Метки

.Net CSS Drupal Eclipse Fedora HTTP Java JavaDB Linux MySQL Oracle Oracle Linux PHP PHPUnit PL/SQL QNX Smarty SWT Twitter UAC UTF8 Web Windows7 XML xUnit Zend Framework Безопасность Книги ОСи разное Регулярные выражения СУБД Тестирование

PHPUnit. Часть 01 Автоматические тесты

Статьи -> Программирование -> PHP

PHPUnit. Часть 01 Автоматические тесты

v:1.0 17.03.2010

Перевод статьи PHPUnit Manual. Chapter 1. Automating Tests.
Автор: Sebastian Bergmann
Перевод: Петрелевич Сергей

Предисловие переводчика
Недавно начал изучать PHPUnit (framework семейства xUnit) и с удивлением обнаружил, что на русском языке нет статей про автоматические тесты для самых-самых чайников.
В первой главе документации по PHPUnit на примерах очень доступно рассказывается, что такое автоматическое тестирование.

Даже хорошие программисты допускают ошибки. Разница между хорошим программистом и плохим заключается в том, что хороший программист как можно чаще использует тесты, чтобы найти свои ошибки. Чем раньше Вы начнете тестировать, тем выше Ваши шансы найти ошибку, и тем ниже цена исправления. Это объясняет, почему откладывание тестирования до момента передачи программы заказчику является очень плохой практикой. Большинство ошибок так и не будет найдено, а цена исправления станет такой высокой, что Вам придется составить большой график работы, т.к. сразу Вы не сможете их все исправить.

Тестирование при помощь PHPUnit принципиально не отличается от того тестирования, которое Вы уже выполняете. Это просто другой подход к работе. Разница заключается в следующем: в одном случае Вы просто *проверяете*, что Ваша программа работает как ожидается, в другом - Вы *выполняете серию тестов*, которые представляют собой выполняемые фрагменты кода для автоматической проверки корректности частей (модулей) программного обеспечения.
Эти выполняемые фрагменты кода называются модульными (Unit) тестами.

В этой статье мы проделаем путь от элементарного теста, который просто выводит на экран результат своей работы до полностью автоматизированного теста.
Допустим, нас попросили протестировать встроенный в PHP массив (`array`). На одном из этапов тестирования надо проверить работу функции `count()`. Мы ожидаем, что для только что созданного массива функция `count()` вернет 0. После добавления элемента в массив `count()` должна вернуть `1`.
Пример 1.1 демонстрирует, что мы хотим проверить.

Пример 1.1: Тестирование операторов массива

```
<?php
$fixture = array();
// $fixture is expected to be empty.

$fixture[] = 'element';
// $fixture is expected to contain one
element.
?>
```

```
* This source code was highlighted with Source Code Highlighter.
```

Самый простой способ проверить, получили ли мы то, что хотели - вывести результат работы функции `count()` на экран.
Вывод надо сделать до и после добавления элемента, смотрите Пример 1.2.
Если мы сначала получили `0` а потом `1`, то `array` и `count()` работают как надо.

Пример 1.2: Использование вывода на экран для проверки операторов массива

```
<?php
$fixture = array();
print count($fixture) . "\n";

$fixture[] = 'element';
print count($fixture) . "\n";
?>
```

```
* This source code was highlighted with Source Code Highlighter.
```

Вывод теста:

```
0 1
```

Мы бы хотели перейти от тестов, которые требуют ручной обработки результатов к тестам, которые могут выполняться автоматически.
В Примере 1.3 мы добавим в тест сравнение ожидаемого результата и фактического значения, выведем `ok` если ожидаемый и фактический результаты совпали.
Если вывод будет `not ok`, значит где-то произошла ошибка.

Пример 1.3: Тестирование операторов массива, сравнение ожидаемого результата и фактического значения

```
<?php
$fixture = array();
print count($fixture) == 0 ? "ok\n" :
"not ok\n";

$fixture[] = 'element';
print count($fixture) == 1 ? "ok\n" :
"not ok\n";
?>
```

```
* This source code was highlighted with Source Code Highlighter.
```

Вывод теста:

```
ok ok
```

А сейчас вынесем сравнение ожидаемого и фактического результата в специальную функцию, которая выбрасывает исключение, если условие не выполняется, смотрите Пример 1.4.
Этот подход дает нам два преимущества: написание тестов заметно упрощается, тест генерирует сообщение только в случае ошибки.

Пример 1.4: Использование функции утверждения для тестирования операторов массива

```
<?php
$fixture = array();
assertTrue(count($fixture) == 0);

$fixture[] = 'element';
assertTrue(count($fixture) == 1);

function assertTrue($condition)
{
```

```
    if (!$condition) {
        throw new Exception('Assertion
failed.');
    }
}
?>
```


Тест полностью автоматизирован. Вместо просто *тестирования*, которое мы выполняли в первой версии теста, мы получили *автоматический тест*.

Цель использования автоматических тестов - допускать меньше ошибок в коде.
Даже если Ваш код все еще не идеален, и Вы используете отличные тесты, начните практиковать автоматические тесты и Вы обнаружите значительное сокращение количества ошибок.
Автоматические тесты обеспечат Вам уверенность в своем коде. Опираясь на эту уверенность, Вы сможете вносить более смелые и решительные изменения в свою программу (Рефакторинг), сможете улучшить взаимоотношения с коллегами и клиентами.
Каждый день, уходя домой, Вы будете знать, что программа стала значительно лучше, чем она была утром.

Метки: PHP Web PHPUnit Тестирование xUnit

Комментарии.

Внимание.
Комментировать могут только зарегистрированные пользователи.
Возможно использование следующих HTML тегов: <a>, <b>, <i>,
.

Аноним 11.06.2010 12:38:46

как-то мало

Petrelevich 11.06.2010 20:06:02

как-то мало
Это только первая часть перевода официальной документации, смотрите продолжение...

Пожалуйста авторизуйтесь или зарегистрируйтесь.