Hunting Bugs While Sleeping

Property-Based Testing with **C#**

Paul Amazona
Developer
@whatevergeek

DataKind


Django Against the Dark Arts
@attacus_au


pyconau
INTERNATIONAL CONVENTION CENTRE
SYDNEY
pyconau


Container image security – Notary
Image repository
Digitally signs images to be published.
The Update Framework (TUF) underlays the tool to maintain up-to-date software.
Allows user to verify content that they download.
Honourable mentions go to Clair


Polylingualism
INTERNATIONAL CONVENTION CENTRE
pyconau


Generative Tests
"I SPEND A LOT OF TIME writing tests
I SHOULD WRITE A PROGRAM AUTOMATING IT!"
WORK
WRITING CODE
WORK ON ORIGINAL TASK
AUTOMATION TAKES OVER
FREE TIME
TIME
Using a library lets you skip the "ongoing development" panel from the original comic
3Ai
Basically automating automated testing!
Hypothesis can generate...
arguments to a test function
entire test programs!
INTERNATIONAL CONVENTION CENTRE
SYDNEY
pyconau

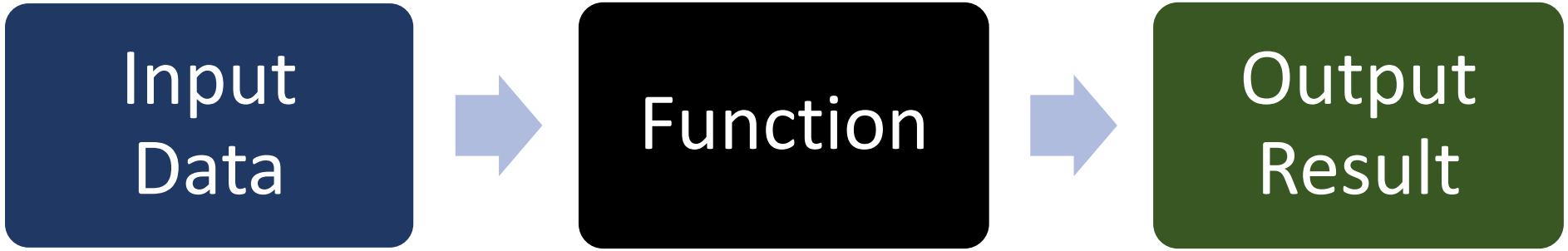

pyconau

Property-Based Testing with **Hypothesis** 

https://tinyurl.com/hypothesis-prezo

Property-Based Testing

A type of testing that asserts based on properties that describe the **relationship** between the **input** and **output** of the **function** being tested.



Testing the Multiply Function

```
[Fact]
public void TestMultiplyUsingExample1()
{
   int expected = 2*3;
   int actual = Arithmetic.Multiply(2, 3);

   Assert.Equal(expected, actual);
}
```

Testing the Multiply Function

Reimplementing

```
[Fact]
public void TestMultiplyUsingExample1()
{
    int expected = 2*3;
    int actual = Arithmetic.Multiply(2, 3);

    Assert.Equal(expected, actual);
}
```

Testing using Example Outputs

```
[Fact]
public void TestMultiplyUsingExample1()
{
    int expected = 6;
    int actual = Arithmetic.Multiply(2, 3);

    Assert.Equal(expected, actual);
}

[Fact]
public void TestMultiplyUsingExample2()
{
    int expected = 20;
    int actual = Arithmetic.Multiply(4, 5);

    Assert.Equal(expected, actual);
}
```

Parameterized Tests

```
[Theory]
[InlineData(2, 3, 6)]
[InlineData(4, 5, 20)]
public void TestMultiplyUsingExample1And2(int factor1, int factor2, int expected)
{
    int actual = Arithmetic.Multiply(factor1, factor2);

    Assert.Equal(expected, actual);
}
```

Multiplication Properties

- **Commutative property**

  When two numbers are multiplied together, the product is the same regardless of the order of the multiplicands.
  For example 4 * 2 = 2 * 4

- **Associative Property**

  When three or more numbers are multiplied, the product is the same regardless of the grouping of the factors.
  For example (2 * 3) * 4 = 2 * (3 * 4)

- **Multiplicative Identity Property**

  The product of any number and one is that number.
  For example 5 * 1 = 5.

- **Distributive property**

  The sum of two numbers times a third number is equal to the sum of each addend times the third number.
  For example 4 * (6 + 3) = 4*6 + 4*3

Property-Based Tests (PBT)

with Predetermined Inputs

```
[Theory]
[InlineData(2, 3)]
[InlineData(4, 5)]
public void TestMultiplyCommutativeProperty(int factor1, int factor2) =>
      Assert.Equal(Multiply(factor1, factor2), Multiply(factor2, factor1));
```

