Тестирование унаследованного кода в php

Голубев Александр
интернет-агентство “RealWeb”

Обо мне

- Последние три года работаю в компании RealWeb
- Долго работал над проектом AdHands — системой управления интернет рекламой
- Люблю писать тесты

Что такое унаследованный код?

- Код, в котором сложно разобраться
- Код, который сложно поддерживать
- Код, для которого нет тестов
- Код, который хочется изменить

Тестирование унаследованного кода

Прежде, чем начать писать тесты, нужно понять:

- Нужно ли вообще их писать
- Какие тесты писать (unit, интеграционные, …)
- Можно ли изменить код, чтобы тесты писать стало проще

Трудности написания тестов для унаследованного кода

- Большое количество неподменяемых зависимостей

```
public function alert() {
    $service = new System_EventService();
    $service->setSelectorStatus(System_EventService::NEW);
    if ($service->hasEvents()) {
        $alerter = new Alerter();
        $alerter->alert($service->getEvents());
    }
}
```

Трудности написания тестов для унаследованного кода 2

- Тяжёлые конструкторы

```
class StatsCalculator {
    public function __construct() {
        $stats = $this->getStatsFromRemoteServer();
    }
    ...
}
```

Трудности написания тестов для унаследованного кода 3

- Статические вызовы

```
public function notify($message) {
    $user = $this->getUserToNotify();
    Mailer::sendMessageTo($user, $message);
}
```

Трудности написания тестов для унаследованного кода 4

- Использование внешних ресурсов без необходимых уровней абстракции

```
public function uploadFilesToDb($dir) {
    …
    $contents = scandir($dir);
    …
}
```

Зависимости в коде

```
public function alert() {
    $service = new System_EventService();
    $service->setSelectorStatus(System_EventService::NEW);
    if ($service->hasEvents()) {
        $alerter = new Alerter();
        $alerter->alert($service->getEvents());
    }
}
```

Решение. Первая попытка

- Подмена реальных классов «заглушками» с помощью изменения include path
- Запуск каждого теста в отдельном процессе php, чтобы заглушки не пересекались
- Создание набора «заглушек» для каждого теста

Первая попытка. Результаты

- Плюсы:
  - Большая гибкость
- Минусы:
  - Низкая скорость выполнения
  - Большое количество «заглушек», сложно поддерживать
  - Нет возможности использовать Mock библиотеки

Вторая попытка

- Использование расширения runkit
- Подмена не класса, а его поведения

```
runkit_method_redefine(
    'System_EventService', 'getEvents', '',
    'return array(new Event(), new Event());'
);
```

Вторая попытка. Результаты

- Плюсы:
  - Не надо писать заглушки
  - Можно объединить с вариантом из первой попытки
- Минусы:
  - Сложно писать, читать и поддерживать
  - Сложные setUp и tearDown методы у тестов
  - Нет возможности для переиспользования

Удачная попытка

```
public function alert() {
    $service = new System_EventService();
    $service->setSelectorStatus(System_EventService::NEW);
    if ($service->hasEvents()) {
        $alerter = new Alerter();
        $alerter->alert($service->getEvents());
    }
}
```

Что хочется получить? – Управление созданием объектов

Удачная попытка 2

- Получение кода метода с помощью Reflection API
- Исправление места создания объекта на необходимое

  Например:

  $service = new System_EventService();

  на

  $service = new System_EventServiceStub();
- Использование библиотеки, формализующей действия по поиску и замене кода

Удачная попытка. Пример

```
self::inject()
    ->stubFor('System_EventService')
    ->intoClass('CronService')
;

self::inject()
    ->stubFor('System_EventService')
    ->intoMethod('alert', 'CronService')
;
```

Удачная попытка. Результаты

- Плюсы:
  - Большая гибкость — вместо создания класса-зависимости можем создать свой класс или использовать mock
  - «Заглушки» можно переиспользовать
  - Тесты проще для понимания и поддержки
- Минусы:
  - Тестируется не тот код, который реально исполняется в приложении

Можно ли положиться на такие тесты?

- Задача — покрыть тестами код, чтобы можно было рефакторить
- После рефакторинга проблема исчезнет
- Можно заменять зависимости не в тестируемом коде, а глубже — в самих классах-зависимостях

Подавление методов

```
self::suppress()->constructor()->inClass('LibLoader');
self::suppress()->method('loadLib')->inClass('ResourceManager');
```

Применение:

- Тестирование методов классов с «тяжелыми» действиями в конструкторах или отдельных методах
- Тестирование классов-наследников библиотечных классов

Изменение поведения методов

```
self::stub()->method('alert')->inClass('Alerter')
    ->returnSelf();
self::stub()->method('getData')->inClass('Provider')
    ->returnCallback(function() { ... });
```

Применение:

- Использование возможностей Mock библиотек без подмены зависимостей
- Замена простым классам-заглушкам

Изменение поведения функций

```
self::stub()->userFunction('file_get_contents')
    ->returnValue(true);
```

Применение:

- Тестирование методов, использующих стандартные функции php

Изменение результата цепочки вызовов

Для случая:

```
$controller->getHelper('View')->getView()->getVars()->clear();
```

Вместо написания 4х Mock'ов или создания 4х заглушек:

```
self::stub()->chain(array('getHelper', 'getView', 'getVars', 'clear'))
    ->inClass('Controller')->returnValue(true);
```

Итого

- Описанный подход — не замена обычным методам тестирования, а дополнение для сложных ситуаций
- Мы используем при написании новых тестов, на новый код
- Перед применением стоит ознакомиться с runkit — его возможностями, ограничениями и особенностями его установки

Ссылки

- Runkit extension for PHP:
  https://github.com/zenovich/runkit

- Использованная библиотека:
  https://github.com/realweb-team/pstub

Вопросы?