integer_net />

Professionelle Webentwicklung

Fabian Schmengler

Pragmatisches Unit Testing

Meet Magento DE 2015

Agenda

- Grundlagen: Warum automatisierte Tests?
- Tests und TDD mit Magento: Überblick und Beispiel
- Ausblick auf Magento 2

Warum Automatisiertes Testen?

- „Beweis“, dass Code wie erwartet funktioniert
- Unerwünschte Nebeneffekte rechtzeitig erkennen
- Refaktorisieren ohne Angst

Test Granularität

- Unit Test
  - White Box: Komponenten isoliert testen
- Integration Test
  - White Box: Zusammenspiel von Komponenten testen
- System Test
  - Black Box: Funktionaler Test des Gesamtsystems

Testgetriebene Entwicklung (TDD)

1. Schreibe einen fehlschlagenden Test
2. Schreibe die minimale Implementierung, die den Test erfolgreich macht
3. Wiederhole (1) und (2) bis Feature komplett
4. Refaktorisiere bis Code sauber

Welchen Wert haben automatisierte Tests?

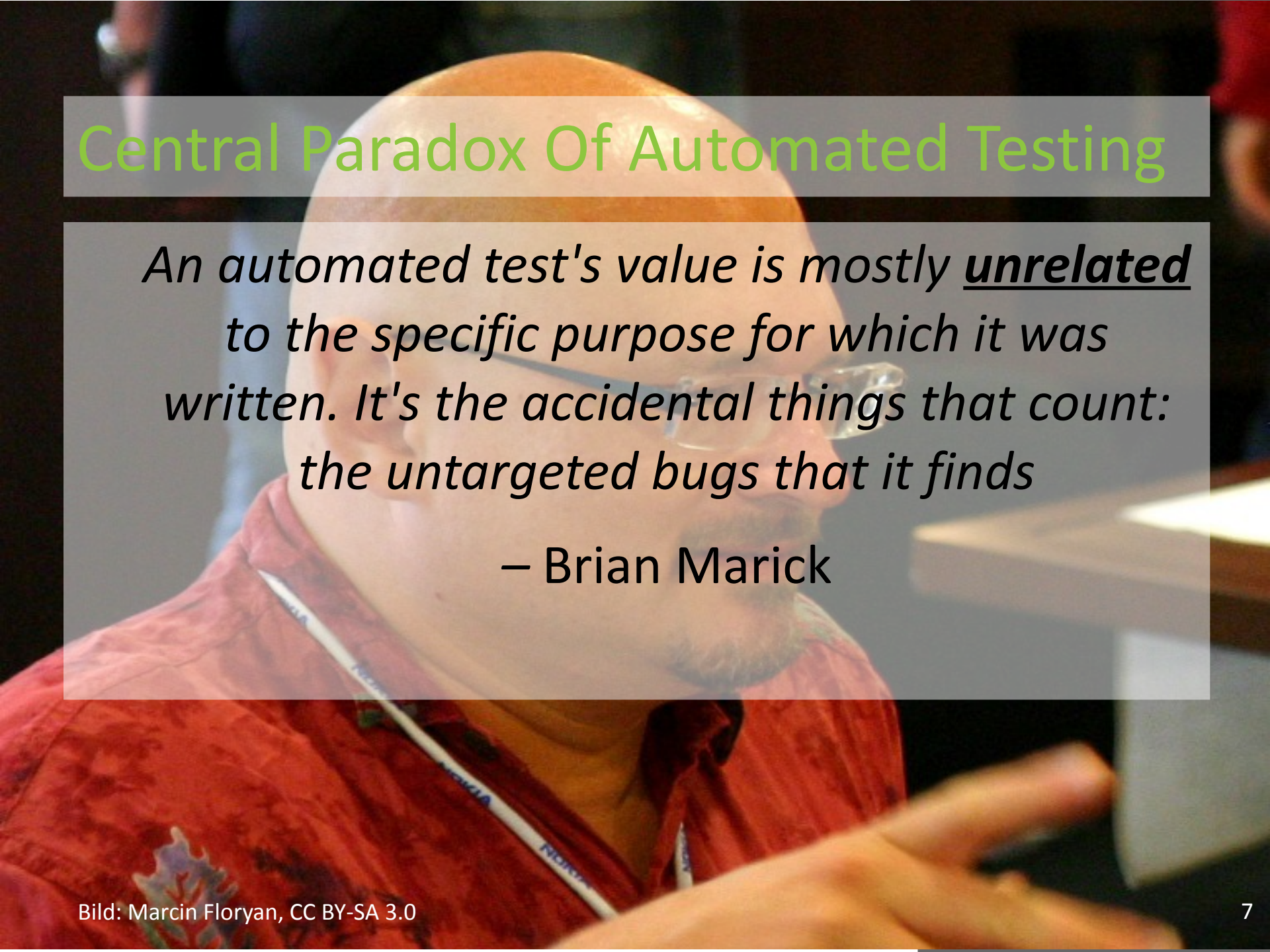
*(L * W) - (M * K)*

(zusätzlich gefundene Bugs) - (nicht gefundene Bugs)

- L = Lebenszeit (Anzahl Durchläufe bis Test veraltet)
- W = Wahrscheinlichkeit, neue Bugs zu finden (Erwartungswert für Bugs pro Durchlauf)
- M = Bugs/Zeit, die beim manuellen Testen gefunden werden können
- K = Kosten (Zeit für die Test-Implementierung)

Sind automatisierte Tests wirklich zu teuer?

Mit dieser Formel lässt es sich prüfen!



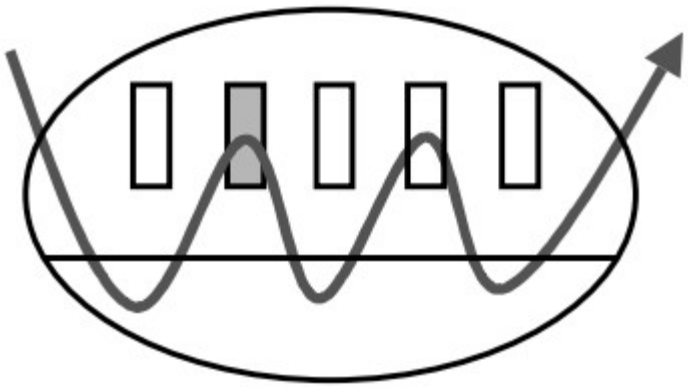
Central Paradox Of Automated Testing

