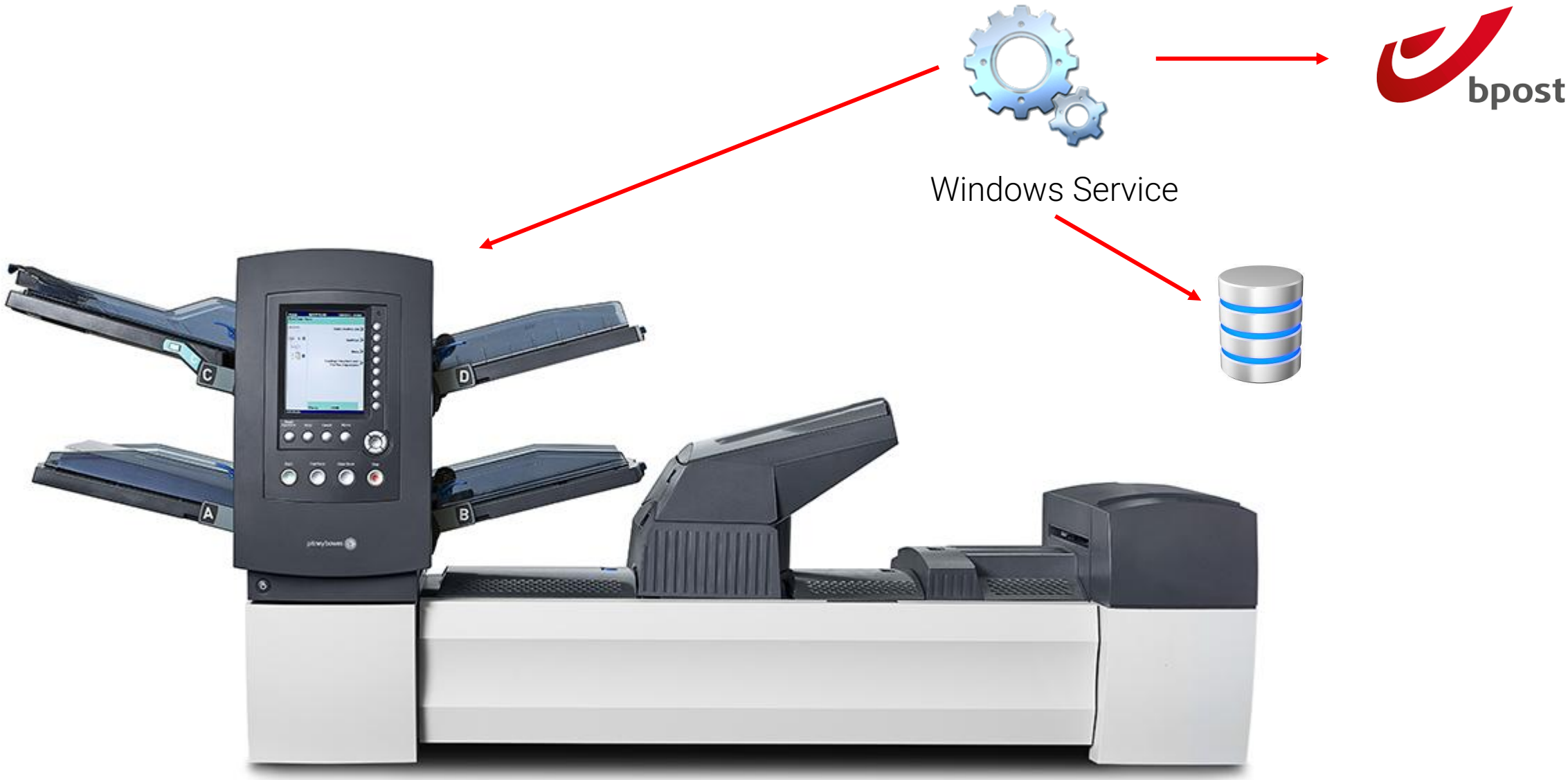
A gentle introduction into Property based Testing

Bart Wullems

https://bartwullems.blogspot.com

wullemsb@gmail.com

Some questions…


Windows Service
bpost
C
D
A
B

Cash Receipt Date:

Letters	0,04 € x 10K
Envelopes	0,12 € x 10K
Registered mail	2,00€ x 10K
Dev Work	60,00€ x 4h
Letters	0,04 € x 10K
Envelopes	0,12 € x 10K
Registered mail	2,00€ x 10K
Total	43.440,00 €
Cash	
Change	

Who am I?

- Now: Lead Architect @ Vlaamse Landmaatschappij
- Before: Technical Director @ Sopra Steria Belgium
- Regular blogger @ https://bartwullems.blogspot.com

Microsoft® Most Valuable Professional

Example Based Testing

Example based testing

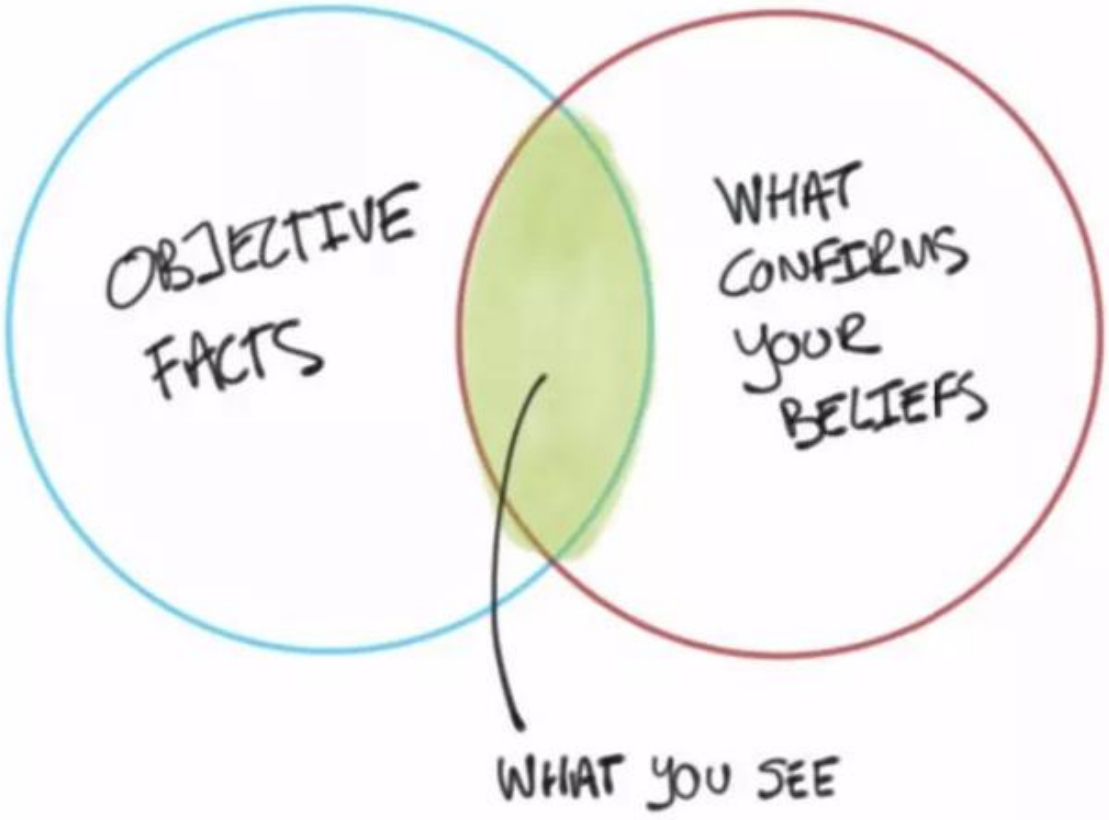
```
[Fact]
| 0 references | 0 changes | 0 authors, 0 changes
public void WhenIAdd1and2IExpect3()
{
    var result = Calculator.Add(1, 2);
    Assert.Equal(3, result);
}
```

Why 1 and 2?

