Unit Testing PHP Applications

Michael Moussa | @michaelmoussa

https://github.com/michaelmoussa/ssp2015-unit-testing

https://joind.in/13437

About me

- PHP developer for 15 years
- Lead Developer, ZAM Network
    - *"We make gaming better!"*
    - lolking.net, wowhead.com, destinydb.com
- Zend Certified PHP Engineer & ZF2 Certified Architect

About this talk

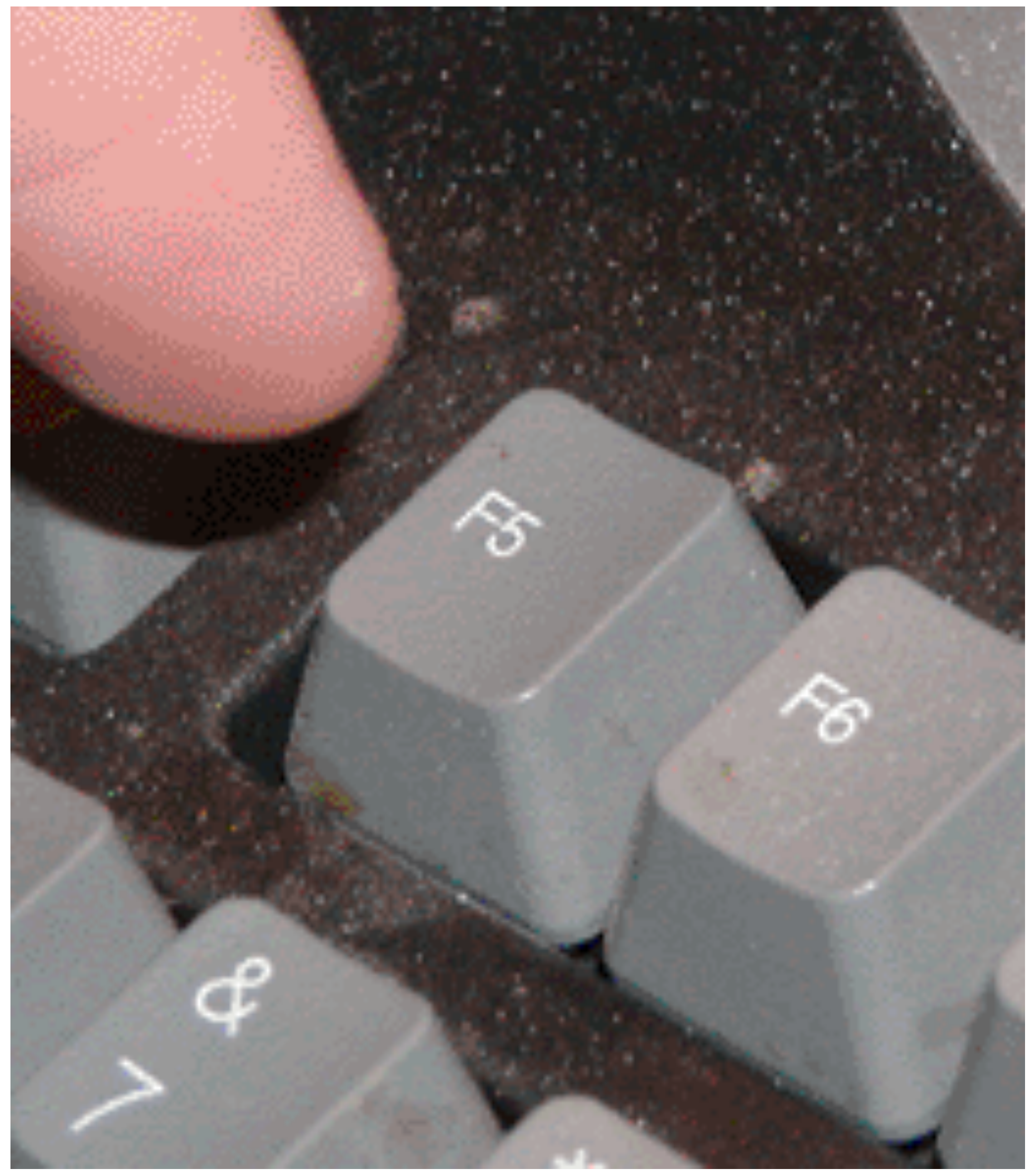
- Overview of unit testing
- Getting started
- Basic test examples
- Testing in isolation
- Testability
- Metrics
- Tips and tricks

Overview of Unit Testing

- An **automated** means of verifying **small portions** of an application against **a specification**.
- Automated - you don't have to go through a lot of effort to run the tests once they're written. Execute a command, and the testing framework does the rest.
- Small portions - individual methods of a class and how they interact with their collaborators.
- A specification - your definition of how things are supposed to work.

"Does this 'unit' of code do what you expect it to do?"

So remember...



NOT THIS!

Getting started

- PHPUnit: the de facto standard library for PHP testing

  https://phpunit.de/

  https://github.com/sebastianbergmann/phpunit

```
composer require phpunit/phpunit --dev
```

Example Time!

Testing in isolation

```php
class WeatherService
{
    public function getTemperature($city)
    {
        $cacheKey = md5($city);
        $weatherData = $this->cache->get($cacheKey);

        if (!$weatherData) {
            $weatherData = $this->httpClient->get(
                'https://some-weather-api.com/temperature/' . urlencode($city)
            )->json();
            $this->cache->set($cacheKey, $weatherData, self::TEMPERATURE_CACHE_TTL);
        }

        return $weatherData;
    }
}
```

- Cache
- External API
- Other possibilities?
    - DB
    - Filesystem

These things are slow and potentially fragile!

So what do we do?

- Don't test the cache server, database server, REST API, etc!
- Test only that *our code* is using them correctly.
- We can do this with **Mock Objects.**

Mock Objects

Objects pre-programmed to behave in certain ways

- Create an object that your class can use for testing purposes
- Tell it which methods it should expect to have called
- Tell it what the parameters should be
- Tell it what value(s) it should return
- Use it in your test instead of a "real" object

Example Time!

