreApp,Version=v3.1) [199 ms] System.Exception : Falsifiable, after 3 tests (3 shrinks) (427708154167105 7,14413507071265457431)
PropertyBasedTesting.ListReverse.AppendThenReverseIsEqualToReverseThenPrepend

System.Exception : Falsifiable, after 3 tests (3 shrinks) (4277081541671057107,
 14413507071265457431)
Last step was invoked with size of 4 and seed of (8928640929221858429,
 1429398716816633359):
Original:
-3
seq [0]
Shrunk:
1
seq [0]
```

# Shrinker

```
AppendThenReverseIsEqualToReverseThenPrepend (.NETCoreApp,Version=v3.1) [199 ms] System.Exception : Falsifiable, after 3 tests (3 shrinks) (4277081541671057107,1441350707126545743 1)
PropertyBasedTesting.ListReverse.AppendThenReverseIsEqualToReverseThenPrepend

System.Exception : Falsifiable, after 3 tests (3 shrinks) (4277081541671057107,
 14413507071265457431)
Last step was invoked with size of 4 and seed of (8928640929221858429,
 1429398716816633359):
Original:
-3
seq [0]
Shrunk:
1
seq [0]
```

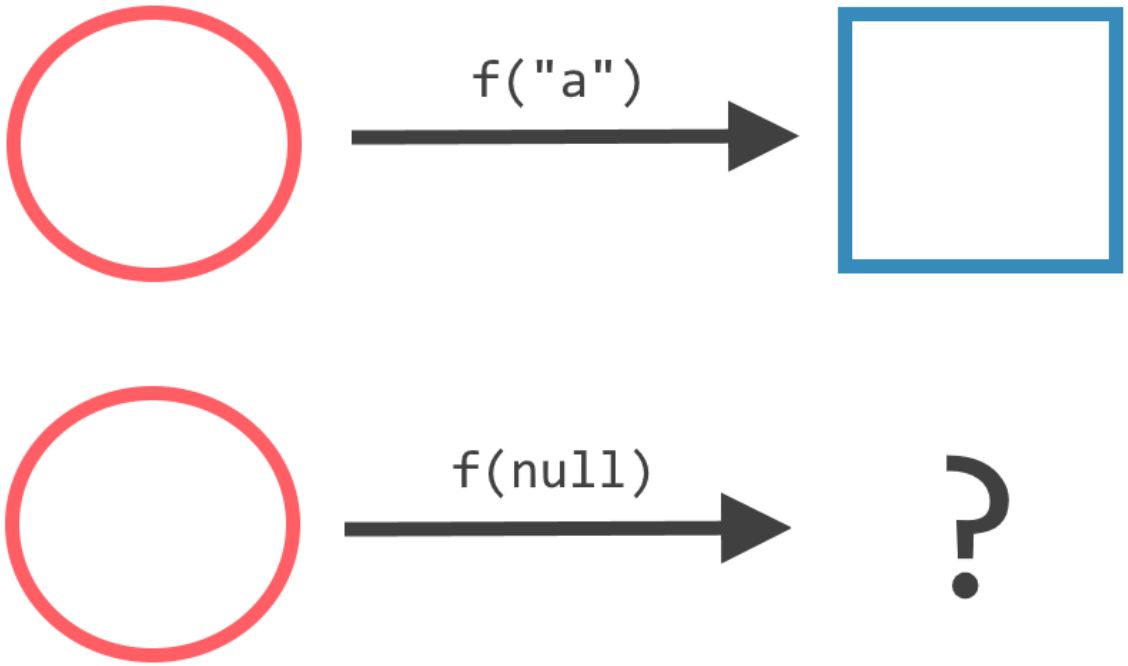
# Seed

```
AppendThenReverseIsEqualToReverseThenPrepend (.NETCoreApp,Version=v3.1) [199 ms] System.Exception : Falsifiable, after 3 tests (3 shrinks) (4277081541671057107,1441350707126545743 1)
PropertyBasedTesting.ListReverse.AppendThenReverseIsEqualToReverseThenPrepend

