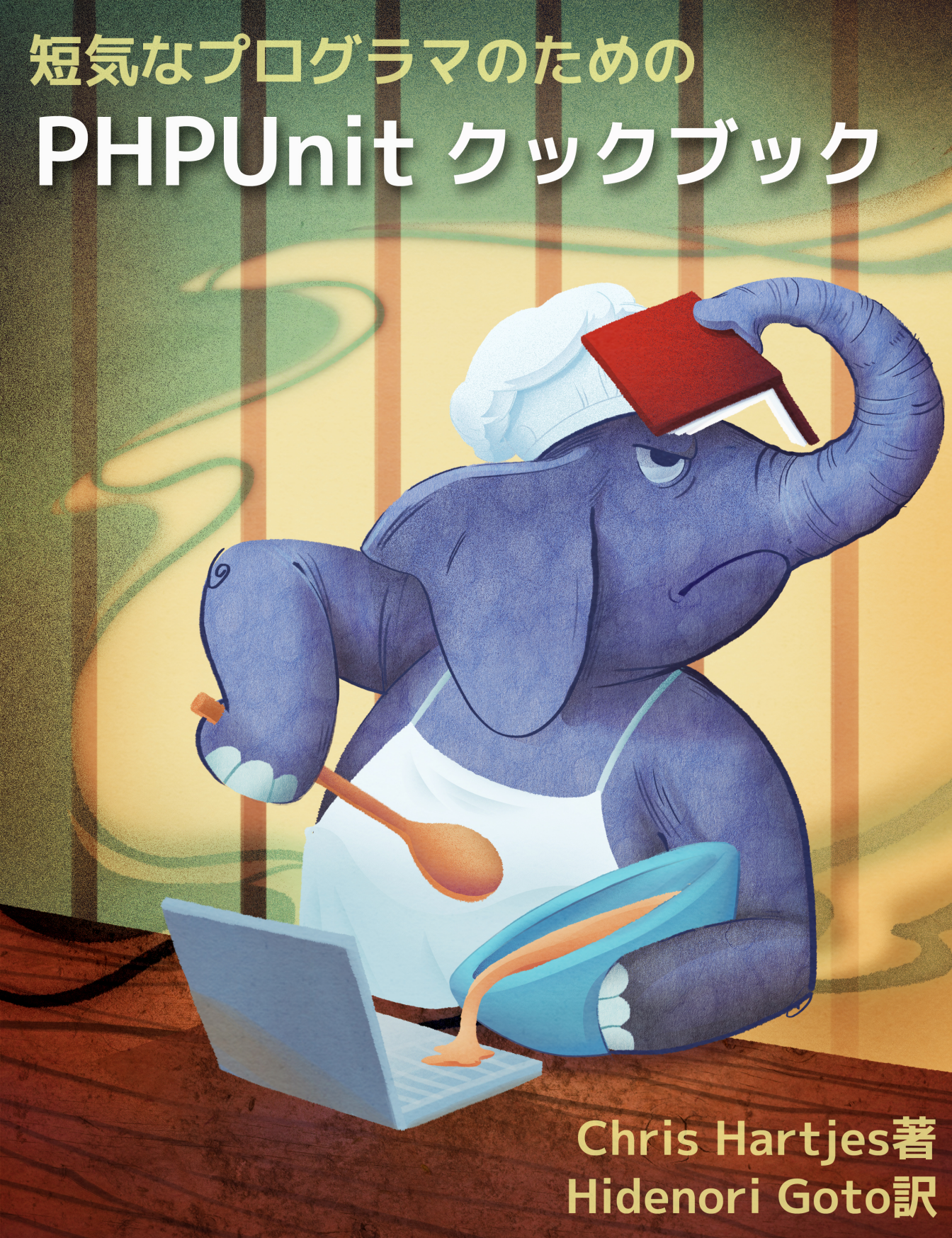
短気なプログラマのための

PHPUnit クックブック



Chris Hartjes著
Hidenori Goto訳

短気なプログラマのための **PHPUnit** クックブック

The Grumpy Programmer's PHPUnit Cookbook

Chris Hartjes and Hidenori Goto

This book is for sale at http://leanpub.com/grumpy-phpunit-jp

This version was published on 2015-01-27

Contents

1. まえがき

前著 “The Grumpy Programmer’s Guide To Building Testable Applications In PHP” を書いた当初の目的は、テストしやすいコードの書き方を多くの開発者に伝えることだった。実は、テスティングツールの使い方については、すでにたくさんの情報があると思っていた。

結果的には、私の考えは部分的にしか正しくなかった。

検索エンジンを使うと、特定のタスクを実行するための方法までは分かる。しかし、表面的な解決策以上の情報を提供しているまとまった情報源は見つけられなかった。

私はさらに調査を進めた。私は、すばらしい製品開発コースに登録して、そこでさらに調査を進める方法を学んだ。実際に現場で PHPUnit を使い、PHP コードのテストを記述しようとすると、かなり苦痛が伴う。そのような苦痛を確実に和らげるソリューションを私は開発し始めた。

その結果がこの本になった。テストを記述するためのコード例の他、私が下したさまざまな決定に関して説明している。

この本は、最初から最後まで目を通さなければならない類の本ではない。気が向いたどこか 1 つの章だけを参照して、テスティングプロセスの一部を確実に理解すればいい。

いつでも、フィードバックは大歓迎。Twitter と App.net (@grmpyprogrammer)、または E メール（chartjes@littlehart.net）でご連絡を。

2. データプロバイダ

2.1 データプロバイダを使う理由

開発者の主な目標の 1 つは、直面している特定の問題を解決するために書くコードを、常に必要最低限にすることだろう。これは、テストの場合も同じである。テストは単なるコードでしかない。