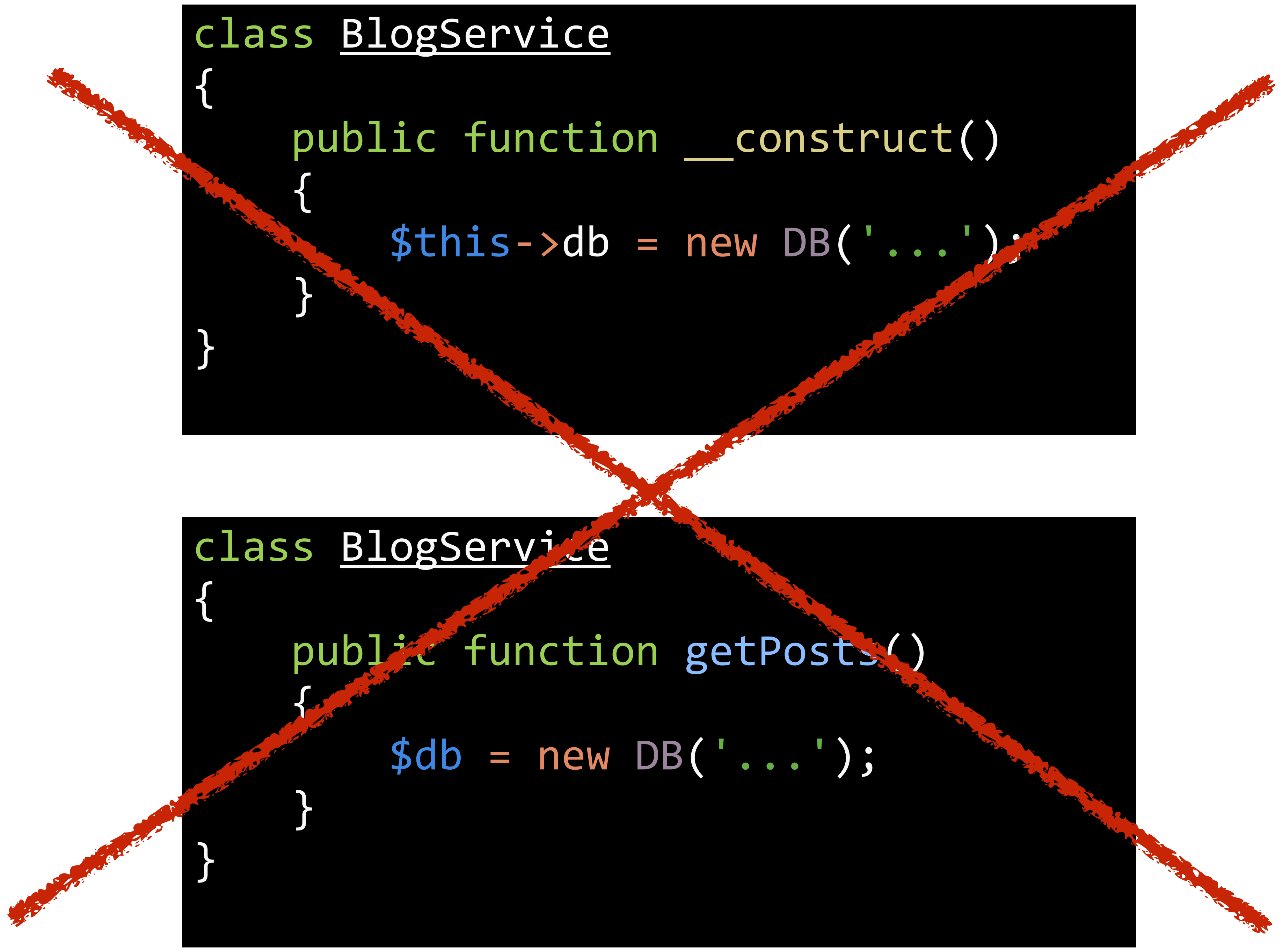
Testability

What makes code difficult to test?

- Objects creating their own dependencies
- Globals
- Static methods
- Singletons
- Classes that do too much

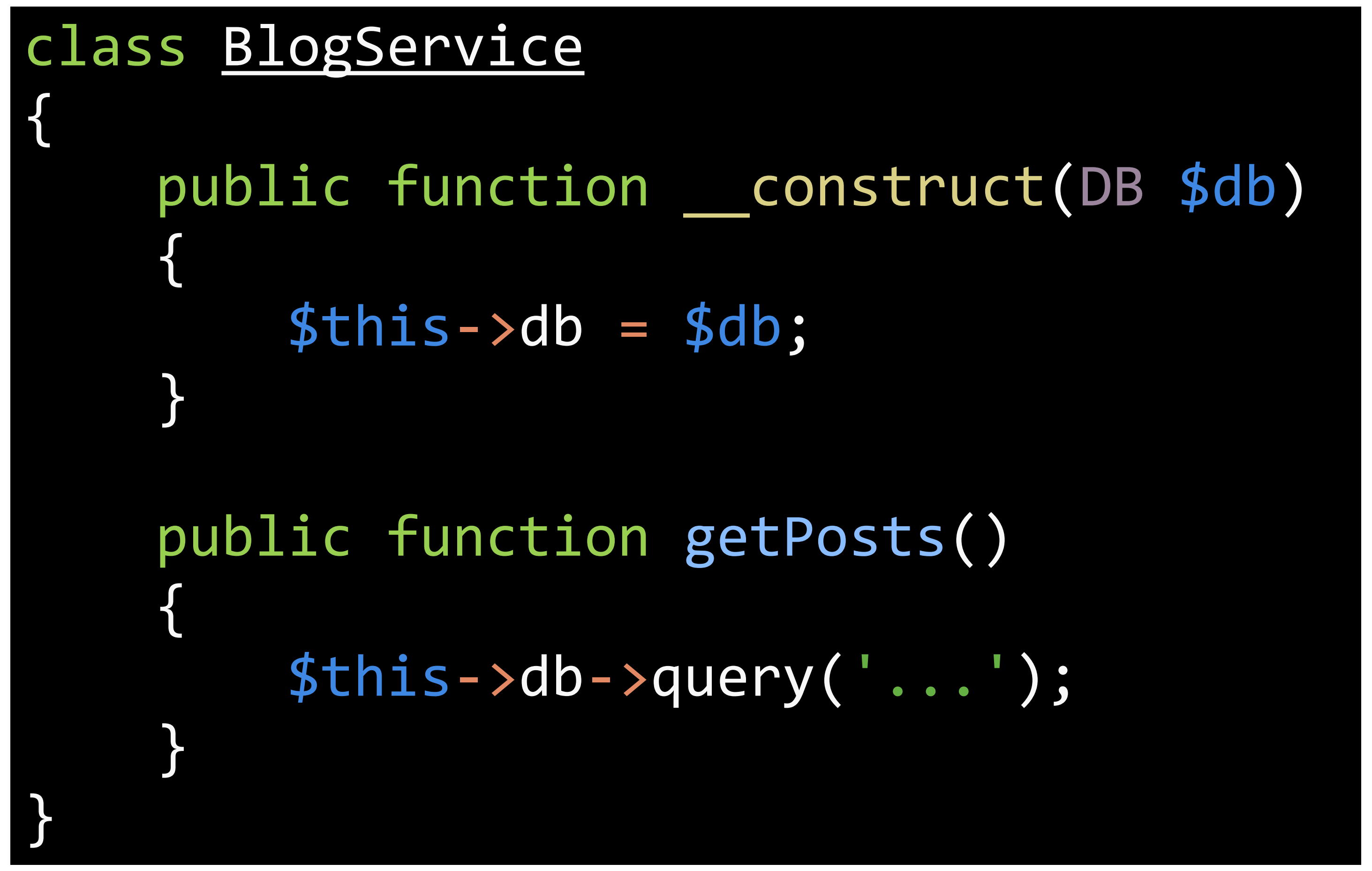
Testability

Objects creating their own dependencies



```php
class BlogService
{
    public function __construct()
    {
        $this->db = new DB('...');
    }
}
```

```php
class BlogService
{
    public function getPosts()
    {
        $db = new DB('...');
    }
}
```



