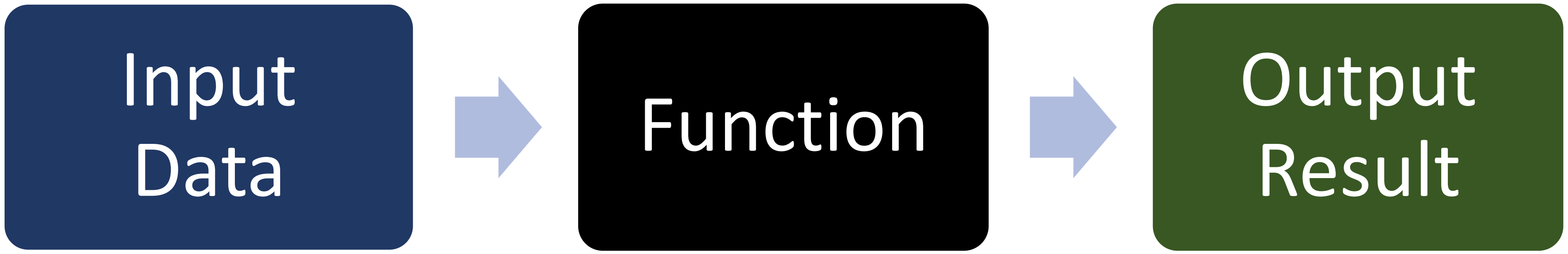
Hunting Bugs While Sleeping

Property-Based Testing with **Java**

Paul Amazona
Developer
@whatevergeek

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

```
public int Multiply(int x, int y) {
    return x * y;
}
```

Testing using Example Outputs

```
@Test
public void TestMultiplyUsingExample1() {
   int expected = 6;
   int actual = Library.Multiply(2, 3);
   assertEquals(expected, actual);
}

@Test
public void TestMultiplyUsingExample2() {
   int expected = 20;
   int actual = Library.Multiply(4, 5);
   assertEquals(expected, actual);
}
```

Parameterized Tests

```
@Test
@Parameters({ "2,3,6", "3,4,12" })
public void TestMultiplyUsingExample(int factor1, int factor2, int expected) {
   int actual = Library.Multiply(factor1, factor2);
   assertEquals(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