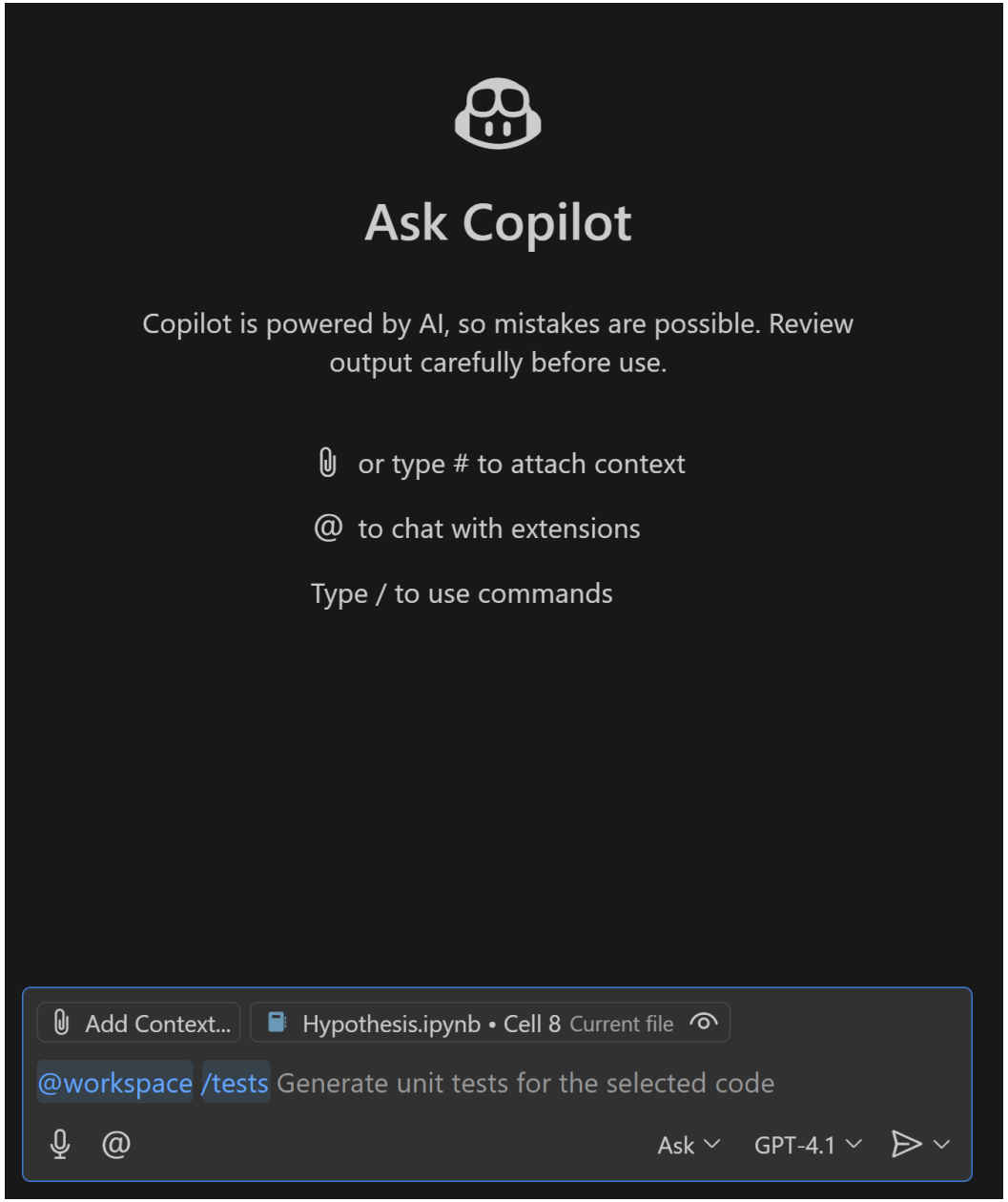
ORDINA
a Sopra steria company

Let’s write some tests…

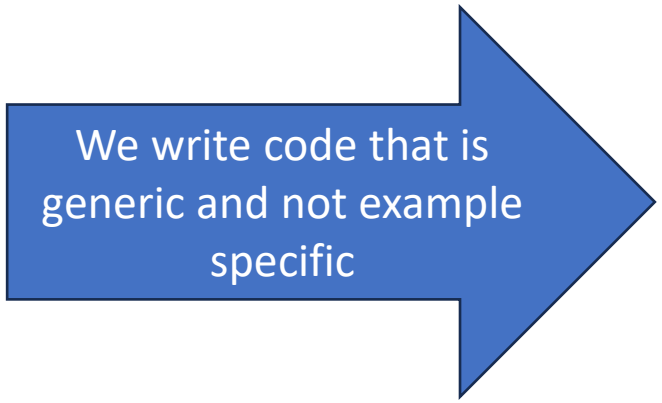
Example based testing

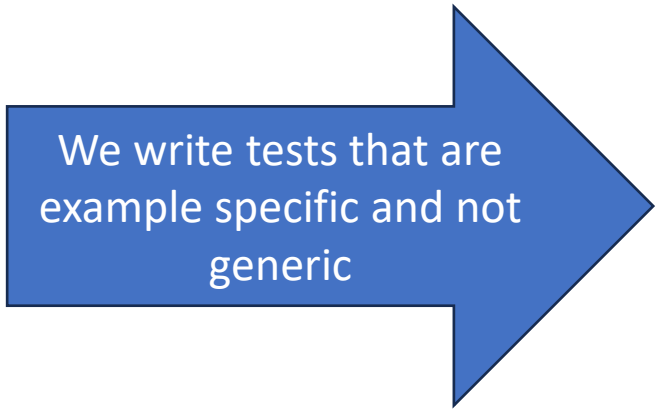


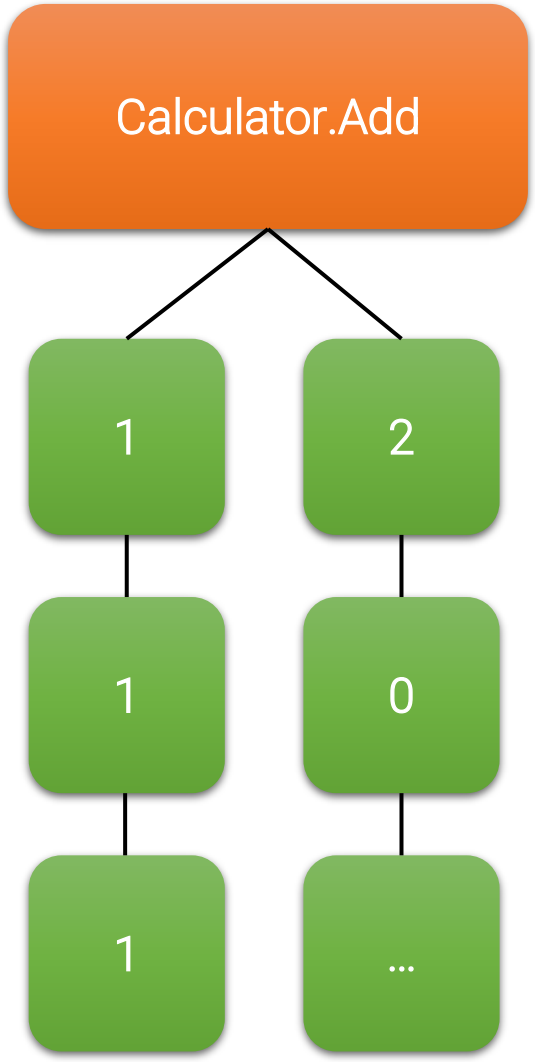
AI can help…


Ask Copilot
Copilot is powered by AI, so mistakes are possible. Review output carefully before use.
or type # to attach context
@ to chat with extensions
Type / to use commands
Add Context...
Hypothesis.ipynb • Cell 8 Current file
@workspace /tests Generate unit tests for the selected code
@
Ask
GPT-4.1

Root cause…


We write code that is generic and not example specific


We write tests that are example specific and not generic


Calculator.Add
1
2
1
0
1
…


Mutation testing
Property based testing

Property Based Testing

Property Based Testing

Property based testing allows you to:

- Specify what values to generate
- Assert on **properties** that are true regardless of the exact value

Property Based Testing

Property-based Testing is an automated testing technique where you incrementally zero in on the correct behavior of a system by describing its properties or qualities in general terms and then use randomly generated test data to execute deterministic tests.