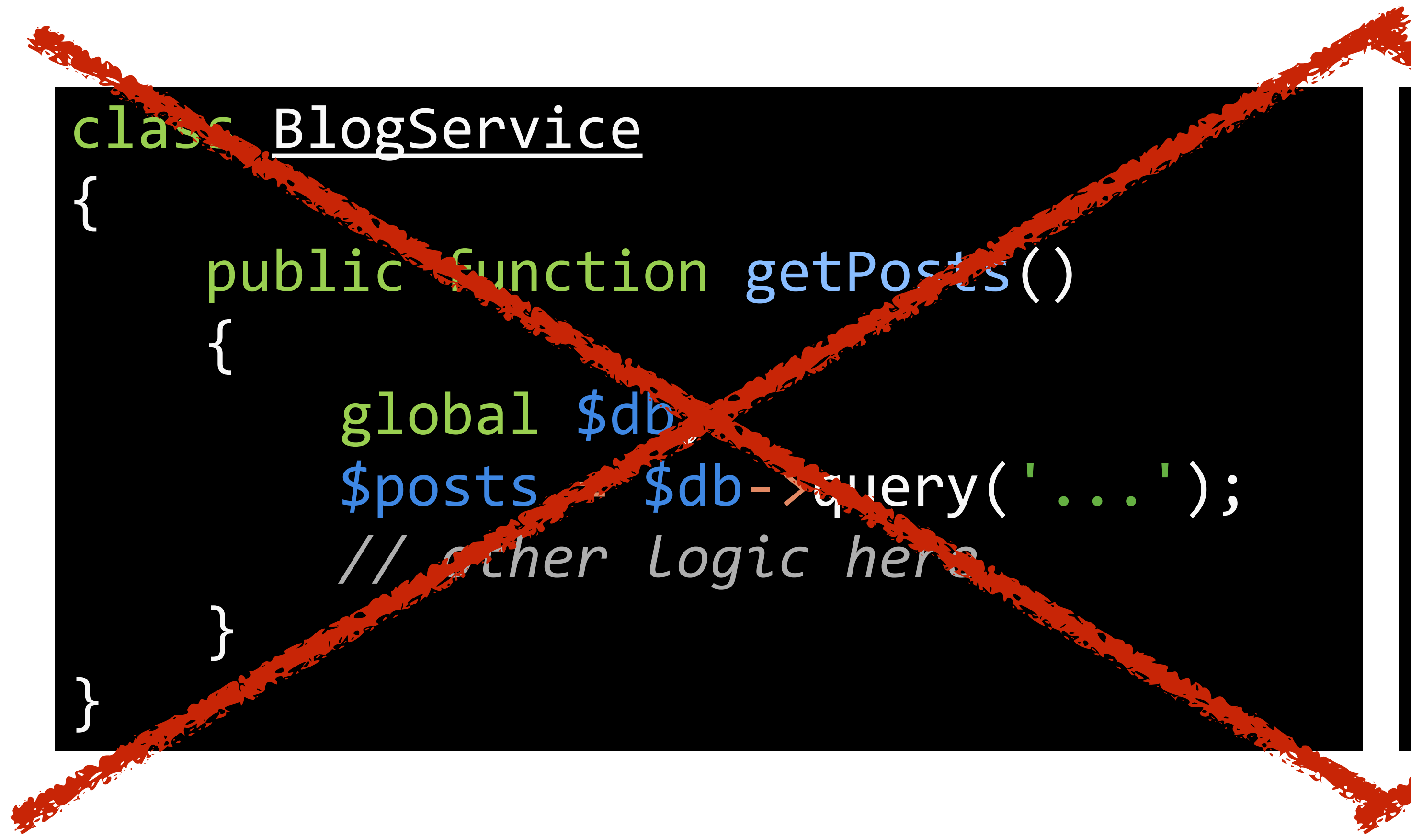
```php
class BlogService
{
    public function __construct(DB $db)
    {
        $this->db = $db;
    }

    public function getPosts()
    {
        $this->db->query('...');
    }
}
```

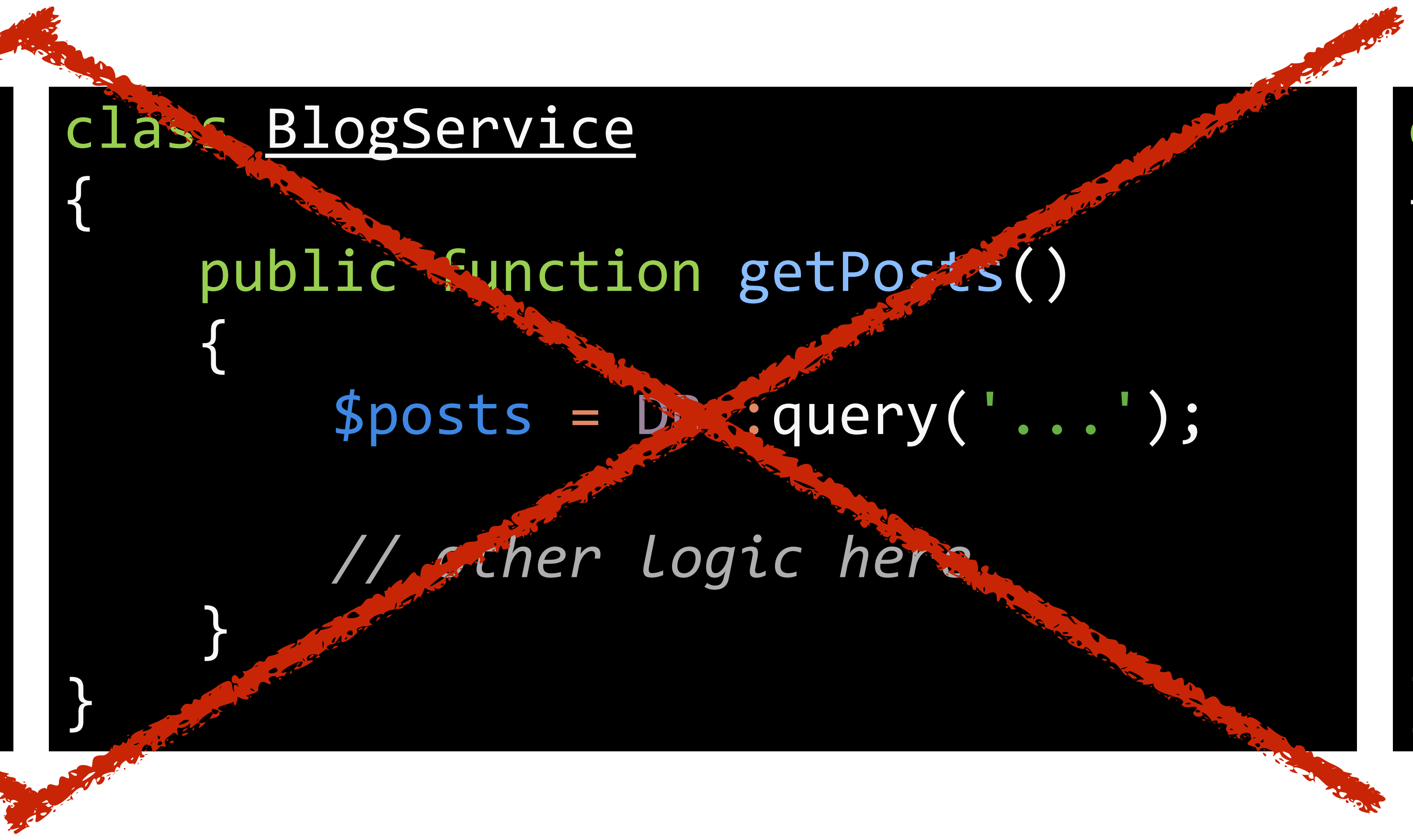
Instead, use **Dependency Injection** (aka "pass things in as parameters")

