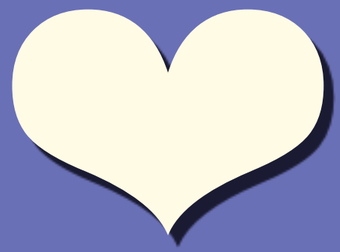
Property Based Testing



Example Scenario:

Multi Source Inventory

(c) Vinai Kopp - http://vinaikopp.com - twitter://@VinaiKopp - #MageTesFest 2019

I WANT IT!
IS IT READY?

Let's test!

What do we test?

Scenario

1 website
1 source
1 stock
1 product qty 1
1 order qty 1

```
assertTrue($isItemSoldOut)
```

It works!

...BUT DOES IT?

Scenario 2

2 website
2 sources
2 stocks
1 product, qty 1 in each source
1 order from source 1, qty 1

```
assertTrue($isItemSoldOutForStock1)
```

How confident are you?

Okay, a few more cases...

Each of the prior tests but with:

2 products

Oh, and also with:

2 orders

What about backorders?

But what about qty thresholds?

But what about positive & negative qty thresholds?

And decimal quantities?

And shouldn't we test that if a product is not sold out it still is salable?

Are there new edge cases that are introduced by interacting features?

...It's impossible!

Let's just forget about automated testing.

KTHXBYE

QUESTIONS?

Real hero: John Hughes

Don't write tests,
generate them!

-- John Hughes

Instead of examples...

We describe the full range of valid inputs

1+ websites
1+ sources
1+ products assigned to sources at random
Each with a qty in of -max_int to max_int
Each with or without backorders enabled
Each with a threshold of -max_int to max_int
Each with or without decimal quantities
1+ orders per product with a total qty from .0001 to the salable amount

Let the test framework

GENERATE

a random sample for each input

And we do that hundreds of times.

And we do that ~~hundreds~~ N times.

Unrealistic!

It takes too much time.

1-5 websites
1-5 sources
1-100 products assigned to sources at random
Each with a qty of -100 to 100
Each with or without backorders enabled
Each with a threshold of -10 to 10
Each with or without decimal quantities
n orders per product with a total qty from .0001 to the salable amount

S.S.A.F.

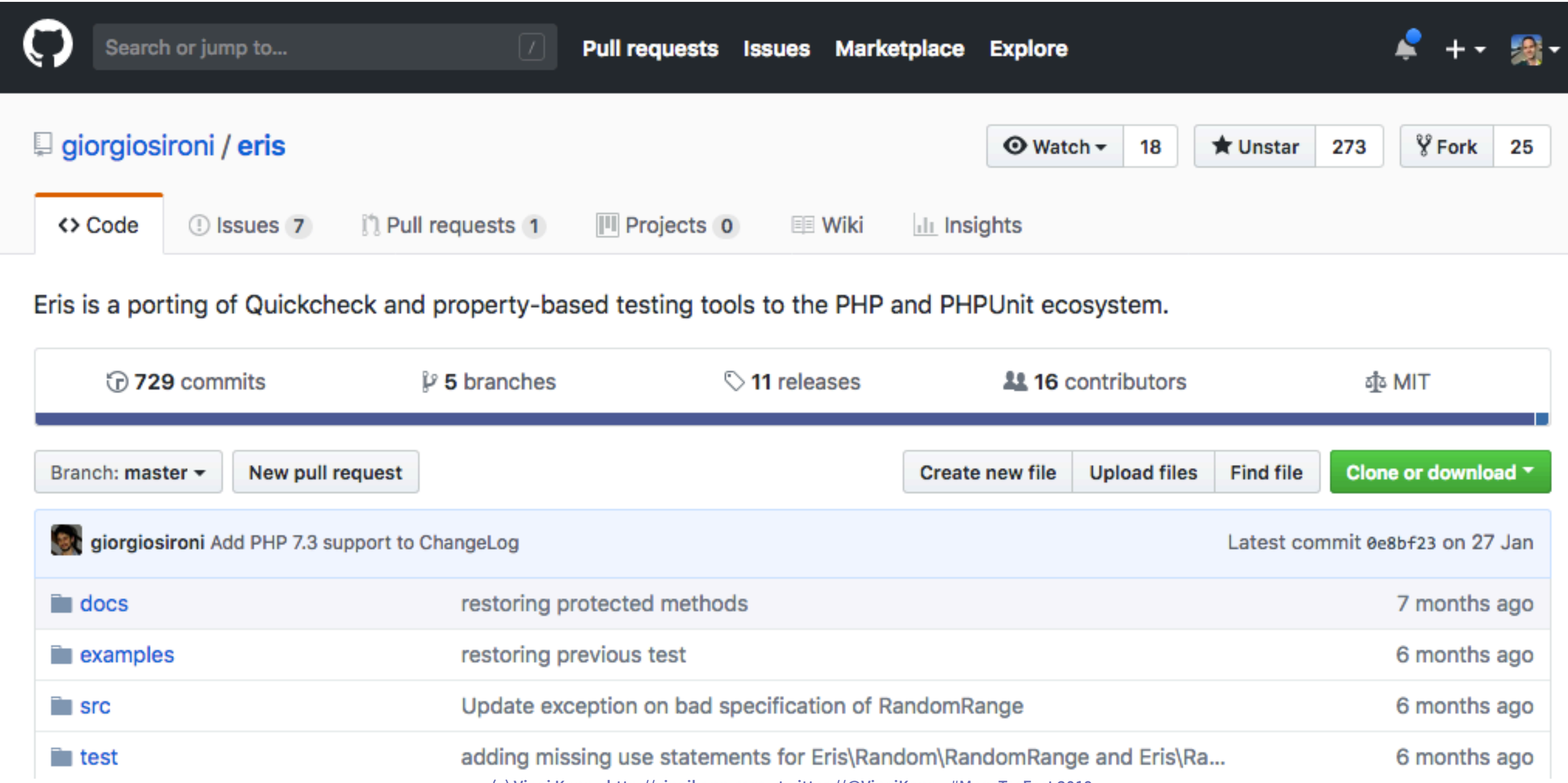
(STILL SLOW AS HELL)