What is a ‘property’?

Property ≈ Requirement

What is a ‘property’?

You don’t know the exact input

So you don’t know what the exact output should be

But you can describe a relationships between input and output

What are the properties of the ‘add’ function?

ORDINA
a Sopra steria company

Properties?

```
[Fact]
| 0 references | 0 changes | 0 authors, 0 changes
public void WhenIAddTwoRandomNumbersTheResultShouldNotDependOnParameterOrder()
{
    for (int i = 0; i < 100; i++)
    {
        //Arrange
        var random = new Random();
        int input1 = random.Next();
        int input2 = random.Next();

        //Act
        var result1 = Calculator.Add(input1, input2);
        var result2 = Calculator.Add(input2, input1);

        //Assert
        Assert.Equal(result1, result2);
    }
}
```

Properties

```
[Fact]
| 0 references | 0 changes | 0 authors, 0 changes
public void WhenIAdd1TwiceTheResultIsTheSameAsWhenAdding2()
{
    for (int i = 0; i < 100; i++)
    {
        //Arrange
        var random = new Random();
        int input = random.Next();

        //Act
        var result1 = Calculator.Add(Calculator.Add(input,1),1);
        var result2 = Calculator.Add(input, 2);

        //Assert
        Assert.Equal(result1, result2);
    }
}
```

Properties

```
[Fact]
| 0 references | 0 changes | 0 authors, 0 changes
public void WhenIAddZeroTheInputIsNotChanged()
{
    for (int i = 0; i < 100; i++)
    {
        //Arrange
        var random = new Random();
        int input = random.Next();

        //Act
        var result1 = Calculator.Add(input,0);
        var result2 = input;

        //Assert
        Assert.Equal(result1, result2);
    }
}
```

Properties

When I add 2 random numbers, the result should not depend on parameter order

Properties [edit]

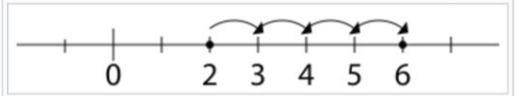
jump" that has a distance of 2 followed by another that is long as 4, is the same as a translation by 6.

Commutativity [edit]

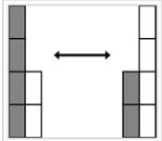
Addition is commutative, meaning that one can change the order of the terms in a sum, but still get the same result. Symbolically, if a and b are any two numbers, then

$$a + b = b + a.$$

The fact that addition is commutative is known as the "commutative law of addition" or "commutative property of addition". Some other binary operations are commutative, such as multiplication, but many others are not, such as subtraction and division.

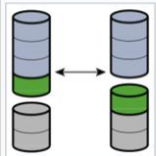


A number-line visualization of the unary addition 2 + 4 = 6. A translation by 4 is equivalent to four translations by 1.



4 + 2 = 2 + 4 with blocks

Associativity [edit]



2 + (1 + 3) = (2 + 1) + 3 with segmented rods

Addition is associative, which means that when three or more numbers are added together, the order of operations does not change the result.

As an example, should the expression $a + b + c$ be defined to mean $(a + b) + c$ or $a + (b + c)$? Given that addition is associative, the choice of definition is irrelevant. For any three numbers a, b, and c, it is true that $(a + b) + c = a + (b + c)$. For example, $(1 + 2) + 3 = 3 + 3 = 6 = 1 + 5 = 1 + (2 + 3)$.

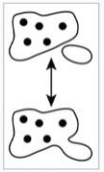
When addition is used together with other operations, the order of operations becomes important. In the standard order of operations, addition is a lower priority than exponentiation, nth roots, multiplication and division, but is given equal priority to subtraction.[21]

Identity element [edit]

Adding zero to any number, does not change the number; this means that zero is the identity element for addition, and is also known as the additive identity. In symbols, for every a, one has

$$a + 0 = 0 + a = a.$$

This law was first identified in Brahmagupta's *Brahmasphutasiddhanta* in 628 AD, although he wrote it as three separate laws, depending on whether a is negative, positive, or zero itself, and he used words rather than algebraic symbols. Later Indian mathematicians refined the concept; around the year 830, Mahavira wrote, "zero becomes the same as what is added to it", corresponding to the unary statement $0 + a = a$. In the 12th century, Bhaskara wrote, "In the addition of cipher, or subtraction of it, the quantity, positive or negative, remains the same", corresponding to the unary statement $a + 0 = a$.[22]



5 + 0 = 5 with bags of dots

Properties

When I add 1 twice, the result is the same as when adding 2

Properties [edit]

Commutativity [edit]

Addition is commutative, meaning that one can change the order of the terms in a sum, but still get the same result. Symbolically, if *a* and *b* are any two numbers, then

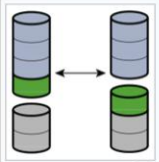
$$a + b = b + a.$$

The fact that addition is commutative is known as the "commutative law of addition" or "commutative property of addition". Some other binary operations are commutative, such as multiplication, but many others are not, such as subtraction and division.

"jump" that has a distance of 2 followed by another that is long as 4, is the same as a translation by 6.

A number-line visualization of the unary addition 2 + 4 = 6. A translation by 4 is equivalent to four translations by 1.

Associativity [edit]

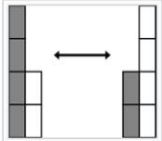


2 + (1 + 3) = (2 + 1) + 3 with segmented rods

Addition is associative, which means that when three or more numbers are added together, the order of operations does not change the result.

As an example, should the expression $a + b + c$ be defined to mean $(a + b) + c$ or $a + (b + c)$? Given that addition is associative, the choice of definition is irrelevant. For any three numbers *a*, *b*, and *c*, it is true that $(a + b) + c = a + (b + c)$. For example, $(1 + 2) + 3 = 3 + 3 = 6 = 1 + 5 = 1 + (2 + 3)$.

When addition is used together with other operations, the order of operations becomes important. In the standard order of operations, addition is a lower priority than exponentiation, nth roots, multiplication and division, but is given equal priority to subtraction.[21]



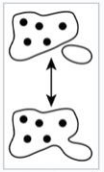
4 + 2 = 2 + 4 with blocks

Identity element [edit]

Adding zero to any number, does not change the number; this means that zero is the identity element for addition, and is also known as the additive identity. In symbols, for every a, one has

$$a + 0 = 0 + a = a.$$

This law was first identified in Brahmagupta's *Brahmasphutasiddhanta* in 628 AD, although he wrote it as three separate laws, depending on whether *a* is negative, positive, or zero itself, and he used words rather than algebraic symbols. Later Indian mathematicians refined the concept; around the year 830, Mahavira wrote, "zero becomes the same as what is added to it", corresponding to the unary statement $0 + a = a$. In the 12th century, Bhaskara wrote, "In the addition of cipher, or subtraction of it, the quantity, positive or negative, remains the same", corresponding to the unary statement $a + 0 = a$.[22]



5 + 0 = 5 with bags of dots

Properties

When I add 0, the input is not changed

Properties [edit]

"jump" that has a distance of 2 followed by another that is long as 4, is the same as a translation by 6.