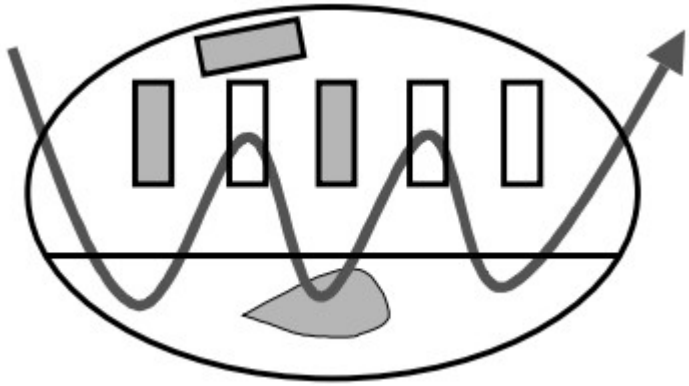
*An automated test's value is mostly **unrelated** to the specific purpose for which it was written. It's the accidental things that count: the untargeted bugs that it finds*

– Brian Marick

Test bei Feature-Änderung

- Test für Feature muss geändert werden, bevor er Fehler finden kann
- Integrationstest für anderes Feature findet Fehler durch Nebeneffekt
- Unit Test findet keine Fehler durch Nebeneffekt





Tests in Magento

Testbarkeit in Magento

- Magento 1 wurde ohne Tests entwickelt
- Testbarkeit war also nie Qualitätskriterium
- Isoliertes Testen von Modulen ist schwer
  - Zu viele Abhängigkeiten
- Integrationstests sind auch schwer
  - Zu viel Global State

EcomDev_PHPUnit To The Rescue!

- Hat Unit Testing in der Magento Welt salonfähig gemacht
  - Mocks für Models, Helpers, Blocks
  - Fixtures
  - Controller Tests mit Session Simulation
  - Tests für Layouts und Konfigurationen
- Naturgemäß komplex, fehleranfällig

Unit Testing?

- Unit Test für z.B. Block Rewrites oder Observer?
  - Ist das Model vollständig geladen?
  - Was steht in der Session?
  - Was wird aus der Registry geholt?
- Aufwändiges Mocking notwendig
- Bugs entstehen am ehesten, weil die Parameter nicht wie erwartet sind. Der Unit Test in seiner heilen Mock-Welt bekommt das nicht mit.
- Isoliertes Testen für die meisten Magento-Anpassungen sinnlos

Eine wichtige Erkenntnis...