2 fix websites
2 fix sources
1 product
Stock qty of -100 - 100
With a threshold of -10 to 10
With backorders enabled if threshold < 0
With decimal quantities
1 - 10 orders per product

That might work

How exactly?

Details please



(c) Vinai Kopp - http://vinaikopp.com - twitter://@VinaiKopp - #MageTesFest 2019

```
$check = $this->forAll(...$generators);
```

And `...$generators` is what?

```
$simpleProducts = Generator\simpleProduct();
$inStockQtys = Generator\chooseFloat(-100, 100);
$minQtyThresholds = Generator\chooseFloat(-100, 100);
$nOrdersForProduct = Generator\choose(1, 10);

$check = $this->forAll(
    $simpleProducts,
    $inStockQtys,
    $minQtyThresholds,
    $nOrdersForProduct
);
```

Generators describe the range of valid inputs

Generators

are

Composeable

```
Generator\int();    // => 29384
Generator\neg();    // => -6
Generator\nat();    // => 0
Generator\pos();    // => 15
Generator\float(); // => 21.92836
```

```
Generator\string(); // => Jx@
Generator\names();  // Alice
```

```
Generator\choose(1, 1000) // => 55
```

```
Generator\vec(4, Generator\names());
// => ["Zénobin", "Dee", "Allen", "Theophania" ],
```

```php
public static function skuGenerator(): Generator
{
    return Generator\suchThat(

        function (string $s) {
            return $s !== '' && strlen($s) <= 64;
        },

        Generator\string()
    );
}
```

```php
public static function simpleProductGenerator(): Generator
{
    return Generator\bind(

        Generator\tuple(
            self::skuGenerator(),
            self::productNameGenerator()
        ),

        function (array $skuAndName) {
            [$sku, $name] = $skuAndName;

            $product = self::createSimpleProduct($sku, $name);
            return Generator\constant($product);
        }
    );
}
```

```php
public static function simpleProductsGenerator(): Generator
{
    return Generator\seq(
        self::simpleProductGenerator()
    );
    // => array of zero or more products
}
```