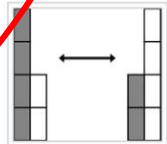
A number-line visualization of the unary addition 2 + 4 = 6. A translation by 4 is equivalent to four translations by 1.

Commutativity [edit]

Addition is commutative, meaning that one can change the order of the terms in a sum, but still get the same result. Symbolically, if *a* and *b* are any two numbers, then

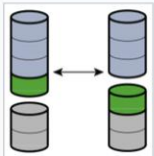
$$a + b = b + a.$$

The fact that addition is commutative is known as the "commutative law of addition" or "commutative property of addition". Some other binary operations are commutative, such as multiplication, but many others are not, such as subtraction and division.



4 + 2 = 2 + 4 with blocks

Associativity [edit]



2 + (1 + 3) = (2 + 1) + 3 with segmented rods

Addition is associative, which means that when three or more numbers are added together, the order of operations does not change the result.

As an example, should the expression $a + b + c$ be defined to mean $(a + b) + c$ or $a + (b + c)$? Given that addition is associative, the choice of definition is irrelevant. For any three numbers a, b, and c, it is true that $(a + b) + c = a + (b + c)$. For example, $(1 + 2) + 3 = 3 + 3 = 6 = 1 + 5 = 1 + (2 + 3)$.

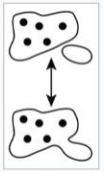
When addition is used together with other operations, the order of operations becomes important. In the standard order of operations, addition is a lower priority than exponentiation, nth roots, multiplication and division, but is given equal priority to subtraction.[21]

Identity element [edit]

Adding zero to any number, does not change the number; this means that zero is the identity element for addition, and is also known as the additive identity. In symbols, for every a, one has

$$a + 0 = 0 + a = a.$$

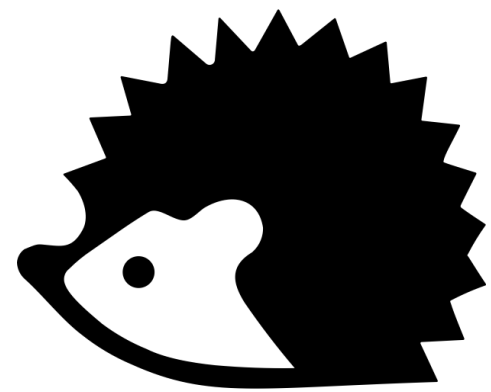
This law was first identified in Brahmagupta's *Brahmasphutasiddhanta* in 628 AD, although he wrote it as three separate laws, depending on whether *a* is negative, positive, or zero itself, and he used words rather than algebraic symbols. Later Indian mathematicians refined the concept; around the year 830, Mahavira wrote, "zero becomes the same as what is added to it", corresponding to the unary statement $0 + a = a$. In the 12th century, Bhaskara wrote, "In the addition of cipher, or subtraction of it, the quantity, positive or negative, remains the same", corresponding to the unary statement $a + 0 = a$.[22]



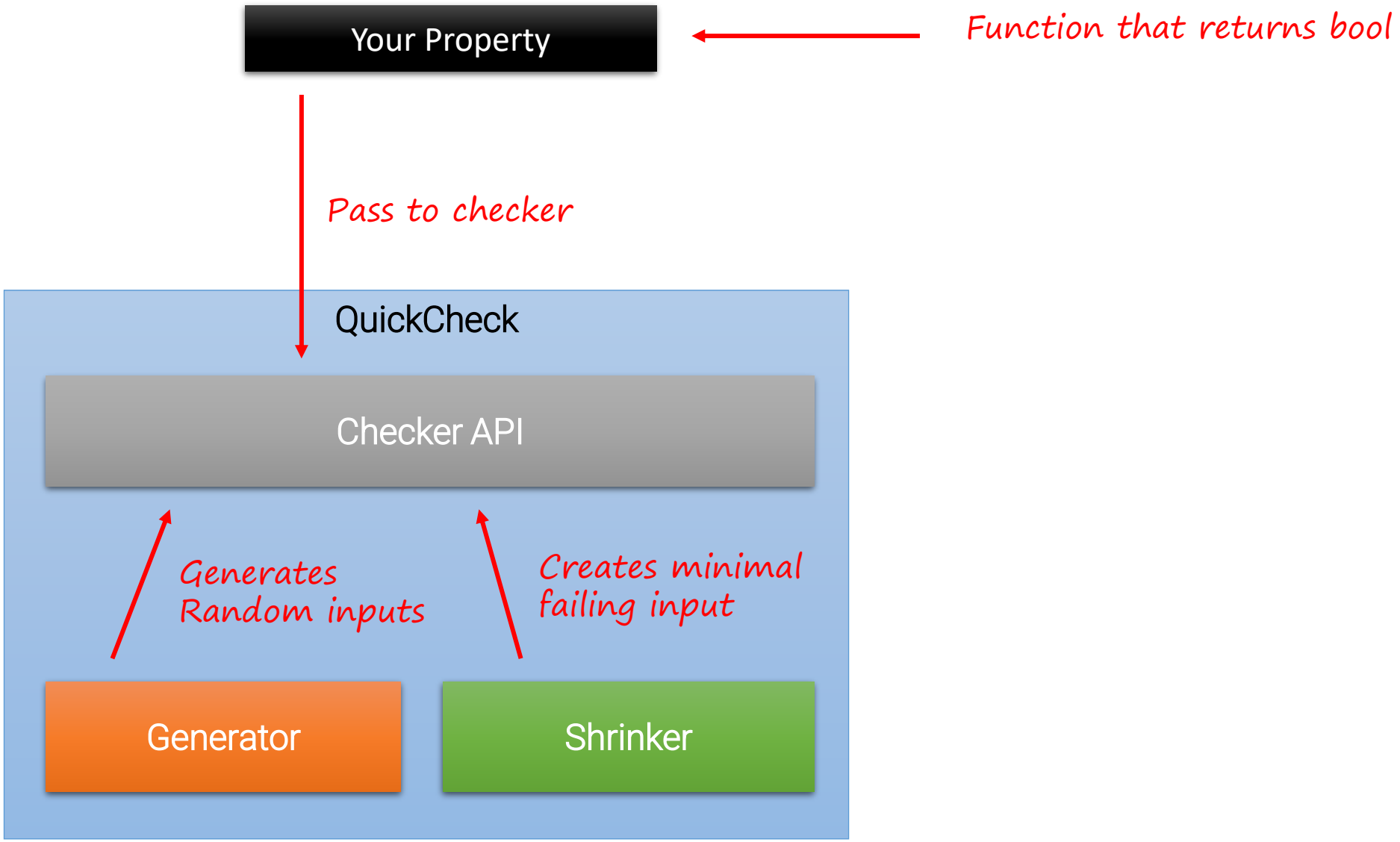
5 + 0 = 5 with bags of dots

Introducing QuickCheck & beyond

QuickCheck



https://en.wikipedia.org/wiki/QuickCheck


Your Property
Function that returns bool
Pass to checker
QuickCheck
Checker API
Generates
Random inputs
Creates minimal
failing input
Generator
Shrinker