EcomDev_PHPUnit

=

Integration Test Framework

Tools für Integrationstests

- Für Extensions:
  - EcomDev_PHPUnit
  - Testen gegen verschiedene Magento Versionen mit Aoe Mage Test Stand (auf Travis CI oder Jenkins)
- Für Shops:
  - xtest

TDD mit Magento

- Für komplexen Business-Code mit wenig Interaktion zu Magento
- PRO Tipp: Logik komplett von Magento entkoppeln und nur mit PHPUnit testen
  - Schnelle Testdurchläufe
  - Saubererer Code
  - Portierung zwischen Magento 1 und Magento 2 möglich

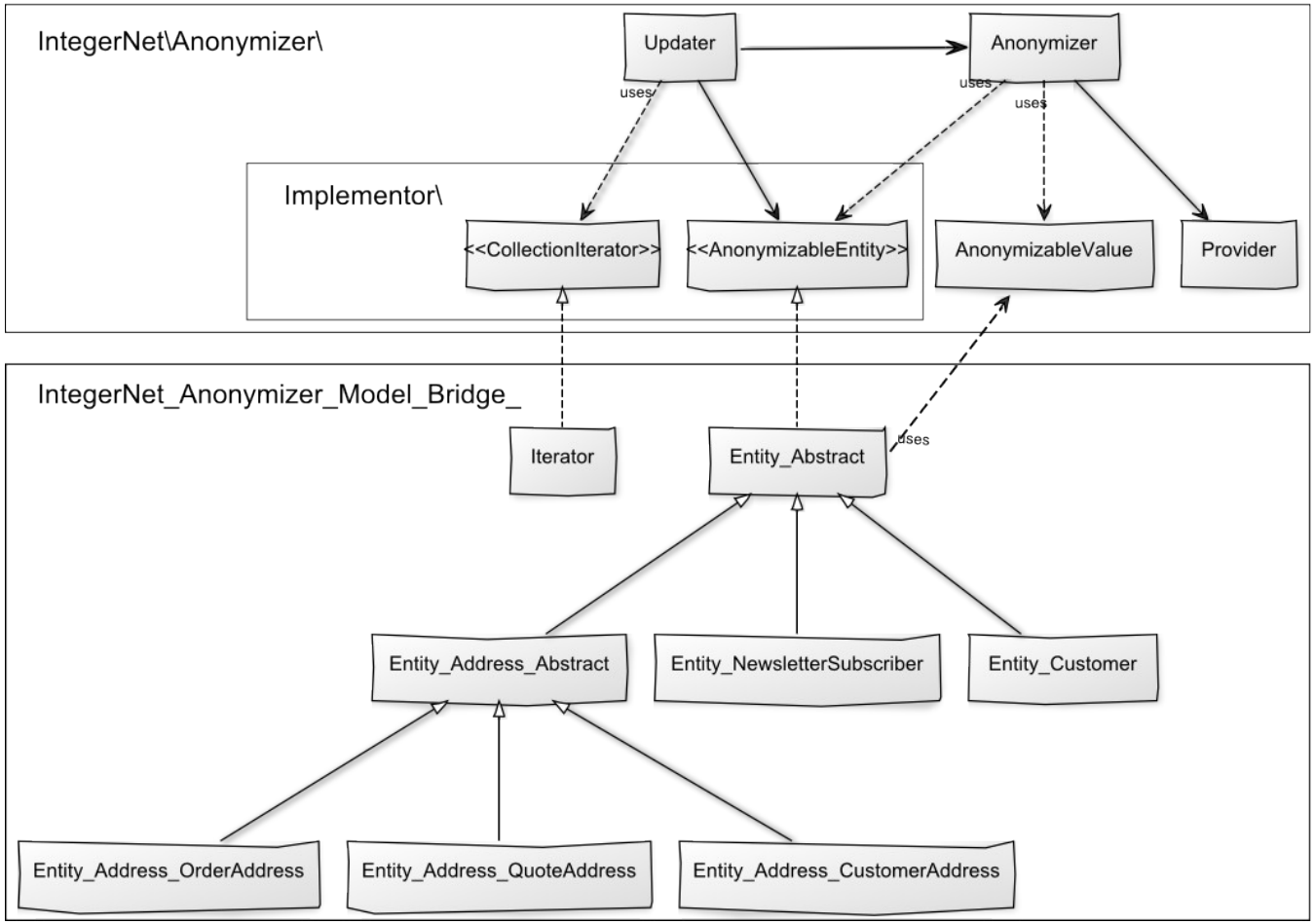
Beispiel: IntegerNet_Anonymizer

- Anonymisiert kundenbezogene Daten mit Dummy-Daten aus Faker
- Berücksichtigt Assoziationen

  order address == customer address
  => dummy order address == cummy customer address