Generating interdependent values is a bit more complex...

```php
Generator\bind(
    Generator\suchThat(
        function (array $tuple) {
            [$inStockQty, $minInStockQty] = $tuple;
            return $minInStockQty <= 0 || $minInStockQty > $inStockQty;
        },
        Generator\tuple(
            $this->chooseFloat(0, 100), // inStockQty
            $this->chooseFloat(-100, 100) // minInStockQty
        )
    ),
    function (array $tuple): Generator {
        [$inStockQty, $minInStockQty] = $tuple;
        $salableQty = $inStockQty - $minInStockQty;
        return Generator\associative([
            'inStockQty' => Generator\constant($inStockQty),
            'minQtyThreshold' => Generator\constant($minInStockQty),
            'orderQty' => Generator\frequency(
                [1, $salableQty], // place orders for full purchasable quantity
                [2, $this->chooseFloat(0, $salableQty - 0.0001)] // purchase below salable quantity
            )]
        );
    }
);
```

Generators describe the full scope of valid inputs

```
$check = $this->forAll(...$generators);
```

AND THEN?

Declare System Properties

Property:

Something that is always true

```php
$check = $this->forAll(Generator\int(), Generator\int());

$check(function (int $a, int $b) {
    $this->assertSame(
        $a + $b,
        $b + $a
    );
});
```

How about something Magento related?

Quote Merging:

```
(customer quote item qty) + (guest quote item qty)
== merged quote item qty
```

```
(customer quote item count) + (guest quote item count)
>= merged quote item count
```

```
$check = $this->forAll(
    $this->simpleProducts(),
    $this->simpleProducts()
);
```

```
$check(function (array $inCustomerQuote, array $inGuestQuote) {

    $customerQuote = $this->createQuote();
    $guestQuote = $this->createQuote();

    $this->addProductsToQuote($customerQuote, $inCustomerQuote);
    $this->addProductsToQuote($guestQuote, $inGuestQuote);

    $customerQuoteItemQtyBefore = $customerQuote->getItemsQty();
    $guestQuoteItemQtyBefore = $guestQuote->getItemsQty();
```

```
$customerQuote->merge($guestQuote);
```

```
    $this->assertSame(
        $customerQuoteItemQtyBefore + $guestQuoteItemQtyBefore
        $customerQuote->getItemsQty()
    );

    $this->assertLessThanOrEqual(
         count($inCustomerQuote) + count($inGuestQuote),
         count($customerQuote->getAllVisibleItems()
    );
});
```

Finding Properties

Reversible algorithms

```
$this->assertSame(
    $input,
    json_decode(json_encode($input), true)
);
```

Algorithms with algebraic properties

```
$this->assertEquals(
    $quoteC->merge($quoteB->merge($quoteA));
    $quoteA->merge($quoteB->merge($quoteC));
);

// $quoteC . ($quoteB . $quoteA)
// ($quoteC . $quoteB) . $quoteA
```

Generative smoke tests

```
$check = $this->forAll(Generator\string());

$check(function(string $requestPayload) use ($api) {
    $response = $api->post($requestPayload);

    $this->assertContains(
        $response->status(),
        [200, 201, 202, 203, 204, 205, 206]
    );

    $this->assertNotNull(
        json_decode($response->body(), true)
    );
});
```

Alternative implementation

```
$this->assertSame(
    $system->process($a, $b),

    $otherImplementation->process($a, $b)
);
```

Testing stateful systems

```
$check = $this->forAll(
    Generator\seq(Generator\domainAction())
);

$check(function(array $actions)) use ($system, $model) {

    foreach ($actions as $takeAction) {
        $takeAction($system);
        $takeAction($model);
    }

    $this->assertSame($system->getState(), $model->getState());
});
```

In a nutshell:

1. Define valid inputs with generators
2. Write tests for properties

How fast is it?

As fast as the system under test

How hard is it?

It takes effort

Is it worth the time?

Does it find bugs?

It does indeed.

While preparing this talk...

Decimal qtys
Backorders enabled
Negative min qty threshold

Often impossible to purchase the complete salable quantity when multiple decimal orders are used.

`Api\GetProductSalableQtyInterface` reports that only `-8.3266726846887E-17` are in stock.