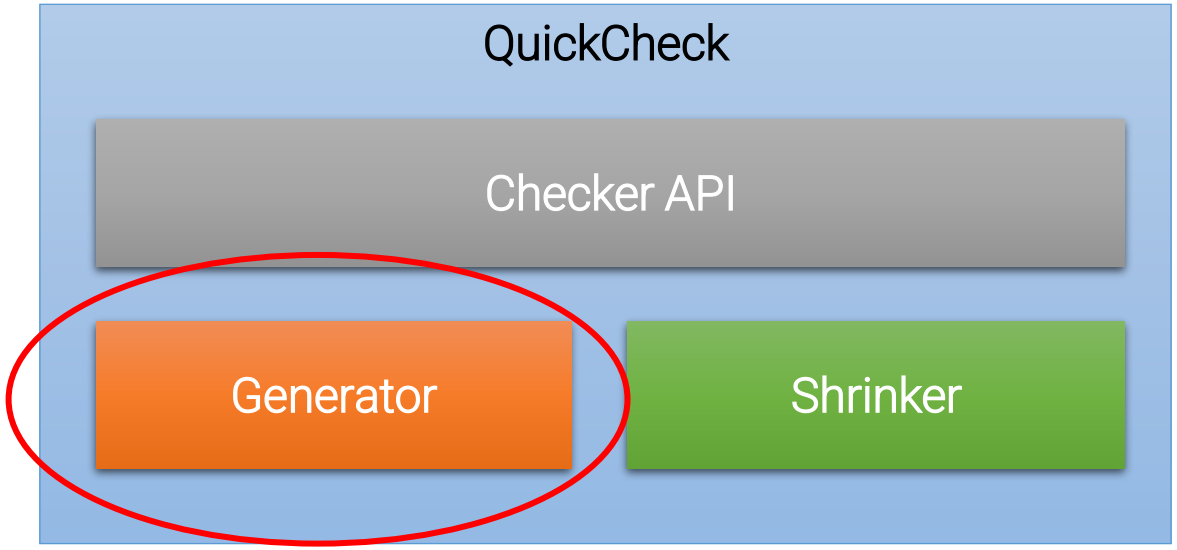
FSCheck

Don't forget to mention CSCheck!

Time to integrate FSCheck in our tests…

Generators

Generators: Making random input



Generate types

“int” generator	0, 1, 3, -2, …
“string” generator	“”, “eiX$a[”, “U%0Ika&r”, …
“bool” generator	true, false, false, true, …
“Tuple<int,int>” generator	(0,0), (1,0), (2,0), (-1,1), (-1,2), …
“Color” generator	Red, Green, Blue, Green, …

Generates values of a custom type

How it works in practice

```
[Property(Verbose =true)]
| 0 references | Wullems Bart, 1 day ago | 1 author, 1 change
public Property Adding_Two_Numbers_Doesnt_Depend_On_Parameter_Order_2(Tuple<int,int> values)
{
    return (Add(values.Item1, values.Item2) == Add(values.Item1, values.Item2)).ToProperty();
}
```

(1) Checker detects that the input is a tuple of (int,int)

Checker API

(2) Appropriate generator is created

Tuple<int,int> generator

(3) Valid values are generated

(0,0) (1,0) (2,0) (-1, 1) (100, -99) …

(4) Values are passed to the property

Let’s look at some generators…

Debugging a disproven property

Debugging a disproven property

```
PropertyBasedTesting.Tests.BuiltInGeneratorsTests.Generator2
  Source: BuiltInGeneratorsTests.cs line 41
  Duration: 202 ms

Message:
  FsCheck.Xunit.PropertyFailedException :
  Falsifiable, after 1 test (1 shrink) (StdGen (2052518649,297245404)):
  Original:
  NonEmptyArray [|null; ""|]
  Shrunk:
  NonEmptyArray [|""|]
```

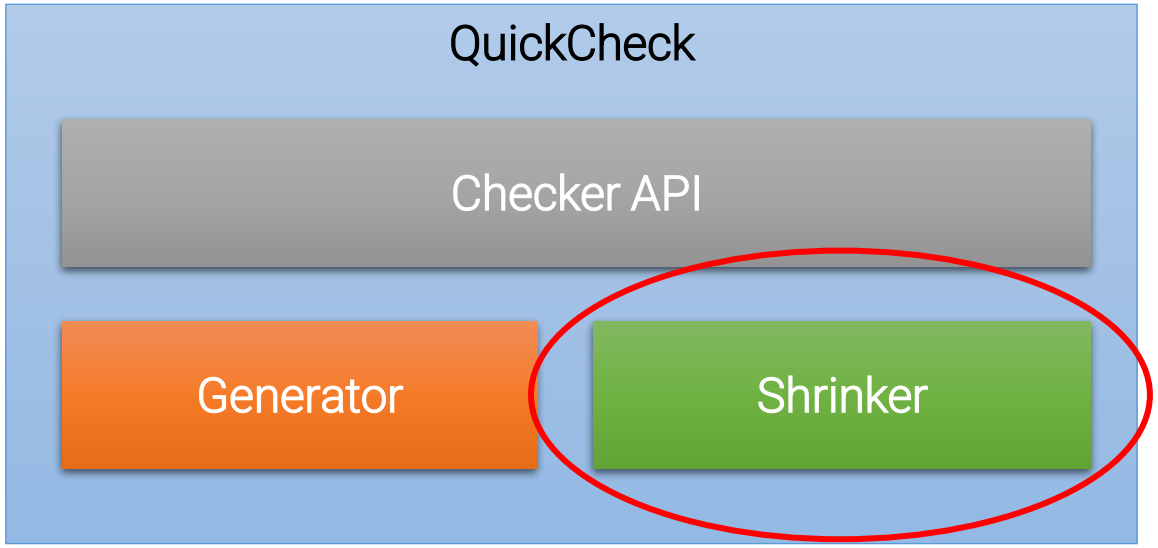
Locking input

Shrinking

Locking our input…

Shrinking

Shrinking



How shrinking works

Property to test – we know it's gonna fail!

```
public Property IntTester(int x) => (x < 81).ToProperty();
```

"int" generator	0, 1, 3, -2, 34, -65, 100

Fails at 100!

How shrinking works

```
public Property IntTester(int x) => (x < 81).ToProperty();
```

Shrink list for 100

0, 50, 75, 88, 94, 97, 99

Fails at 88!

Generate a new sequence up to 100

How shrinking works

```
public Property IntTester(int x) => (x < 81).ToProperty();
```

Shrink list for 88

0, 44, 66, 77, 83, 86, 87

Fails at 83!

Generate a new sequence up to 88

How shrinking works

```
public Property IntTester(int x) => (x < 81).ToProperty();
```

Shrink list for 83

0, 42, 63, 73, 78, 81, 82

Fails at 81!

Generate a new sequence up to 83

How shrinking works

```
public Property IntTester(int x) => (x < 81).ToProperty();
```

Shrink list for 81

0, 41, 61, 71, 76, 79, 80

All pass!