Testability

Globals, Static Methods, Singletons

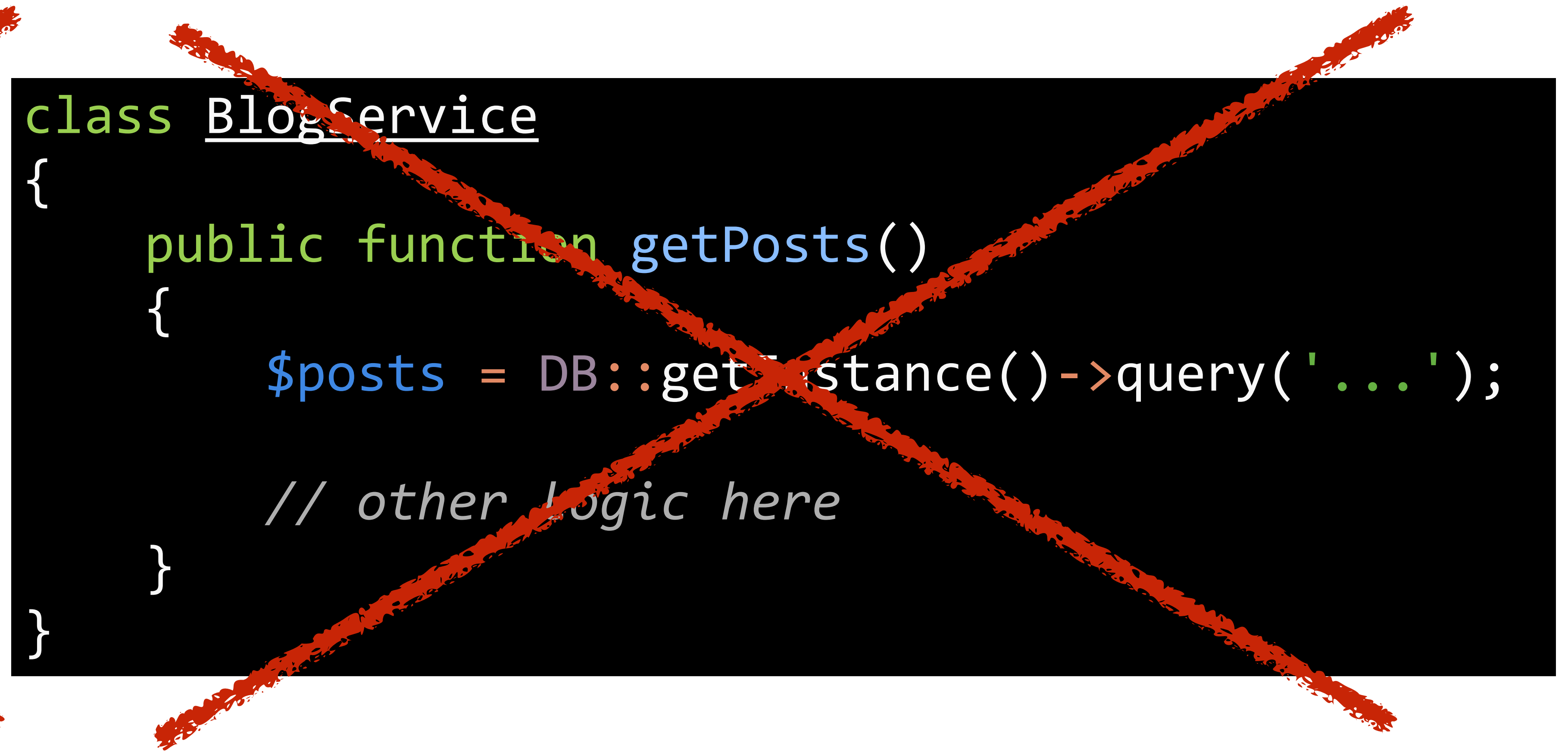


```
class BlogService
{
    public function getPosts()
    {
        global $db;
        $posts = $db->query('...');
        // other logic here
    }
}
```



```
class BlogService
{
    public function getPosts()
    {
        $posts = DB::query('...');

        // other logic here
    }
}
```



```
class BlogService
{
    public function getPosts()
    {
        $posts = DB::getInstance()->query('...');

        // other logic here
    }
}
```

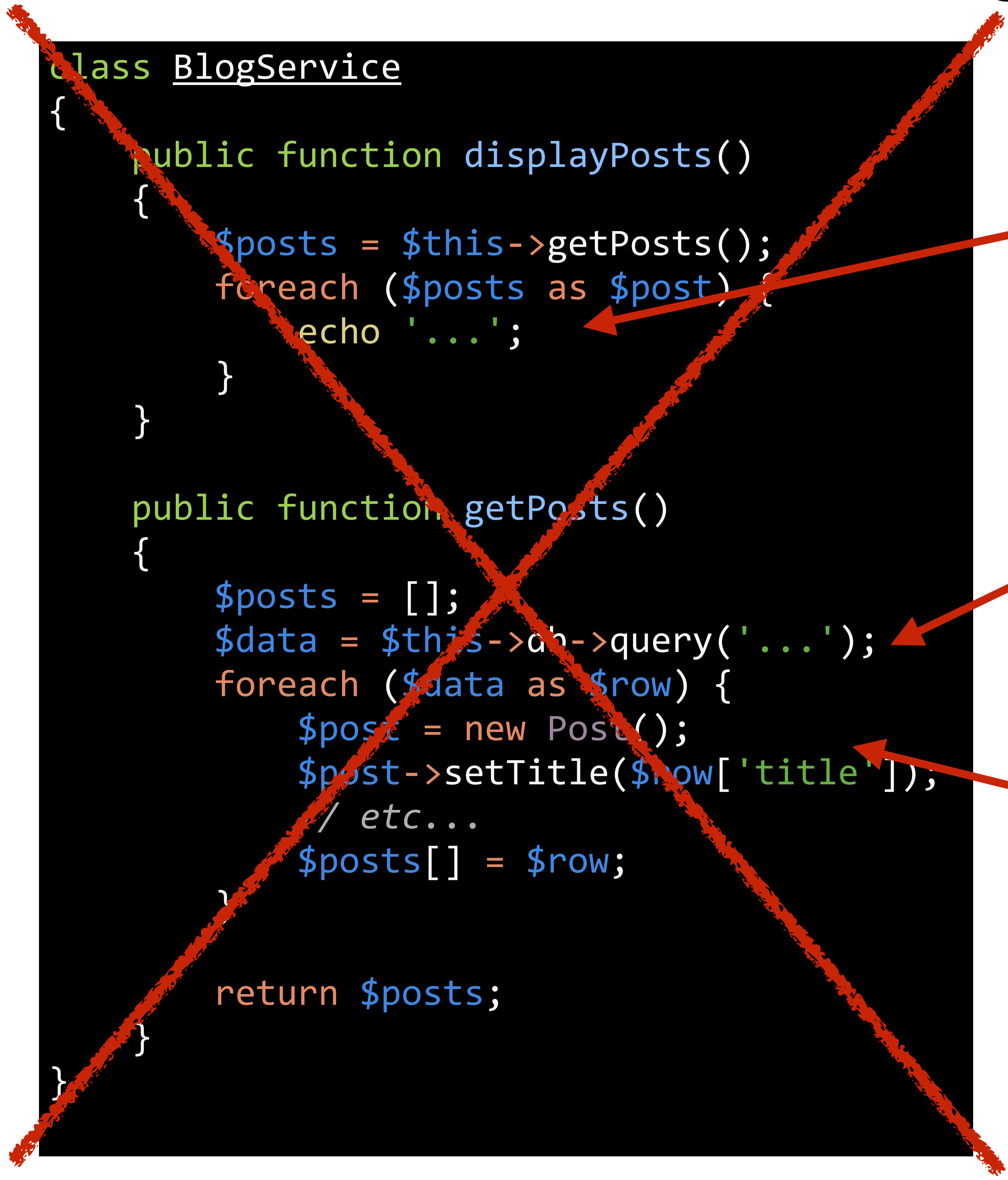
These are great ways to make your code really hard to test!