System.Exception : Falsifiable, after 3 tests (3 shrinks) (4277081541671057107,
 1441350707126545743 1)
Last step was invoked with size of 4 and seed of (8928640929221858429,
 1429398716816633359):
Original:
-3
seq [0]
Shrunk:
1
seq [0]
```

# Co vlastně testovat?

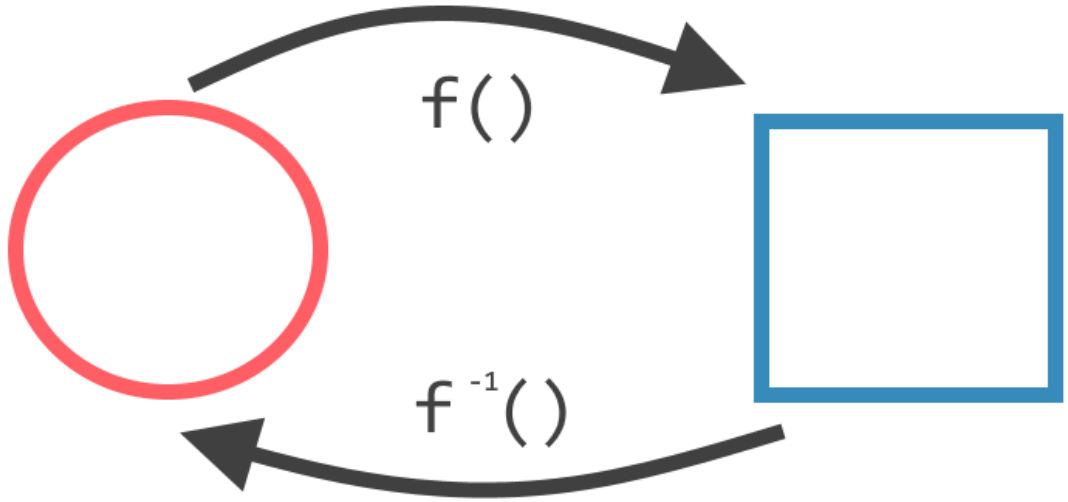
# Fuzz testování

- Kontrola, že kód nepadá při různých vstupech



# Inverzní funkce

- Encode / Decode
- Serialize / Deserialize
- Write / Read

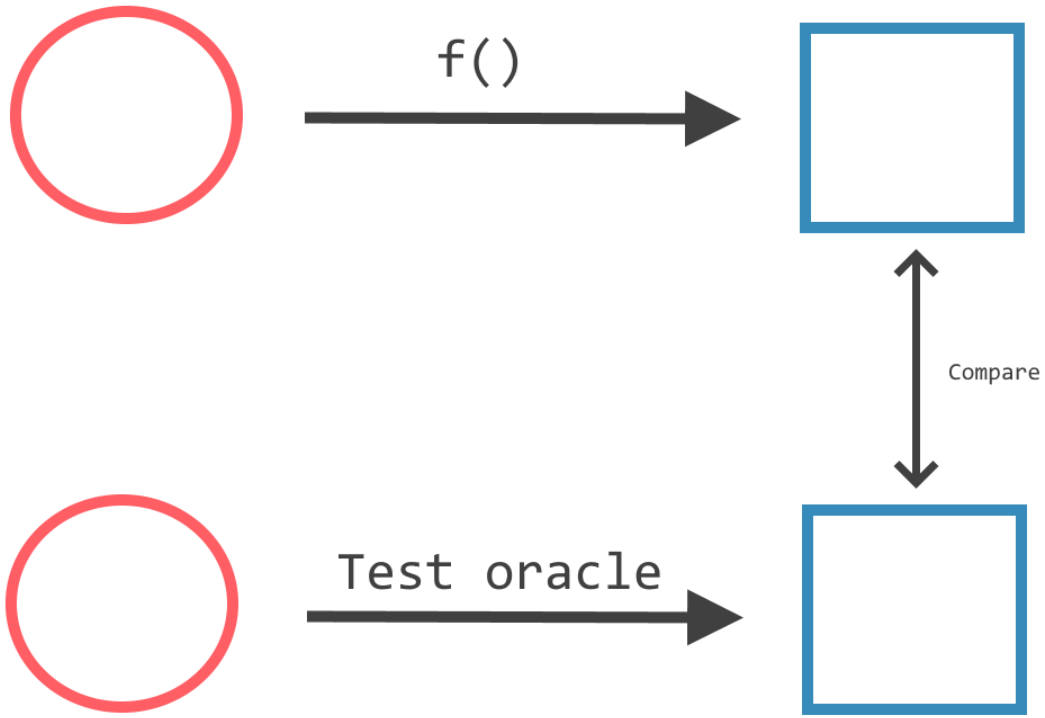


# Idempotence

- List.Distinct()
- String.Trim()

# Test oracle

- Nová vs stará implementace
- Naivní vs optimalizovaná implementace



# Shrnutí

- Testy se často hodí kombinovat.
- Example based testy jsou obvykle názornější.
- Jeden generativní test dokáže nahradit spoustu example based testů.
- Generativní test spíš odhalí krajní případy.

# Pár odkazů

- FSharpForFunAndProfit.com
- fscheck.github.io/FsCheck

# Dotazy?

- marek.tresnak@outlook.com
- github.com/MewsSystems/developers

MEWS