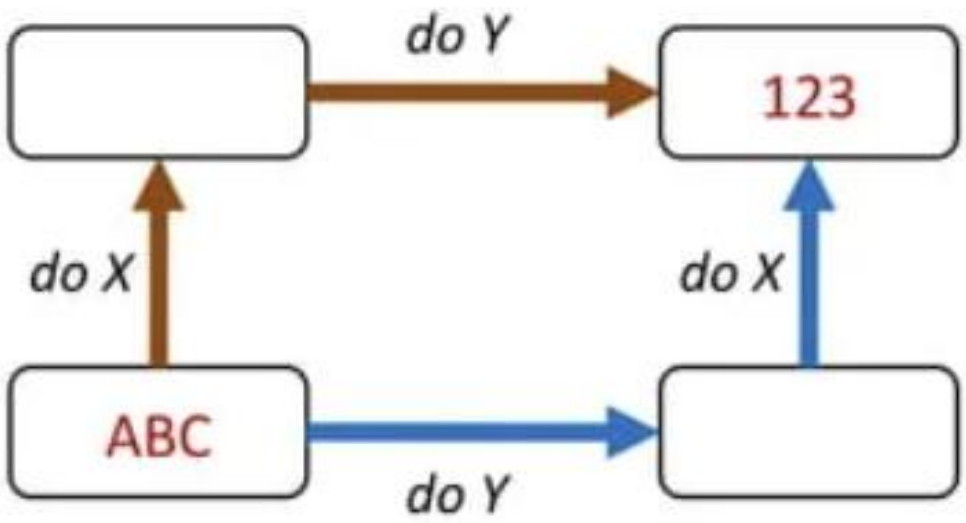
Generate a new sequence up to 81

Shrink has determined that 81 is the smallest failing input

Shrinking time…

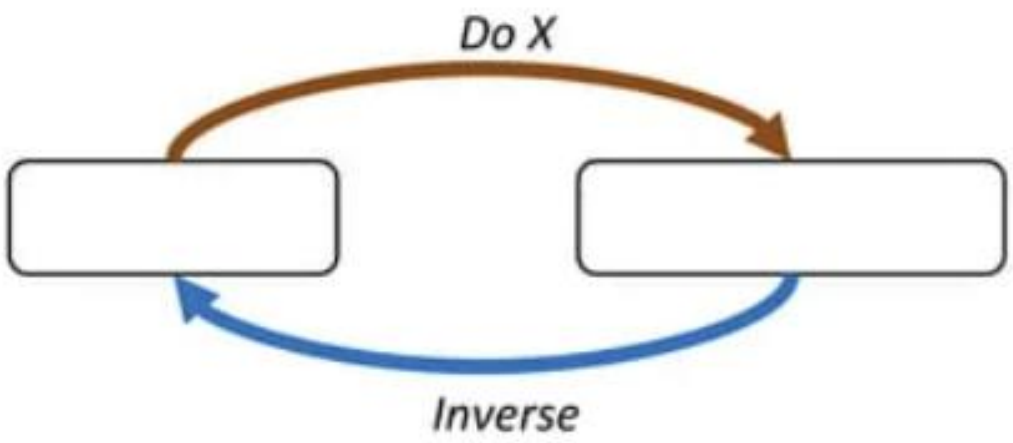
Identifying properties

Different paths, same destination



- Examples
  - Commutivity
  - Associativity
  - Map

There and back again



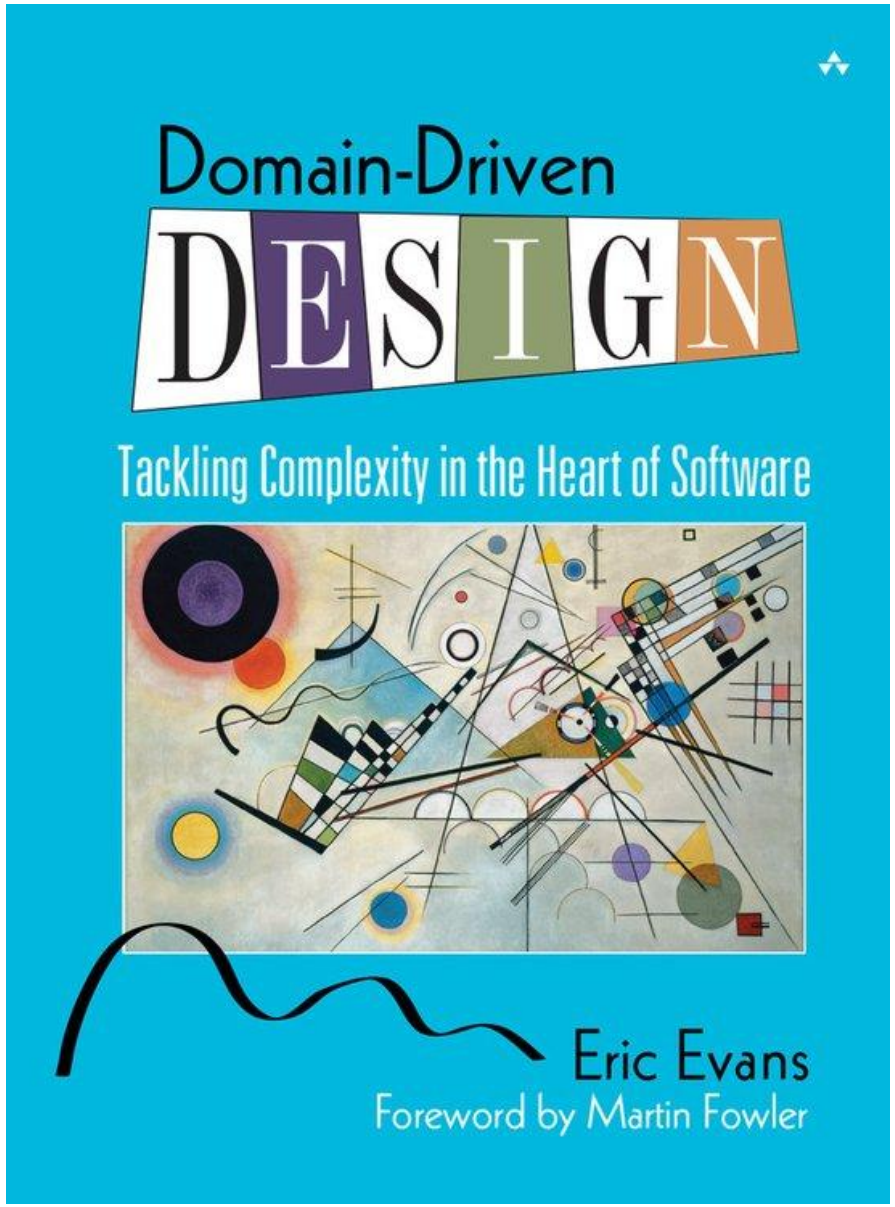
- Examples
  - Serialization/Deserialization
  - Encrypt/Decrypt
  - Addition/Subtraction
  - Write/Read
  - Set/Get

Some things never change

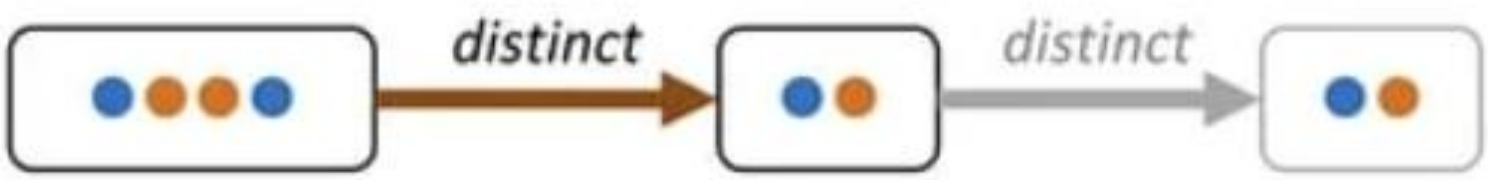
- Examples
  - Size of a collection
  - Contents of a collection
  - Balanced trees
  - Conservation laws (total money in accounting system)

Business Invariants

A business invariant is a rule or constraint in a business domain that must always hold true.