Property-Based Tests (PBT)

with Predetermined Inputs

```
public static IEnumerable<object[]> GetPredeterminedFactorList() =>
       Enumerable.Range(1, 100).Select(f => new object[] { f });

[Theory]
[MemberData(nameof(GetPredeterminedFactorList))]
public void TestMultiplyIdentityProperty(int factor) =>
       Assert.Equal(Multiply(factor, 1), factor);

[Theory]
[InlineData(2, 3, 4)]
[InlineData(4, 5, 6)]
public void TestMultiplyDistributiveProperty(int factor1, int factor2, int factor3) =>
    Assert.Equal(Multiply(factor1, (factor2 + factor3)),
    Multiply(factor1, factor2) + Multiply(factor1, factor3));
```

Property-Based Tests (PBT)

with Randomized Inputs

```
[Property]
public void TestMultiplyCommutativeProperty(int factor1, int factor2) =>
      Assert.Equal(Multiply(factor1, factor2), Multiply(factor2, factor1));
```

Property-Based Tests Libraries…

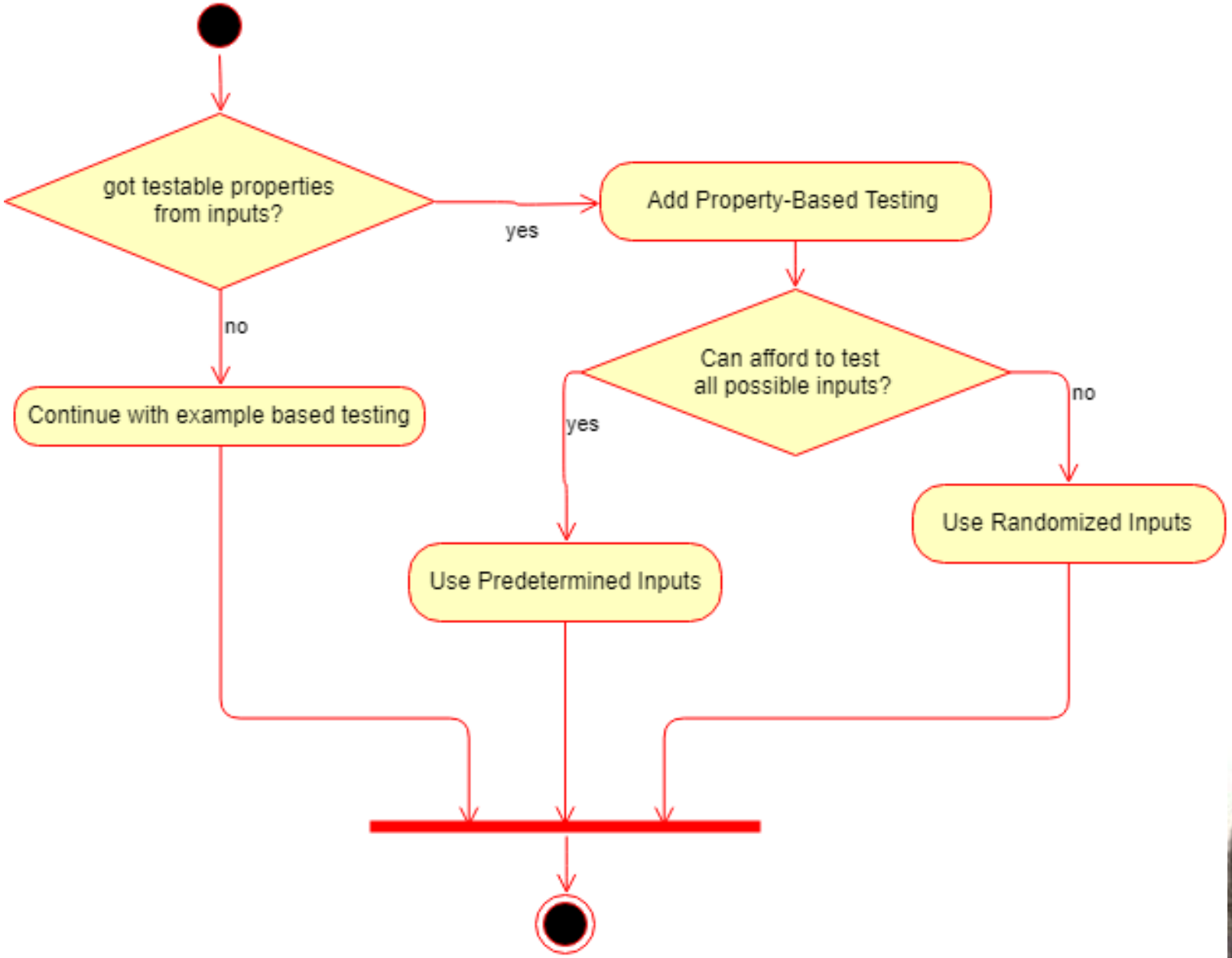
Language	PBT Libraries
python	hypothesis
.NET (C#, F#, etc)	FsCheck
haskell	quickcheck
java	junit-quickchek
javascript	fast-check
swift	SwiftCheck
scala	ScalaCheck