- Logik als eigenständige Bibliothek (in lib/)
- Magento-Modul implementiert nur Interfaces zum lesen und schreiben von Daten und delegiert an die Bibliothek

Logik von Magento entkoppelt



Unit Tests für Bibliothek

- Testgetriebene Entwicklung mit PHPUnit
- Hier können Testdaten effizient variiert werden um Sonderfälle abzudecken
- Testlaufzeit 150 ms (keine Datenbankzugriffe, keine Magento-Instantiierung)

Integrationstests für Magento-Modul

- EcomDev_PHPUnit
- Bridge Test: Geben die Bridge Implementierungen die korrekten Attribut-Werte zurück und funktioniert das setzen neuer Werte?
- Anonymizer Test: Werden alle Daten tatsächlich verändert?
- Beschränkung auf wenige Durchläufe, keine ausführlichen Testdaten notwendig
- Testlaufzeit: 4 s

Lessons Learned

- Feature Code > Support Code

  => entkoppeln lohnt sich
- Bridge Pattern zur Anbindung an Magento
- Library 100% Test Driven, Modul teilweise

Weitere Beispiele

- https://github.com/magento-hackathon/DerivedAttributes
- https://github.com/EcomDev/EcomDev_LayoutCompiler

Systemtests

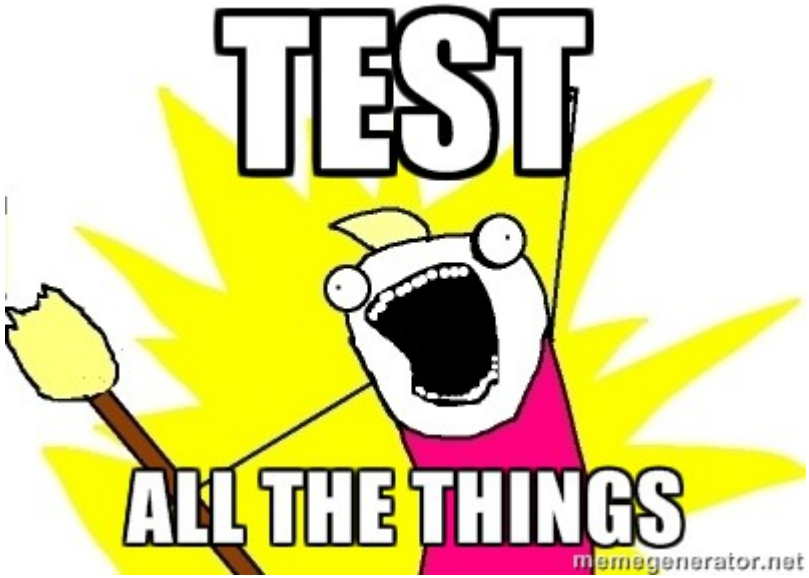
- Funktionale Tests
  - Automatisierung bei guten Integrationstests nicht zwingend notwendig
  - xtest verbindet beides (via PHPUnit_Selenium)
  - Wegwerf-Tests mit Selenium IDE (Firefox Plugin)
- Smoke Tests
  - Durchlaufen der Basis-Shop-Funktionalität, um gravierende Fehler zu entdecken

Typische Fehler

Fehler 1: Umgang mit fehlerhaften Tests

- Typischer Umgang mit degenerierter Test Suite:
  - **Phase 1**: Viele Stunden mit Reparieren von Tests verbringen
  - **Phase 2**: Bekannte False Positives ignorieren
  - **Phase 3**: Alle Failures ignorieren. Keine neuen Tests mehr schreiben, weil eh nicht mehr erkennbar ist, welche Tests zu Recht fehlschlagen und welche nicht.
- Defekte Tests lieber deaktivieren, als sie mitzuschleifen!
- Test-Strategie regelmäßig hinterfragen: Tun wir (noch) das Richtige? *Haben die Tests noch Wert?*

