Artikel Test-Organisation in grösseren Magento(1)-Projekten

Ein Tipp von unserem Entwickler Claudio Kressibucher

Köln/Bergisch Gladbach

www.kennziffer.com GmbH
Friedrich-Ebert-Straße 75
51429 Bergisch Gladbach

Telefon 02204-84266 0

Münster

www.kennziffer.com GmbH
Königstraße 32-33
48143 Münster

Telefon 0251-590 595 81

Aachen

team in medias GmbH
Karl-Friedrich-Straße 74
52077 Aachen

Telefon 0241-980 998 70

Kapstadt/Südafrika

Marc Koebler
3 Sillery Rd
7935 Bergvliet-Cape Town

Telefon 02204-84266 2

Artikel Test-Organisation in grösseren Magento(1)-Projekten

Tests mit EcomDev_PHPUnit

Das Testing Framework von EcomDev für Magento 1 registriert genau eine Testsuite für die gesamte Magento-Installation. Die verschiedenen Extensions können sich für diese Testsuite registrieren, indem sie in der Magento Konfiguration phpunit/suite/modules einen XML-Knoten für ihr Modul hinzufügen (beschrieben in EcomDev_PHPUnit Manual auf Seite 5).

```
<phpunit>
      <suite>
        <modules>
            <Your_ModuleName />
        </modules>
      </suite>
</phpunit>
```

Die Erfahrung

Wir haben nun bei unseren Projekten bei den kundenspezifischen Modulen genau dies gemacht. Neben den selbst entwickelten Modulen nutzen wir auch community-Module, die z.T. Tests für EcomDev enthalten.

Erstes Problem: Ein Class-Rewrite lässt Test in community-Modul fehlschlagen

Anfangs hat das noch zufriedenstellend geklappt. Irgendwann mussten wir dann aber einen Block-Rewrite für ein Community-Modul definieren. Der Test dieses Moduls ist nun fehlgeschlagen, da es geprüft hatte, ob für

einen bestimmten block-alias genau die vom Modul vorgesehene Klasse instanziiert wurde.

Was kann man tun? Wir können ja nicht Code im Community-Modul ändern. Der Gedanke war, nur noch die eigenen Module zu testen. Dies versuchten wir zu lösen, indem wir das Bootstrap script angepasst haben, um per Regex (geprüft gegen die Modul-Namen) alle Module auszuschliessen, die nicht mit unserem vendor-Namen beginnen. Die Testsuite lief nun wieder erfolgreich durch und dauerte ausserdem weniger lang, da weniger Tests (aber nach unserer Meinung die für uns wichtigen) ausgeführt wurden.

Zweites Problem: Wachsende Testsuite, zu langsam

Doch auch wenn nur die eigenen Tests ausgeführt werden, dauert dies mitunter sehr lange. Insbesondere, da man mit EcomDev_PHPUnit nicht nur Unit-Tests im eigentlichen Sinne, sondern v.a. Integrations-Tests mit zum Teil aufwändigen Datenbank-Fixtures schreiben kann. Natürlich sollte man von Zeit zu Zeit mal die gesamte Testsuite durchlaufen lassen (idealerweise vom CI-Server), doch für die Entwicklung an einem spezifischen Modul oder Feature ist das nicht unbedingt erforderlich.

Die Stärken von PHPUnit

Aus meiner Sicht die stärkste Eigenschaft von PHPUnit ist die Eignung, trotz des "Unit" im Namen, eigentlich alle Arten von Tests ausführen zu können. Diese Tests lassen sich sehr gut in verschiedenen Testsuites organisieren. Falls die Initialisierung unterschiedlich ist (z.B. bei Unit- vs. Akzeptanztests), kann dafür eine separate Konfiguration (phpuni.xml file) mit spezifischem bootstrap-Script erstellt werden. So kann man über die Wahl des Konfigurations-Files und die Testsuite auch nur einzelne Teile

der gesamten Suite ausführen, was während der Entwicklung enorm Zeit einspart.

Beispiel-Projekt

Unsere neueren Projekte verwenden composer, um sowohl Abhängigkeiten von Drittherstellern wie auch projektspezifische in das Hauptprojekt zu laden. Das Projekt sieht dabei dann in etwa so aus:

```
$ tree
.
├── composer.json
├── composer.lock
├── public
│   └── app
│       ├── code
│       │   └── local
│       │       ├── Inmedias
│       │       └── The_Module -> ../../../../vendor/Inmedias/The_Module/code
│       └── Mage.php
└── vendor
    └── Inmedias
        └── The_Module
            └── code
```

Da das Module Inmedias/The_Module prinzipiell unabhängig vom Projekt ist, bietet es sich an, die Test-Konfiguration auch im composer-package zu verwalten anstatt im Hauptprojekt.