This is supposed to be easy.

Inject those dependencies instead!

Testability

Classes that do too much



```
class BlogService
{
    public function displayPosts()
    {
        $posts = $this->getPosts();
        foreach ($posts as $post) {
            echo '...';
        }
    }

    public function getPosts()
    {
        $posts = [];
        $data = $this->db->query('...');
        foreach ($data as $row) {
            $post = new Post();
            $post->setTitle($row['title']);
            // etc...
            $posts[] = $row;
        }

        return $posts;
    }
}
```

Renderer

Data retriever

Data mapper

Too much to do
=
Too much to test

Single Responsibility Principle

"a class should have one, and only one, reason to change"

Fewer reasons to change
=
Fewer reasons to *break*

Metrics

- Code Coverage
- **C**hange **R**isk **A**nalysis and **P**redictions
  - Yes - "CRAP"
  - No, I didn't come up with the name

Code Coverage

- Measures how much of your code is being executed by which part(s) of your unit test suite
- Higher coverage -> Lower risk
    - *Usually!*
- You can have 100% coverage and still have all sorts of problems.

It's a guideline!

Example Time!

Change **R**isk **A**nalysis and **P**redictions

"*designed to analyze and predict the amount of effort, pain, and time required to maintain an existing body of code*" [1]

$$\text{CRAP}(m) = \text{comp}(m)^2 * (1 - \text{cov}(m) / 100)^3 + \text{comp}(m)$$

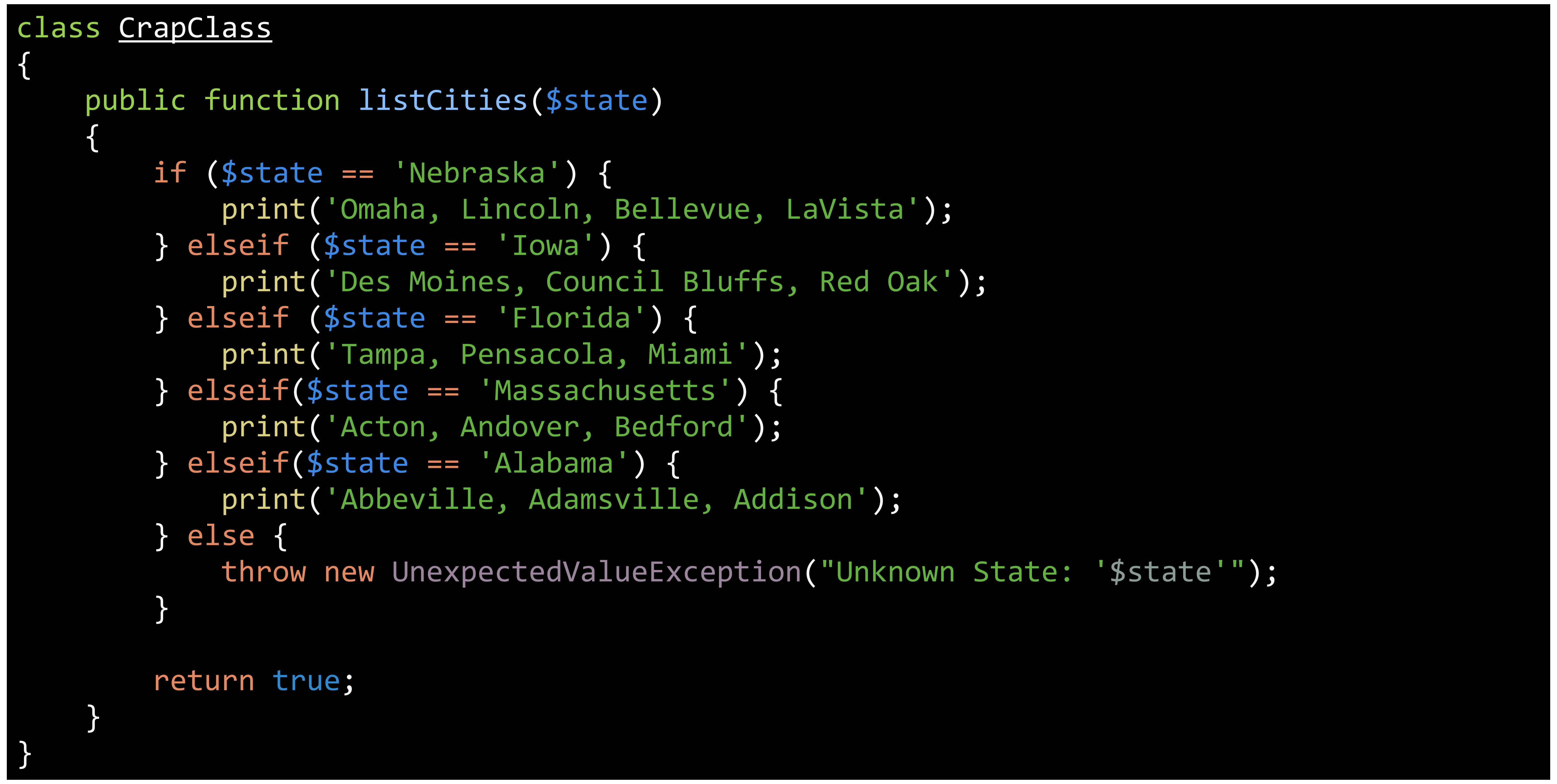
- **m**: a *method*
- **comp**: the method's "cyclomatic ***comp***lexity"
  - A measurement of how "complex" a method is based on how many "decisions" can be made in it.
  - No decisions = complexity of 1
- **cov**: the method's code ***cov***erage

Simplified formula: high complexity + few tests = crap

1 http://www.artima.com/weblogs/viewpost.jsp?thread=210575

CRAPpy Example

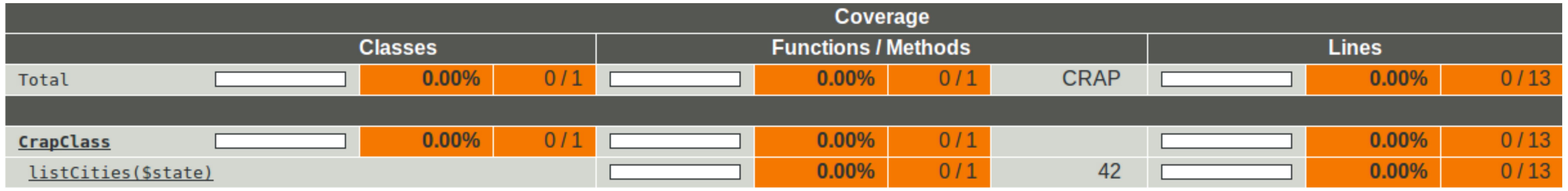
First, write some CRAPpy code...



```
class CrapClass
{
    public function listCities($state)
    {
        if ($state == 'Nebraska') {
            print('Omaha, Lincoln, Bellevue, LaVista');
        } elseif ($state == 'Iowa') {
            print('Des Moines, Council Bluffs, Red Oak');
        } elseif ($state == 'Florida') {
            print('Tampa, Pensacola, Miami');
        } elseif($state == 'Massachusetts') {
            print('Acton, Andover, Bedford');
        } elseif($state == 'Alabama') {
            print('Abbeville, Adamsville, Addison');
        } else {
            throw new UnexpectedValueException("Unknown State: '$state'");
        }

        return true;
    }
}
```