```
@Test
@Parameters({ "2,3", "3,4" })
public void TestMultiplyCommutativeProperty(int factor1, int factor2) {
       assertEquals(Library.Multiply(factor1, factor2), Library.Multiply(factor2, factor1));
}
```

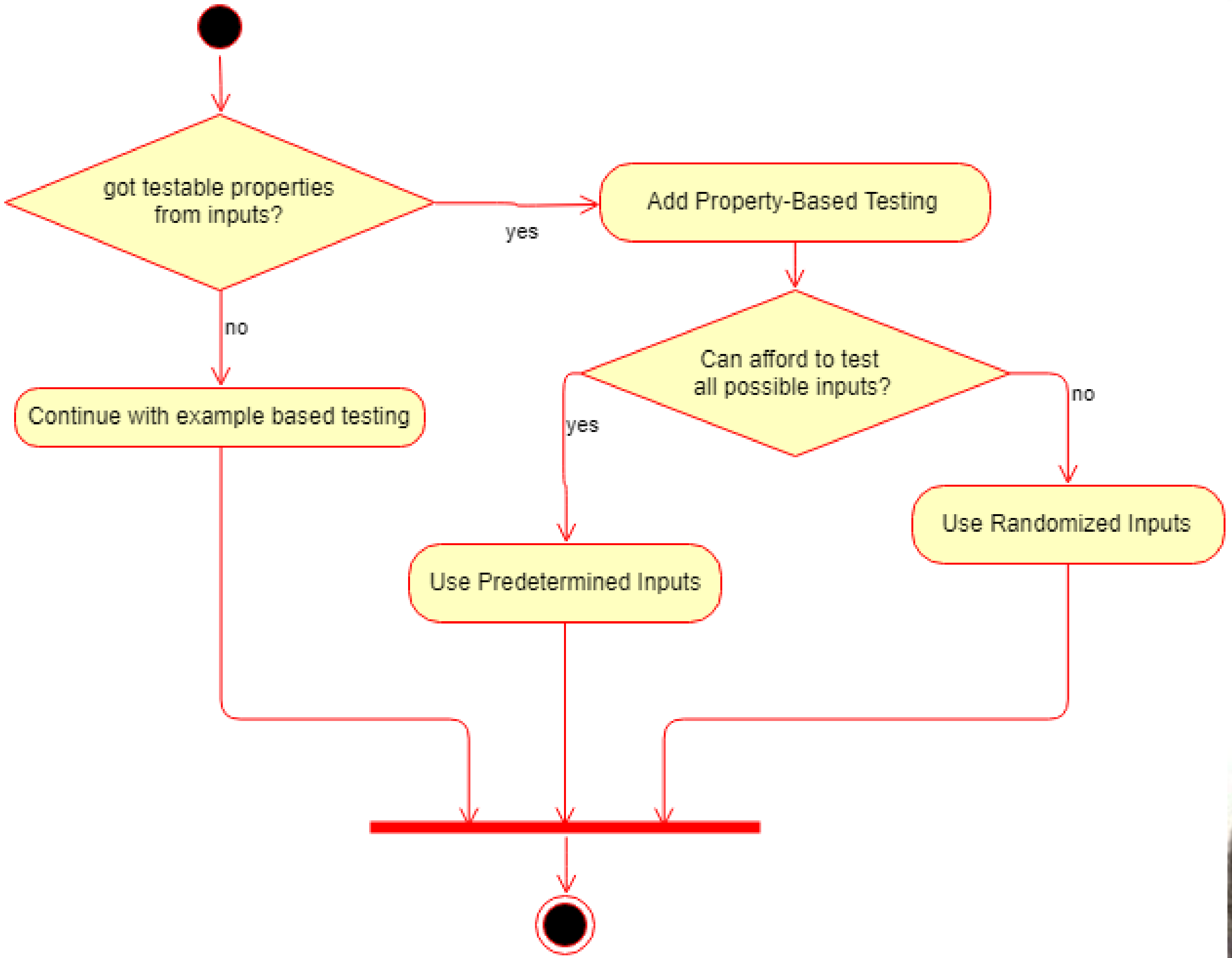
Property-Based Tests (PBT)

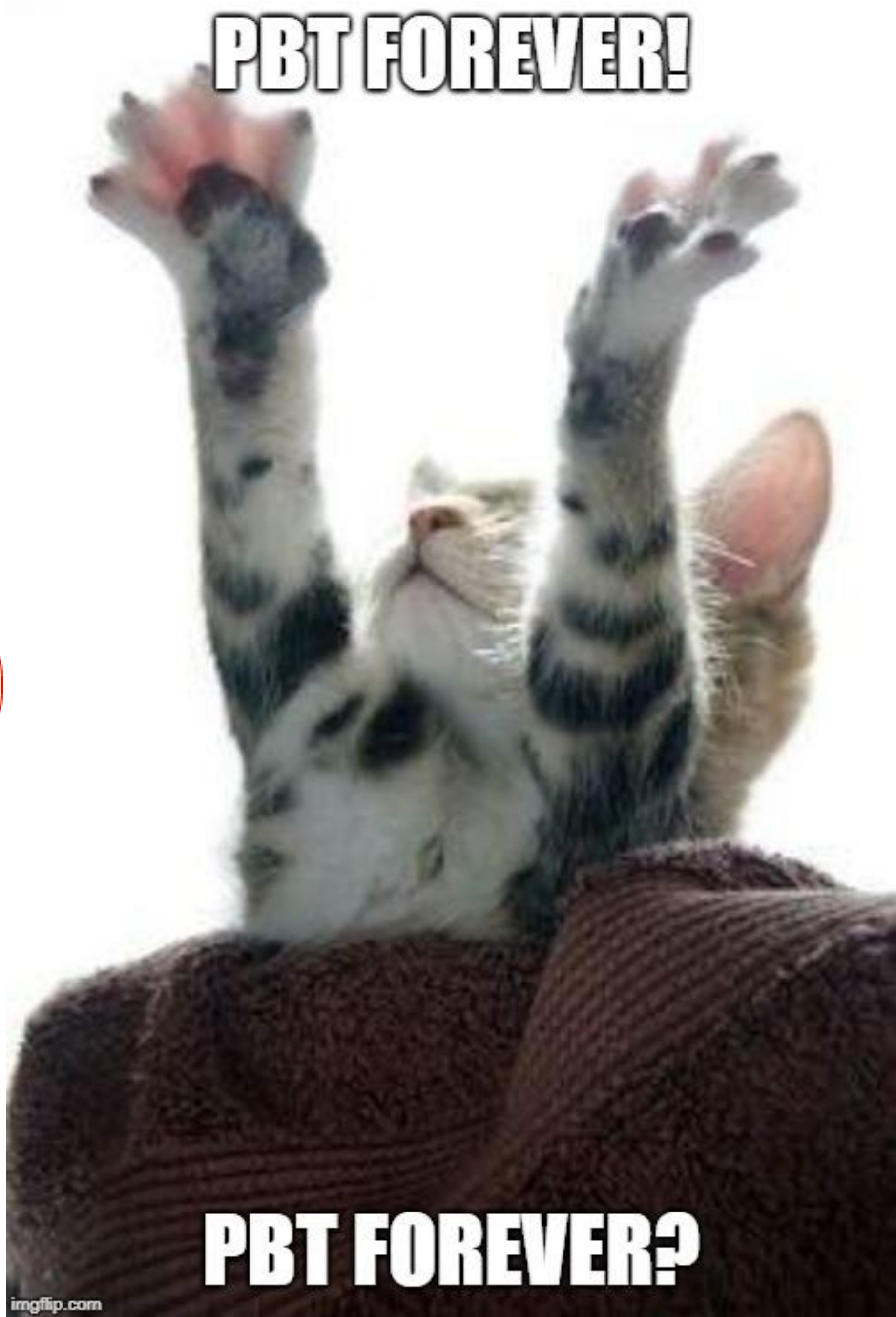
with Randomized Inputs

```
@Property(trials = 10)
public void TestCommutativeProperty(int factor1, int factor2) {
	assertEquals(Library.Multiply(factor1, factor2), Library.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


got testable properties from inputs?
yes
no
Add Property-Based Testing
Can afford to test all possible inputs?
yes
no
Continue with example based testing
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# Version of the Talk:
https://github.com/whatevergeek/csharp_netcore_pbt_demo

Java Demo Source Code:
https://github.com/whatevergeek/java_pbt_demo