オレがテストについて最初に学んだことの 1 つは、広範囲に及ぶテストスイートを書いていて得られた教訓だが、テストに重複がないかを常に探せということだ。テストが重複している状況を説明しよう。

プログラマなら FizzBuzz[1] の問題はご存知だろう。採用面接の中で、解決すべき問題としてよく出されるからかもしれないが。FizzBuzz 問題は、プログラミングの基本について多く触れているので、提起するには良い問題だと思う。

FizzBuzz のテストを書くときに開発者がしたいことは、値一式を入出力の値として渡して、FizzBuzz の仕様を満たすことを確認することだ。このために、テストする入出力の値だけが異なり、それ以外はまったく同じであるようなテストを複数書いてしまうかもしれない。これを簡素化するための機能が、データプロバイダである。

データプロバイダを使うと、テスト用のデータセットを、パラメータとしてテストメソッドに渡すことができる。データプロバイダを定義するには、データセットを使うテストクラスにデータプロバイダとなるメソッドを作成し、テストメソッドのパラメータと一致する値の配列を返すようにする。

複雑に聞こえるかもしれないが、実際はそうでもない。例を見てみよう。

2.2 すべてのテストを見る

データプロバイダのことを知らない場合、FizzBuzz のテストを次のように書くだろう。

[1] http://en.wikipedia.org/wiki/FizzBuzz

```php
<?php
class FizzBuzzTest extends PHPUnit_Framework_Testcase
{
    public function setup()
    {
        $this->fb = new FizzBuzz();
    }

    public function testGetFizz()
    {
        $expected = 'Fizz';
        $input = 3;
        $response = $this->fb->check($input);
        $this->assertEquals($expected, $response);
    }

    public function testGetBuzz()
    {
        $expected = 'Buzz';
        $input = 5;
        $response = $this->fb->check($input);
        $this->assertEquals($expected, $response);
    }

    public function testGetFizzBuzz()
    {
        $expected = 'FizzBuzz';
        $input = 15;
        $response = $this->fb->check($input);
        $this->assertEquals($expected, $response);
    }

    function testPassThru()
    {
        $expected = '1';
        $input = 1;
        $response = $this->fb->check($input);
        $this->assertEquals($expected, $response);
```

```
39      }
40  }
```

このテストコードから、以下のパターンが見て取れる。

- 複数の入力値
- セットアップと実行が非常に似ているテスト
- 同じアサーションが繰り返し使われている

2.3 データプロバイダの作成

データプロバイダは、テストクラス内にある別のメソッドで、主に入出力の値として使うデータの配列を返す。各データ自体も配列である。PHPUnit により、データプロバイダが返したデータは、テストメソッドのシグネチャに合うパラメータへ変換される。

```
1   <?php
2   public function fizzBuzzProvider()
3   {
4       return array(
5           array(1, '1'),
6           array(3, 'Fizz'),
7           array(5, 'Buzz'),
8           array(15, 'FizzBuzz')
9       );
10  }
```

プロバイダの関数名にはどんな名前でも使えるが、常識的な名前を付けること。テストが失敗したときに、'ex1ch2' のような意味のない名前のデータプロバイダでは困ったことになる。

データプロバイダを使うには、テストメソッドの docblock にアノテーションを記述する。アノテーションのパラメータとして、データプロバイダのメソッド名を指定する。

```php
<?php
/**
 * Test for our FizzBuzz object
 *
 * @dataProvider fizzBuzzProvider
 */
public function testFizzBuzz($input, $expected)
{
    $response = $this->fb->check($input);
    $this->assertEquals($expected, $response);
}
```

テストは 1 つだけになり（メンテナンスするコードが少ない）、データプロバイダを介して好きなだけシナリオを追加できる（すばらしい）。この例では、実際に必要なテストだけになるように、書いているテストコードを批判的に分析するという教訓も含んでいる。

データプロバイダを使用すると、PHPUnit はプロバイダから渡されるデータ 1 件ごとにテストメソッドを実行する。テストが失敗すると、テストに失敗したデータの、データセット連想配列におけるインデックスが表示される。

2.4 複雑な例

簡単なデータプロバイダしか無理だ、という気分にならないでほしい。必要なのは、配列の配列を返すことだけなんだ。各データセットは、テストメソッドが期待しているパラメータと一致する。次に複雑な例を示す。

```php
<?php
public function complexProvider()
{
    // CSV ファイルからデータを読み取る
    $fp = fopen("./fixtures/data.csv");
    $response = array();

    while ($data = fgetcsv($fp, 1000, ",")) {
        $response[] = array($data[0], $data[1], $data[2]);
    }

```

```
12      fclose($fp);
13
14      return $response;
15  }
```

データプロバイダで許可されることだけしかできないと思う必要はない。目標は、テストに役に立つデータセットを作成することである。

2.5 データプロバイダのトリック

データプロバイダが返すのは連想配列なので、より分かりやすいキーにしておけば、デバッグしやすくなる。たとえば、FizzBuzz テストのデータプロバイダを次のようにリファクタリングできる。

```
1   <?php
2   return array(
3       'one'      => array(1, '1'),
4       'fizz'     => array(3, 'Fizz'),
5       'buzz'     => array(5, 'Buzz'),
6       'fizzbuzz' => array(15, 'FizzBuzz')
7   );
```

また、データプロバイダは、同じクラス内のメソッドである必要はない。他のクラスのメソッドでも、public と定義されていれば使用できる。名前空間も使用できる。以下に 2 つ例を示す。

- `@dataProvider Foo::dataProvider`
- `@dataProvider Grumpy\Helpers\Foo::dataProvider`

これにより、データプロバイダだけを定義したヘルパークラスを作成して、テスト自体の重複コードの量を減らすことができる。