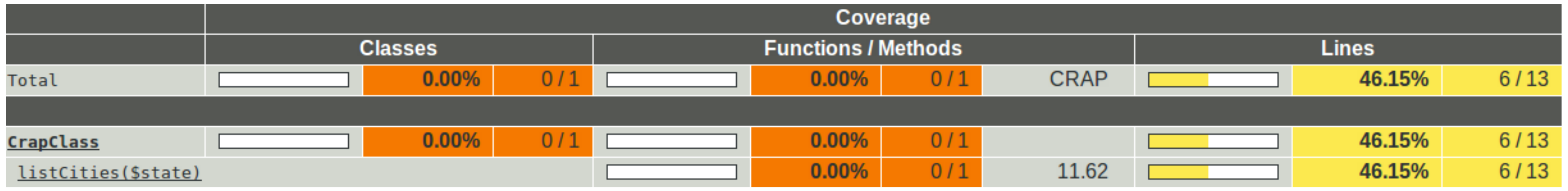
... but don't test it



	Coverage								
	Classes		Functions / Methods				Lines		
Total	0.00%	0 / 1	0.00%	0 / 1	CRAP		0.00%	0 / 13
CrapClass	0.00%	0 / 1	0.00%	0 / 1			0.00%	0 / 13
listCities($state)			0.00%	0 / 1	42		0.00%	0 / 13

Credit: http://www.levihackwith.com/how-to-read-and-improve-the-c-r-a-p-index-of-your-code/

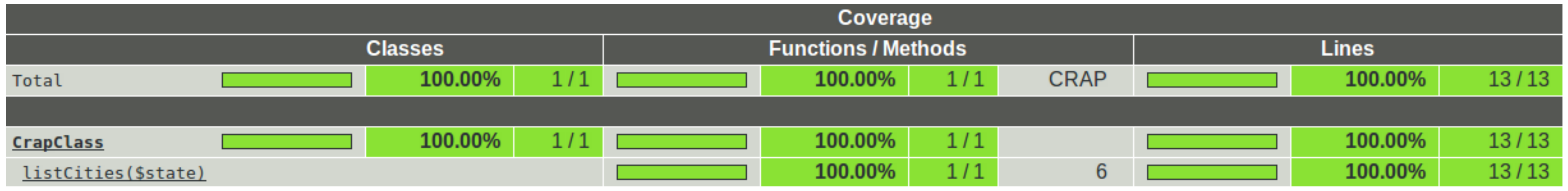
But what if we add some tests?



	Coverage									
	Classes			Functions / Methods				Lines		
Total		0.00%	0 / 1		0.00%	0 / 1	CRAP		46.15%	6 / 13
CrapClass		0.00%	0 / 1		0.00%	0 / 1			46.15%	6 / 13
listCities($state)					0.00%	0 / 1	11.62		46.15%	6 / 13

46.15% code coverage dropped the CRAP score from 42 to 11.62!

What if we have 100% coverage?

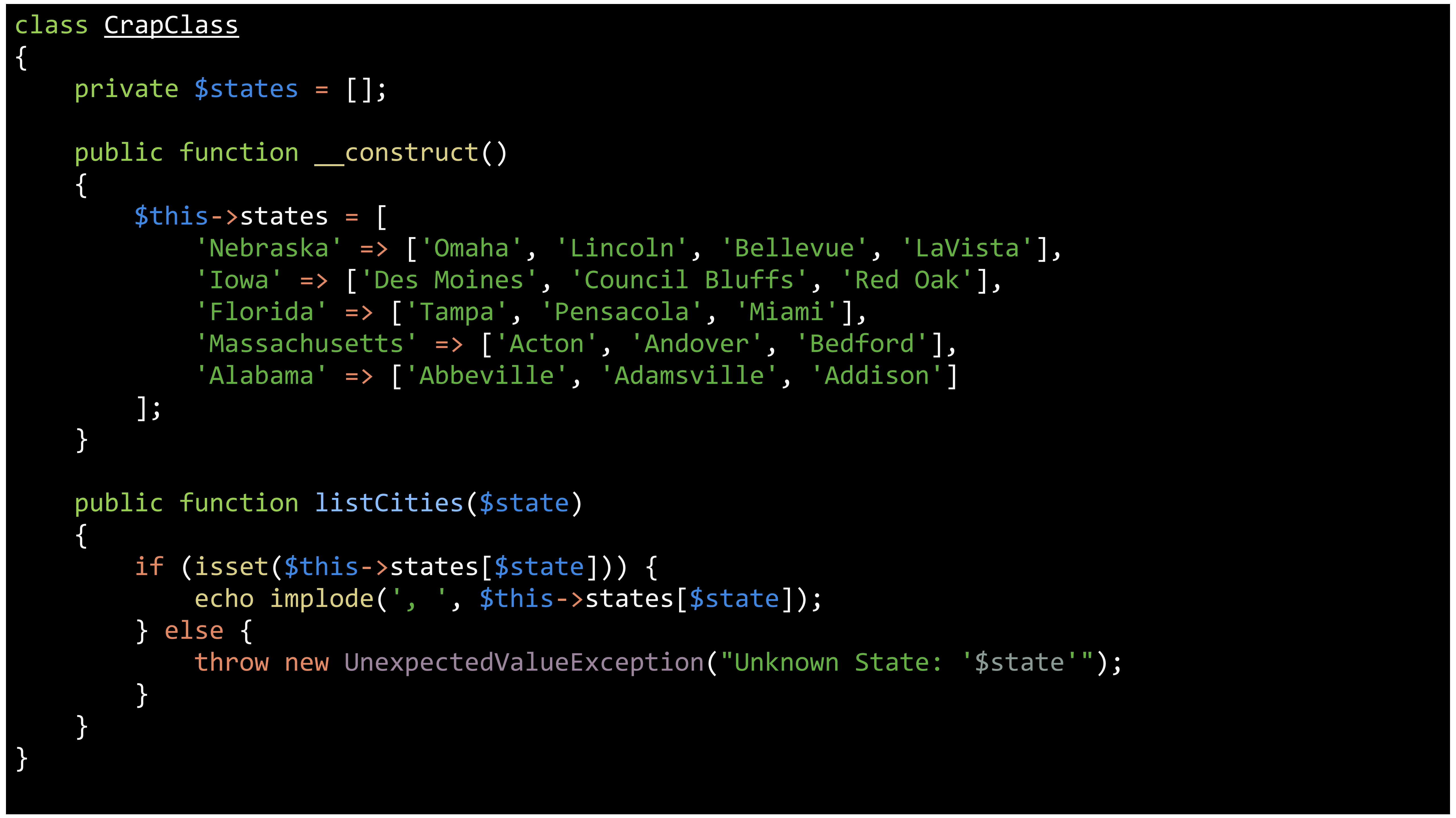


	Coverage									
	Classes			Functions / Methods				Lines		
Total		100.00%	1 / 1		100.00%	1 / 1	CRAP		100.00%	13 / 13
CrapClass		100.00%	1 / 1		100.00%	1 / 1			100.00%	13 / 13
listCities($state)					100.00%	1 / 1	6		100.00%	13 / 13

Adding 100% test coverage dropped the C.R.A.P. score from 42 to 6!