Fehler 2: Tests um der Tests willen



- Anzeichen für TDT (Test Driven Test):
  - Code Coverage um jeden Preis
  - Unit Tests ohne TDD

Fehler 3: Gar keine Tests

- Für Mini-Projekte kann manuelles Testen genügen
- Automatisierung von Smoke Tests immer sinnvoll, wenn die Infrastruktur vorhanden ist
  - Bedenke: *(L * W) - (M * K)*
- Keine Test-Automatisierung => viel Zeit für manuelles Testen einplanen
- Kein manuelles Testen => viel Bugfixing im Panik-Modus einplanen

Magento 2

- Eigenes Test Framework
- Core ist mit Unit Tests versehen, aber großteils
  - Portiert von Magento 1
  - Nicht testgetrieben entwickelt
- Abhängigkeiten sind durch DI expliziter, aber
  - Immer noch viel zu instabil
  - Immer noch gibt es schwarze Löcher wie die Registry

```
     * @SuppressWarnings(PHPMD.ExcessiveParameterList)
     */
    public function __construct(
        \Magento\Framework\Model\Context $context,
        \Magento\Framework\Registry $registry,
        \Magento\Framework\Api\ExtensionAttributesFactory $extensionFactory,
        AttributeValueFactory $customAttributeFactory,
        \Magento\Store\Model\StoreManagerInterface $storeManager,
        \Magento\Catalog\Api\ProductAttributeRepositoryInterface $metadataService,
        Product\Url $url,
        Product\Link $productLink,
        \Magento\Catalog\Model\Product\Configuration\Item\OptionFactory $itemOptionFactory,
        \Magento\CatalogInventory\Api\Data\StockItemInterfaceFactory $stockItemFactory,
        \Magento\Catalog\Model\Product\Option $catalogProductOption,
        \Magento\Catalog\Model\Product\Visibility $catalogProductVisibility,
        \Magento\Catalog\Model\Product\Attribute\Source\Status $catalogProductStatus,
        \Magento\Catalog\Model\Product\Media\Config $catalogProductMediaConfig,
        Product\Type $catalogProductType,
        \Magento\Framework\Module\Manager $moduleManager,
        \Magento\Catalog\Helper\Product $catalogProduct,
        Resource\Product $resource,
        Resource\Product\Collection $resourceCollection,
        \Magento\Framework\Data\CollectionFactory $collectionFactory,
        \Magento\Framework\Filesystem $filesystem,
        \Magento\Indexer\Model\IndexerRegistry $indexerRegistry,
        \Magento\Catalog\Model\Indexer\Product\Flat\Processor $productFlatIndexerProcessor,
        \Magento\Catalog\Model\Indexer\Product\Price\Processor $productPriceIndexerProcessor,
        \Magento\Catalog\Model\Indexer\Product\Eav\Processor $productEavIndexerProcessor,
        CategoryRepositoryInterface $categoryRepository,
        Product\Image\CacheFactory $imageCacheFactory,
        \Magento\Catalog\Model\ProductLink\Management $linkManagement,
        \Magento\Catalog\Api\Data\ProductLinkInterfaceFactory $productLinkFactory,
        \Magento\Catalog\Api\Data\ProductAttributeMediaGalleryEntryInterfaceFactory $mediaGalleryEntryFactory,
        \Magento\Framework\Api\DataObjectHelper $dataObjectHelper,
        array $data = []
    ) {
```

Gilt das alles noch für Magento 2

- Ja!
- Neue Technik für Integration Tests: Magento 2 Test Framework
- Business Logik vom Framework entkoppeln ist immer eine gute Idee

Kontakt

Web: www.schmengler-se.de /www.integer-net.de

Twitter: @fschmengler / @integer_net

Email: fs@integer-net.de

MM15DE: Speaker Corner

Magento Certified DEVELOPER

Magento Certified SOLUTION SPECIALIST

Links

- Brian Marick: When Should a Test Be Automated?
  http://www.exampler.com/testing-com/writings/automate.pdf
- EcomDev_PHPUnit: https://github.com/EcomDev/EcomDev_PHPUnit
- xtest: http://xtest-mage.com/
- Travis CI: https://travis-ci.org/ / https://travis-ci.com/
- Mage Test Stand: https://github.com/AOEpeople/MageTestStand
- IntegerNet_Anonymizer: https://github.com/integer-net/Anonymizer
- Humbug: https://github.com/padraic/humbug