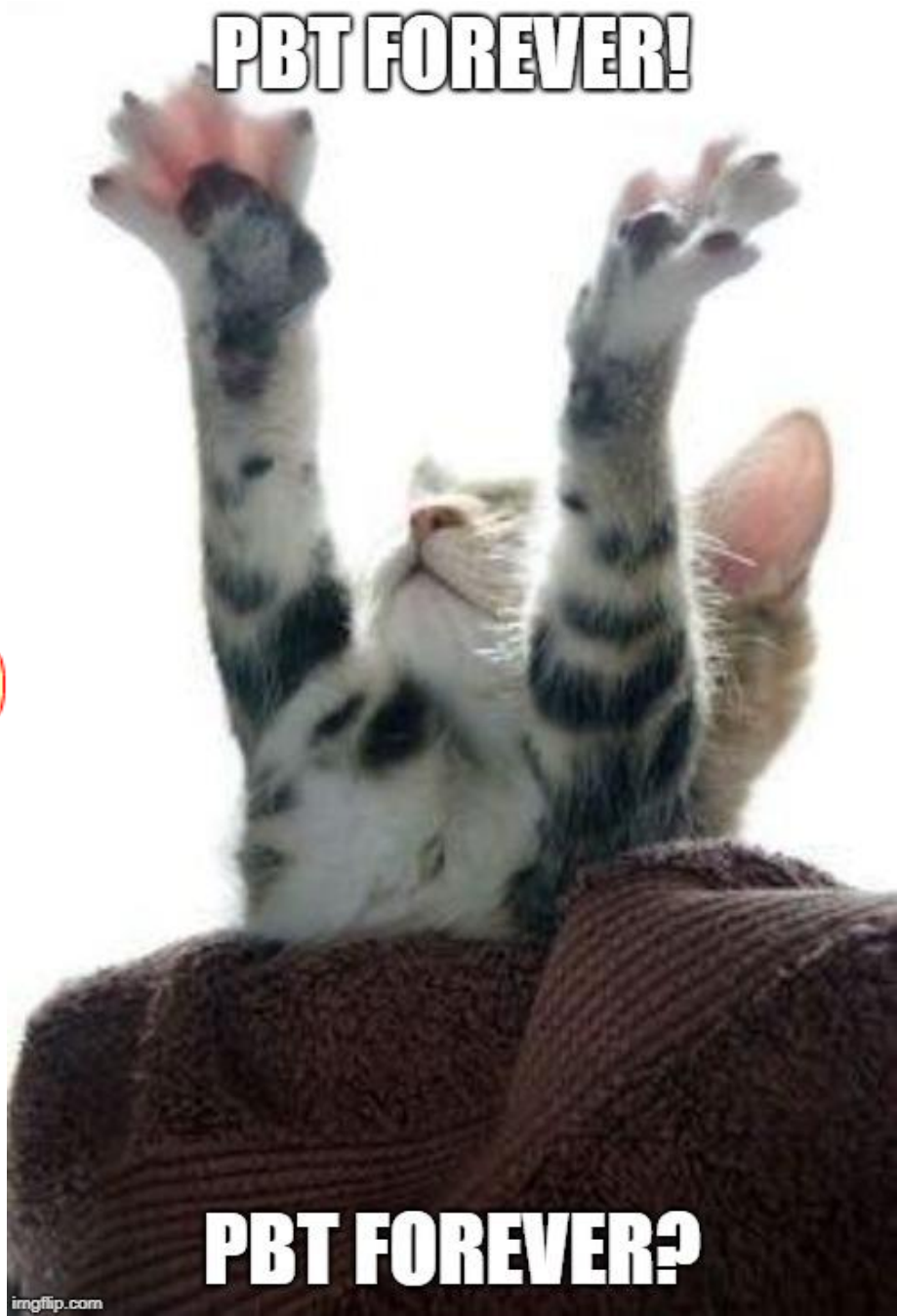
Setting Up a .NET PBT Project

```
<PackageReference Include="FsCheck.Xunit" Version="2.14.0" />
<PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.0.1" />
<PackageReference Include="xunit" Version="2.4.1" />
<PackageReference Include="xunit.runner.visualstudio" Version="2.4.1">
```

DEMO

- Create PBT Project
- Use Test Explorer
- Show failing scenario
- Adjust sample size
- CustomProperty


got testable properties from inputs?
yes
Add Property-Based Testing
no
Continue with example based testing
Can afford to test all possible inputs?
yes
no
Use Predetermined Inputs
Use Randomized Inputs


PBT FOREVER!
PBT FOREVER?
imgflip.com

Hunting Bugs with CI

Pipelines	Process	Characteristics
Normal CI	Build-> Tests with Predetermined Inputs	• Trigger: merge/PR/etc • More predictable duration • Purpose: Detect problems early
Bug Hunting CI	Build->Test with Randomized Inputs	• Trigger: Scheduled build (every hour, every day, etc). • Can take time depending on sample size of randomized inputs • Purpose: Hunt Bugs

Summary and Links

- Testing using Example Outputs
- Parameterized Tests
- Property-based Tests with Predetermined Inputs
- Property-based Tests with Randomized Inputs
- Bug Hunting CI Pipeline

Paul Amazona

@whatevergeek

Python Version of the Talk:
https://tinyurl.com/bughunt-python-pbt

C# Demo Source Code:
https://github.com/whatevergeek/csharp_netcore_pbt_demo