This doesn't mean the code is less "bad" (because it really *is*!)
It just means that there is *less risk* involved in changing it.

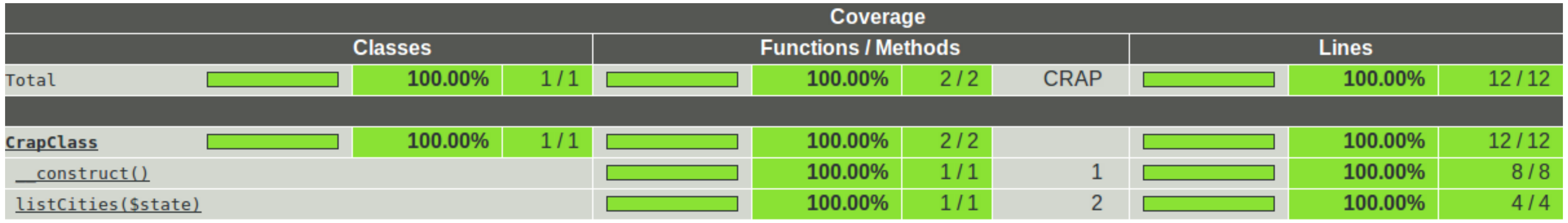
What if we also made our code less complex?



```php
class CrapClass
{
    private $states = [];

    public function __construct()
    {
        $this->states = [
            'Nebraska' => ['Omaha', 'Lincoln', 'Bellevue', 'LaVista'],
            'Iowa' => ['Des Moines', 'Council Bluffs', 'Red Oak'],
            'Florida' => ['Tampa', 'Pensacola', 'Miami'],
            'Massachusetts' => ['Acton', 'Andover', 'Bedford'],
            'Alabama' => ['Abbeville', 'Adamsville', 'Addison']
        ];
    }

    public function listCities($state)
    {
        if (isset($this->states[$state])) {
            echo implode(', ', $this->states[$state]);
        } else {
            throw new UnexpectedValueException("Unknown State: '$state'");
        }
    }
}
```



	Coverage									
	Classes			Functions / Methods				Lines		
Total	🟩	100.00%	1 / 1	🟩	100.00%	2 / 2	CRAP	🟩	100.00%	12 / 12
CrapClass	🟩	100.00%	1 / 1	🟩	100.00%	2 / 2		🟩	100.00%	12 / 12
__construct()				🟩	100.00%	1 / 1	1	🟩	100.00%	8 / 8
listCities($state)				🟩	100.00%	1 / 1	2	🟩	100.00%	4 / 4

This is *a lot* easier to maintain and less risky to change!

Use these metrics to help guide design

- Are you finding it really difficult to cover 100% of a particular method or class?
- Is a particular method's CRAP score really high compared to others in your project?

Your code is trying to tell you something!

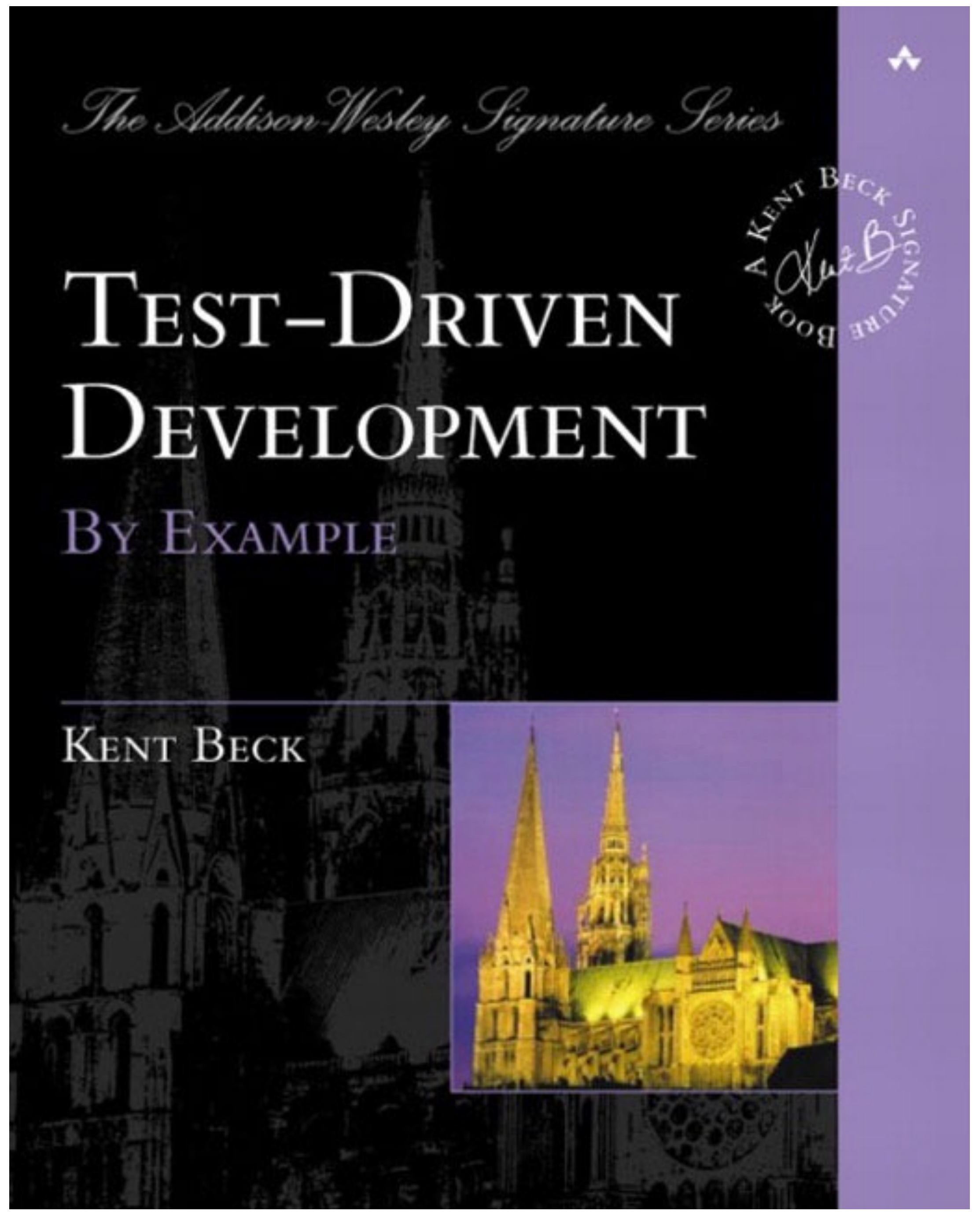
LISTEN!