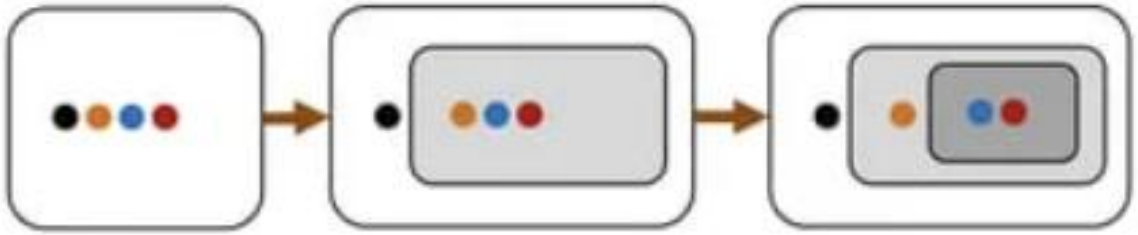


The more things change, the more they stay the same



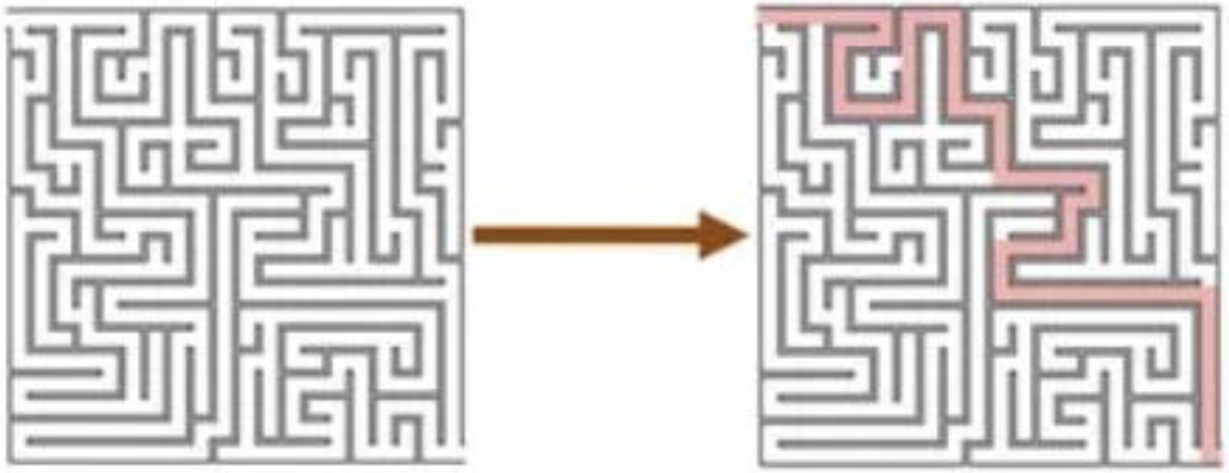
- Idempotence
  - Sort
  - Filter
  - Event processing

Solve a smaller problem first



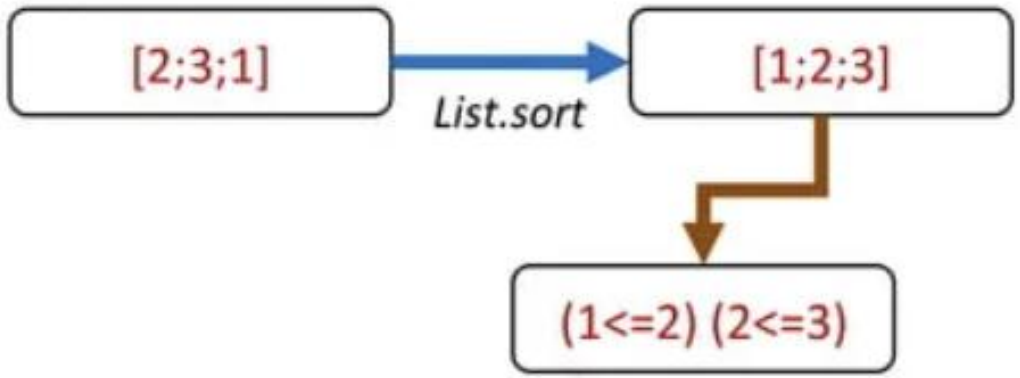
- Divide and conquer algorithms(e.g. quicksort)
- Structural induction (recursive data structures)

Hard to prove, easy to verify



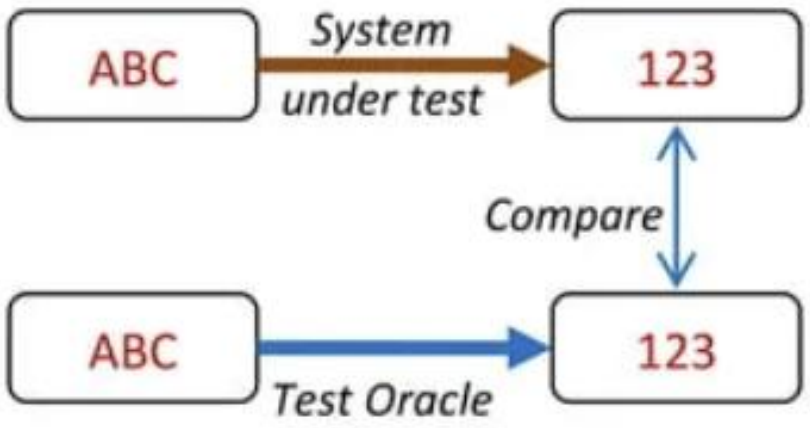
- Algorithms
- Postconditions

Hard to prove, easy to verify



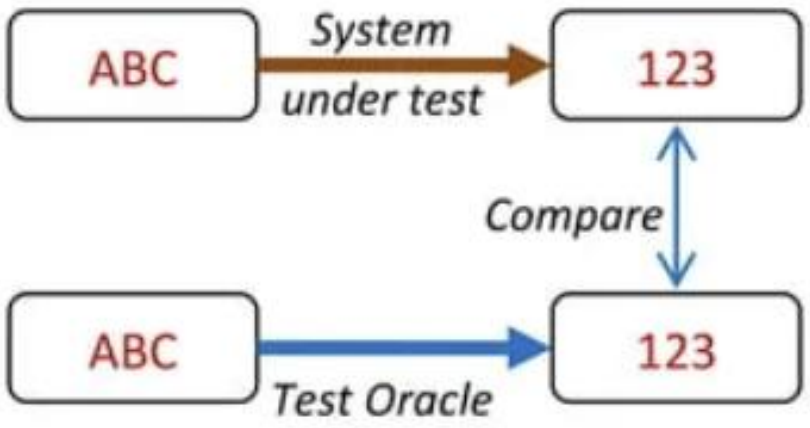
To verify the sort,
Check that each pair is ordered

The test oracle



- Compare optimized with slow brute-force version
- Compare parallel with single thread version

Compare different implementations



- Test your optimized function against a simple, obviously correct version
- fast_sort(list) == naive_sort(list) for all inputs
- Compare new algorithm with established reference implementation

An example…

The diamond kata

Problem Description

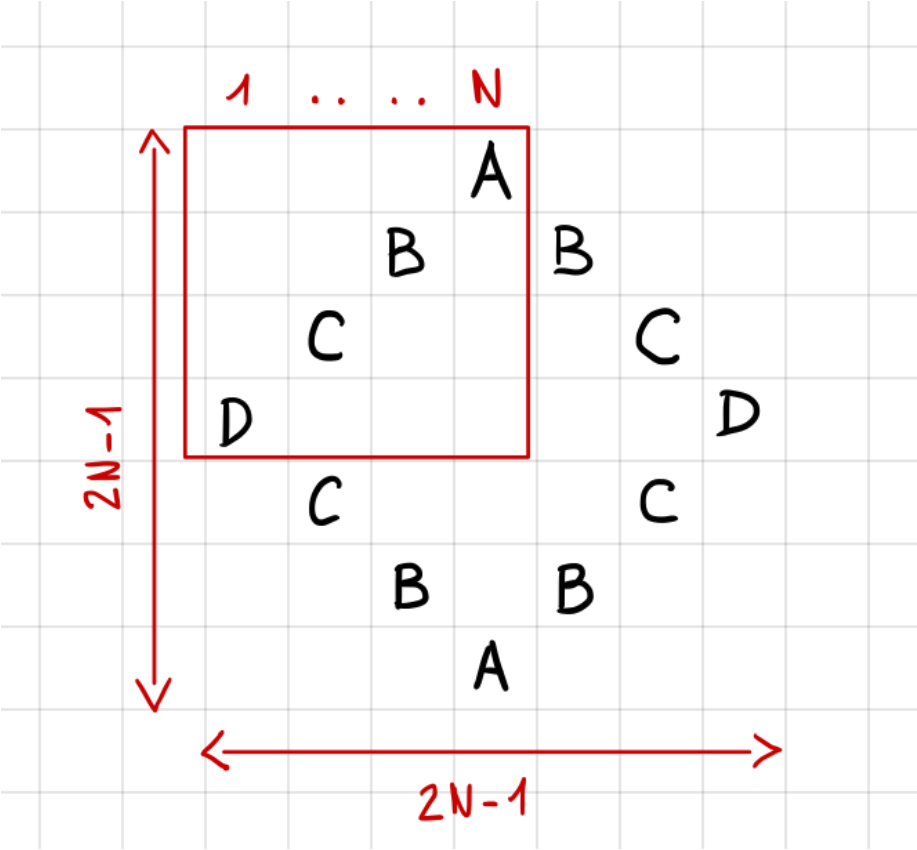
Given a letter, print a diamond starting with 'A' with the supplied letter at the widest point.