Nachfolgen ein Beispiel, wie diese Test-Konfiguration für ein einfaches Magento-Modul aussehen kann:

```
$ tree
.
├── composer.json
├── modman
├── readme.md
├── src
│   ├── app
│   │   ├── code
│   │   │   └── community
│   │   │       └── Inmedias
│   │   │           └── MongoLogger
│   │   │               ├── etc
│   │   │               │   └── config.xml
│   │   │               ├── Helper
│   │   │               │   └── Data.php
│   │   │               ├── LoggerFactoryInterface.php
│   │   │               └── Model
│   │   │                   ├── Factory.php
│   │   │                   └── LoggerProxy.php
│   │   └── etc
│   │       └── modules
│   │           └── Inmedias_MongoLogger.xml
│   └── lib
│       ├── DefaultStructure.php
│       ├── FileLogger.php
│       ├── MongoLogger.php
│       ├── StructuredLogger.php
│       └── StructureProcessorInterface.php
└── test
    ├── integration
    │   ├── bootstrap.php
    │   ├── Helper
    │   │   └── DataTest.php
    │   ├── Model
    │   │   ├── FactoryTest
```

```
│   │   │   └── fixtures
│   │   │       ├── file_backend.yaml
│   │   │       └── mongo_backend.yaml
│   │   ├── FactoryTest.php
│   │   ├── LoggerProxyTest
│   │   │   └── fixtures
│   │   │       └── configLoggerFactory.yaml
│   │   └── LoggerProxyTest.php
│   └── phpunit.xml
├── lib
│   ├── bootstrap.php
│   ├── DefaultStructureTest.php
│   ├── FileLoggerTest.php
│   └── phpunit.xml
├── log
└── test_package.sh
```

Einige Erläuterungen dazu:

- Production code (und was dazu gehört) ist unter ./src
- Tests selbst, bootstrap files und phpunit.xml Konfigurationen sind unter ./test zu finden. Dises Verzeichnis ist unterteilt in die Unterverzeichnisse integration und lib, die je eine Datei bootstrap.php und eine Datei phpunit.xml enthalten. In diesem Modul haben wir Magento-Spezifischen Code unter src/app liegen, und allgemein verwendbaren Code (ohne Abhängigkeiten zu Magento) in eine "Library" ausgelagert.
- Für die Library benötigen wir kein initialisiertes Magento und keine Datenbank. Das bootstrap file inkludiert lediglich autoload.php von composer. Die Tests sind klassische Unit-Tests.
- Für die "Integration", also die Magento-spezifischen Teile des Moduls, verwenden wir EcomDev_PHPUnit. Allerdings definieren wir

eine eigene, "directory" basierte Testsuite anstelle der von EcomDev_PHPUnit dynamisch aus der Konfiguration erstellter Testsuite.

Konfiguration Library-Tests:

```
<?xml version="1.0"?>
<!-- Unit tests for the library (this runs without magento) -->
<phpunit bootstrap="bootstrap.php">
    <testsuites>
        <testsuite name="MongoLogger library">
            <directory suffix="Test.php">.</directory>
        </testsuite>
    </testsuites>

    <filter>
        <whitelist>
            <directory suffix=".php">../../src/lib</directory>
        </whitelist>
    </filter>
</phpunit>
```

Konfiguration Integration-Tests:

```
<?xml version="1.0"?>
<!-- Integration tests for this module, using ecomdev_phpunit -->
<phpunit cacheTokens="true" bootstrap="bootstrap.php">

    <listeners>
        <listener
file="../../../../ecomdev/ecomdev_phpunit/app/code/community/EcomDev/PHPUn
it/Test/Listener.php" class="EcomDev_PHPUnit_Test_Listener" />
    </listeners>

    <testsuites>
        <testsuite name="MongoLogger_Integration">
            <directory suffix="Test.php">.</directory>
        </testsuite>
    </testsuites>

    <filter>
        <whitelist>
            <directory suffix=".php">../../src/app</directory>
        </whitelist>
    </filter>
</phpunit>
```

Fazit

PHPUnit selbst bietet mit den Möglichkeiten der xml-Konfigurationsdatei und insbesondere der testsuites gute Werkzeuge für eine fein-granulare Organisation verschiedener Tests. EcomDev_PHPUnit ist ein gutes Framework, um Magento-spezifischen Code zu testen, allerdings ist meiner Ansicht nach die dynamische Generierung einer einzigen Testsuite aus der Magento-Konfiguration meist nicht der ideale Ansatz. Meine Empfehlung ist, die Tests mittels phpunit Konfiguration und insbesondere Verzeichnis-basierten Testsuites zu organisieren.

Der Autor

Claudio Kressibucher

Entwickler

claudio.kressibucher@inmedias.de

Kein Shop kann komplex genug für ihn sein. Mit schweizer Präzision bringt der Informatiker jede Extension in Magento zum Laufen.

Als Sicherheitsjunkie sorgt er nicht nur bei Magento für Beruhigung beim Kunden, sondern auch bei der Einrichtung seiner Soundanlage ist jedes Kabel an seinem Platz.