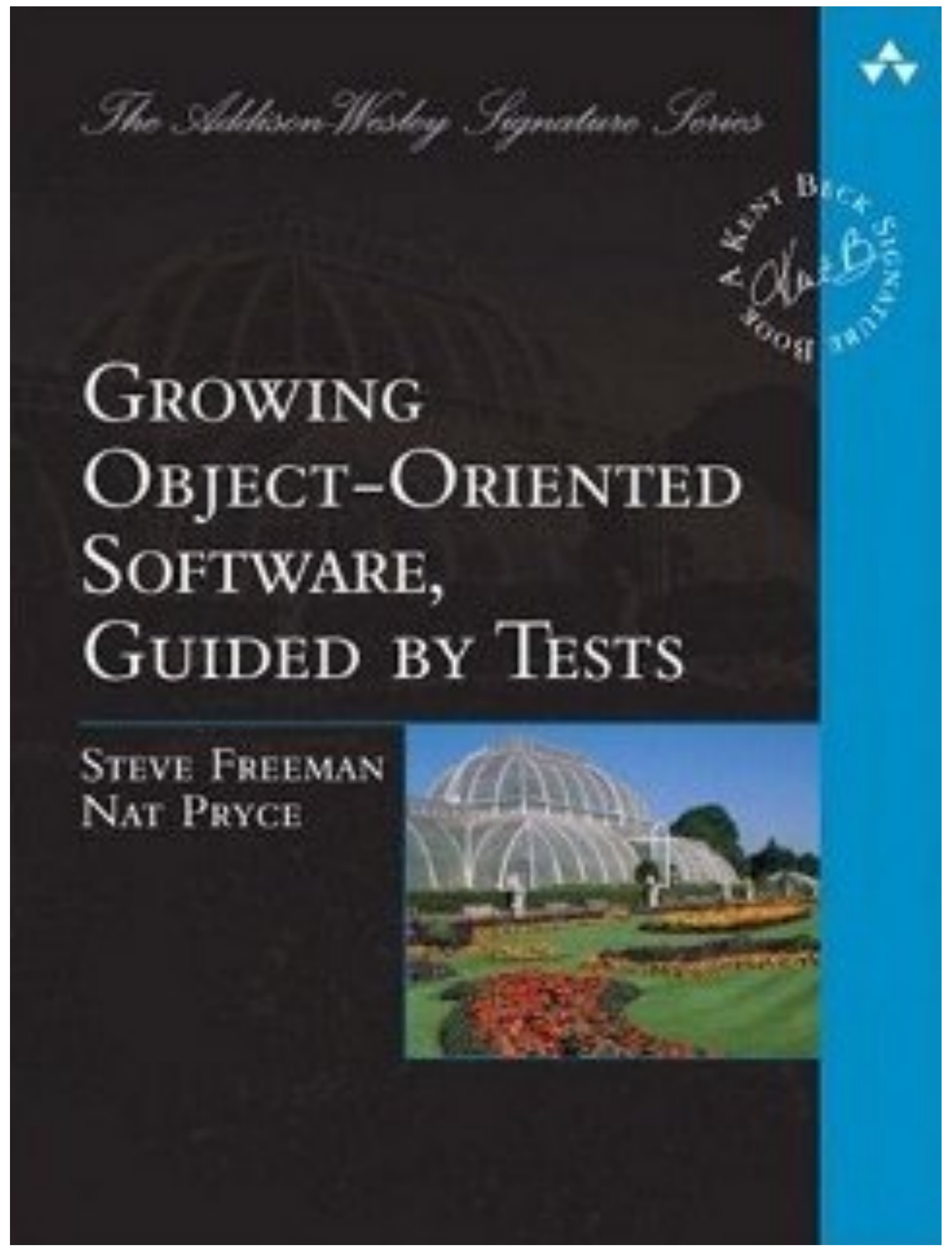
Tips and Tricks

- Save keystrokes with a **phpunit.xml** configuration file
  - And avoid having to `m::close()` with Mockery
- Execute common prep code using **setUp** and **tearDown**
- Run the same test with different sets of input using **Data Providers**
- Mock PDO
- "Mock" Global PHP functions

Example Time!

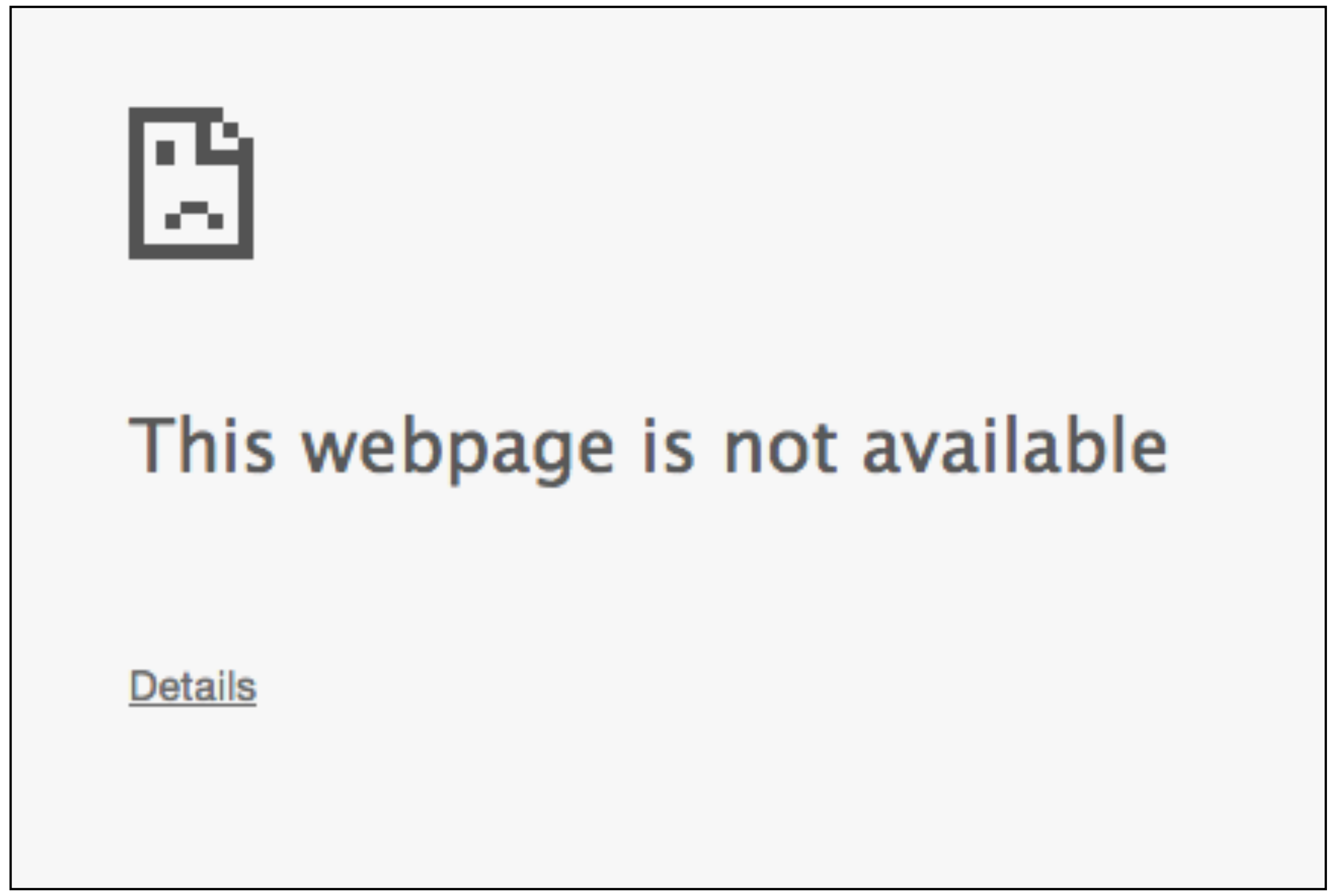
You should read these books


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce

Closing Thought

"*But my change had nothing to do with that!*"


This webpage is not available

Details

... said every developer ever after breaking Production

DON'T BE THAT DEVELOPER!

Unit Testing PHP Applications

Michael Moussa | @michaelmoussa

https://github.com/michaelmoussa/ssp2015-unit-testing

https://joind.in/13437