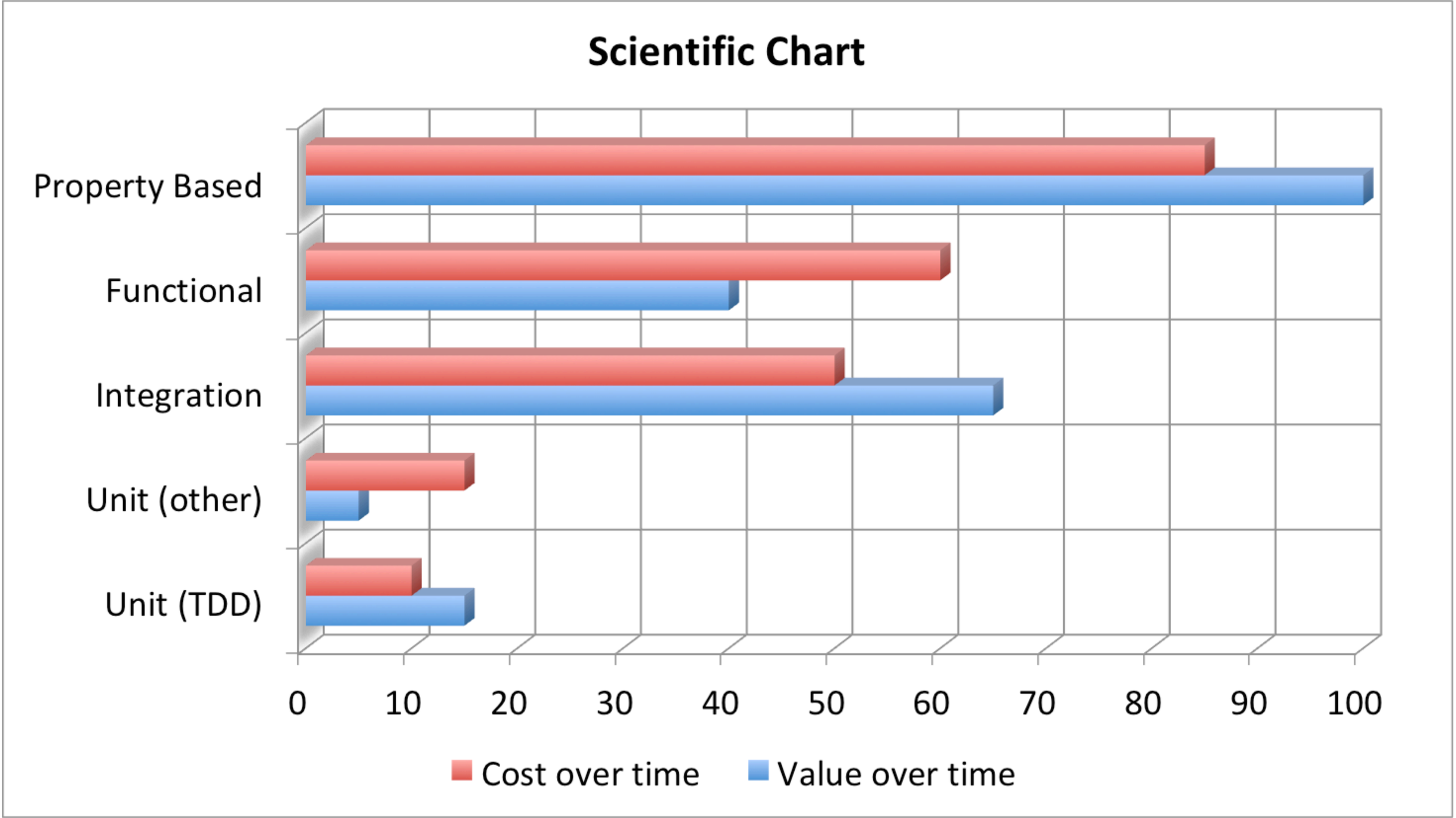
Zero product stock qty
Backorders enabled
Negative min qty threshold

Wrong salable quantity

`Api\GetProductSalableQtyInterface` reports salable quantity as zero instead of the min qty threshold

Property based testing is better than humans at finding edge cases

So... is it WORTH it?


Scientific Chart
Property Based
Functional
Integration
Unit (other)
Unit (TDD)
0
10
20
30
40
50
60
70
80
90
100
Cost over time
Value over time

Thanks to

John Hughes \ QuickCheck

Sebastian Bergman \ PHPUnit

Giorgio Sironi \ Eris

Please come and chat!

IRL RIGHT HERE

twitter://@VinaiKopp

slack://vinai@magentocommeng.slack.com

slack self signup: http://tinyurl.com/engcom-slack