For example: print-diamond 'C' prints

```
  A
 B B
C   C
 B B
  A
```

http://codingdojo.org/kata/Diamond/

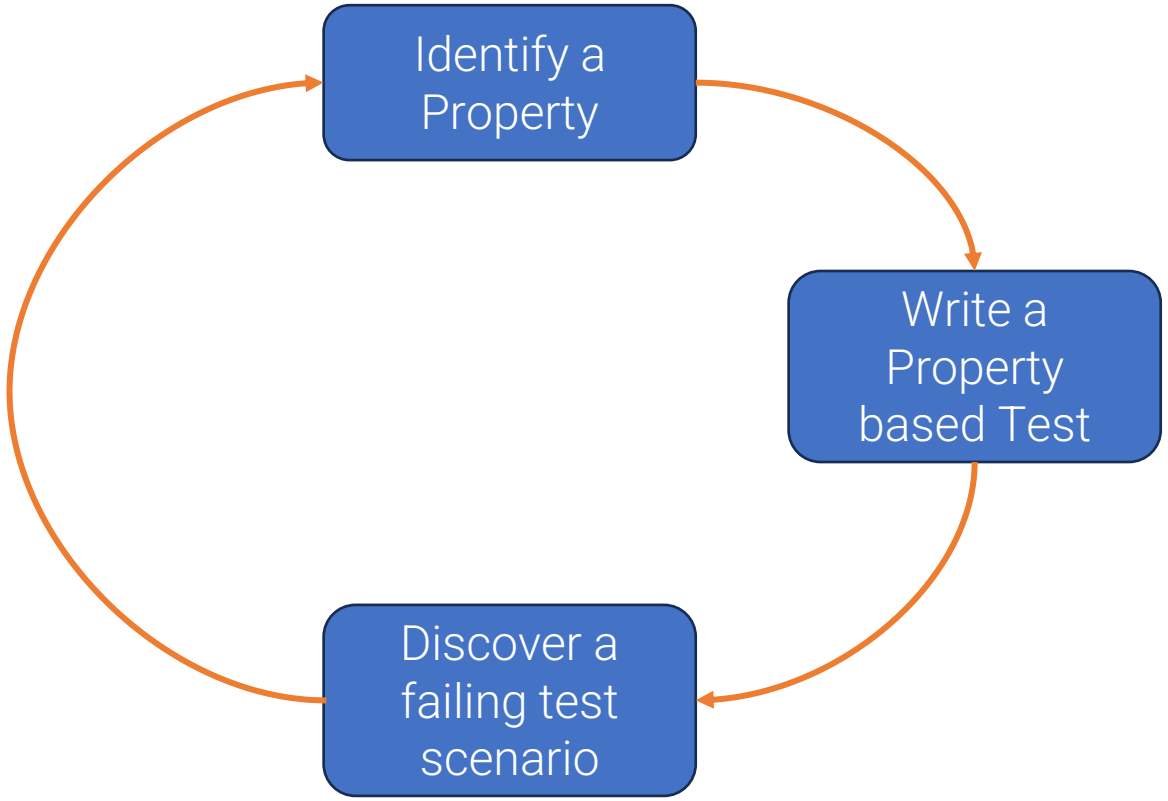
The diamond kata



- Given *N* the position of the (argument) letter in the alphabet:
- The upper left corner of the pattern is filled with a diagonal formed by the letters A,B,C etc. The letter A is in position N (starting from 1), B in position N-1, C, in position N-2, and so on.
- The pattern has horizontal symmetry, which means that flipping it horizontally yields the initial pattern.
- The pattern also has vertical symmetry.
- The height of the diamond is 2N-1.
- The maximum width of the diamond equals its height, also 2N-1.

http://codingdojo.org/kata/Diamond/

The Property based Testing cycle



A real life example…

The maître d' kata

The objective of the exercise is to implement the `MaîtreD` decision logic.

Reservations are accepted on a first-come, first-served basis. As long as the restaurant has available seats for the desired reservation, it'll accept it.

A reservation contains, at a minimum, a date and time as well as a positive quantity. Here's some examples:

Date	Quantity
August 8, 2050 at 19:30	3
November 27, 2022 at 18:45	4
February 27, 2014 at 13:22	12

Notice that dates can be in your future or past. You might want to assume that the maître d' would reject reservations in the past, but you can't assume *when* the code runs (or ran), so don't worry about that. Notice also that quantities are positive integers. While a quantity shouldn't be negative or zero, it could conceivably be large. I find it realistic, however, to keep quantities at low two-digit numbers or less.

https://blog.ploeh.dk/2020/01/27/the-maitre-d-kata/

The maître d’ kata

- Given *a random date n* and *max capacity x:*
- The restaurant should accept reservations as long as they don’t exceed the maximum *capacity x* for *date n*
- The restaurant should decline reservations when they exceed the maximum *capacity x* for *date n*

https://blog.ploeh.dk/2020/01/27/the-maitre-d-kata/

Some real life properties – Inventory and Supply Chain

- Stock levels:
  - After sale, `new_inventory_count == old_count - sold_quantity`
- Reservation system:
  - `available_inventory + reserved_inventory == total_inventory`
- Batch tracking:
  - Items from same batch should have identical expiration dates and lot numbers
- Reorder points:
  - `should_reorder(current_stock, reorder_level)` triggers when stock falls below threshold

Some real life properties – Financial Systems

- Account balances:
  - After any transaction, `sum(all_account_balances) == initial_total` (conservation of money)
- Interest calculations:
  - `compound_interest(principal, 0, periods) == principal` (zero rate means no change)
- Payment processing:
  - `process_refund(process_payment(amount))` should restore original balance
- Double-entry bookkeeping:
  - For every transaction, `sum(debits) == sum(credits)`

Some real life properties – Scheduling and Booking

- Meeting conflicts:
  - No two meetings for same person should overlap in time
- Resource booking:
  - `cancel_booking(make_booking(resource, time))` should free up that time slot
- Capacity constraints:
  - `booked_seats <= total_capacity` for any event
- Time zone handling:
  - Converting meeting time to participant's local time and back should preserve original time

Conclusion

Conclusion

- PBT’s are more general
  - One property-based tests can replace many example-based tests
- PBT’s can reveal overlooked edge cases
  - Nulls, negative numbers, weird strings, etc…
- PBT’s require deep understanding of requirements
  - Property-based tests force you to think ☺
- Example-based tests are still helpful though!
  - Easier to understand for newcomers

Go write some tests!

FULL STACK CONFERENCE

2025 EDITION