Magento Entwickler Schulung

Simon Sprankel

http://www.coderblog.de

10. Juli 2011 – zuletzt aktualisiert am 08.04.2012

Inhaltsverzeichnis

Magento - was ist das eigentlich?

- DIE Open Source Shop Software
- wurde im Juni 2011 von ebay für 180 Millionen Dollar (ca. 126 Millionen Euro) gekauft
- 45 Angestellte 2007 $\rightarrow$ ca. 375+ Angestellte 2012
- mehr als 4000 Extensions
- mehr als 4 Millionen Downloads

Was bringt EUCH das eigentlich?

- Magento kennenlernen
- Magento (ansatzweise) verstehen
- eine extrem gute Software-Architektur kennenlernen
- Geld verdienen können (es gibt **viel** zu wenig Magento Entwickler!)

Wer in Magento einigermaßen fit ist, braucht sich über Nebenjobs keine Gedanken mehr zu machen . . .

Was bringt MIR das eigentlich?

Meine Freundin hat mich fast gesteinigt, als ich ihr gesagt habe, dass ich für die Schulung nichts bekomme, aber . . .

- Spaß :-)
- Erfahrung
- Arbeitsentlastung! Wetten, ich kann euch in Magento Projekte integrieren? :-)

Fragestunde

- Wer kann HTML?
- Wer kann CSS?
- Wer kann PHP?
- Wer kann Java? Okay, dann könnt ihr auch PHP :-)

Entwicklungsumgebung

Hat jeder die Magento-VM [1] installiert oder ein vergleichbares System aufgesetzt? Wir brauchen:

- Linux mit konfiguriertem Apache / PHP / MySql
- Magento Testinstallation inklusive Beispieldaten
- Netbeans (!), PhpStorm, Eclipse PDT, Aptana
- Firefox inklusive Firebug [2] (auch für die Magento-VM Leute!)

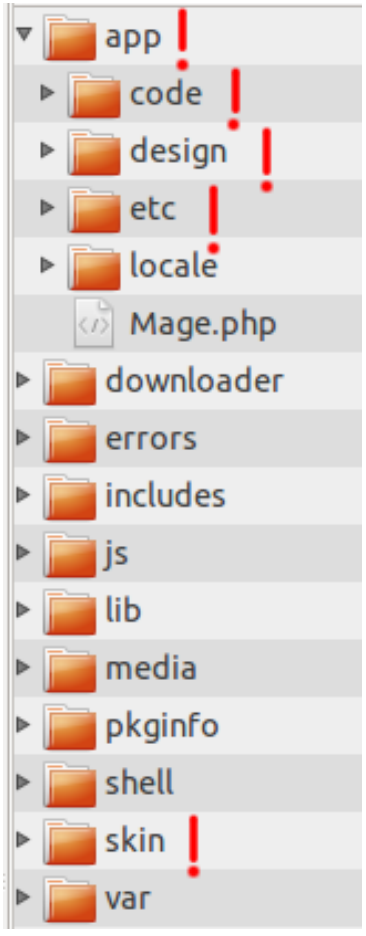
Wer hat schon mit Firebug gearbeitet?

---

[1] `http://www.magento-vm.de`

[2] `https://addons.mozilla.org/de/firefox/addon/firebug/`

Verzeichnisstruktur



Designs erstellen und bearbeiten

- Wo findet man was?
- Interfaces und Themes
- Theme Fallback
- Wie mache ich denn jetzt ein eigenes Theme?
- Learning by doing (!)

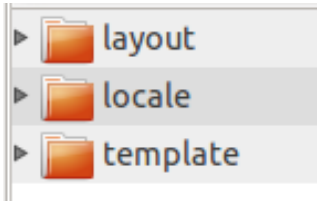
Designs erstellen und bearbeiten
Wo findet man was?

- öffentlich zugängliche Dateien (CSS- und JavaScript-Dateien, Bilder) in `skin/frontend`
- alle anderen Dateien in `app/design/frontend`

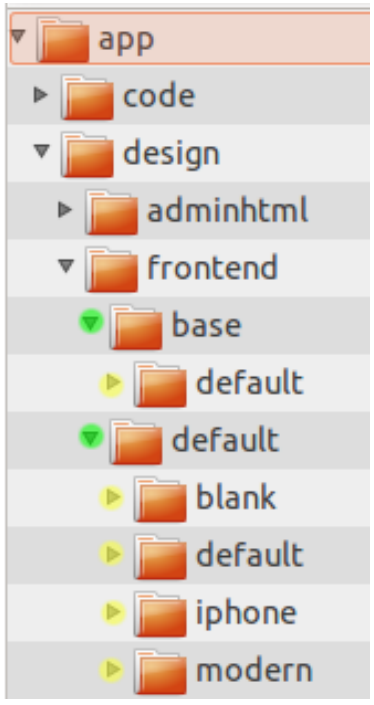
Was sind denn “alle anderen Dateien”?

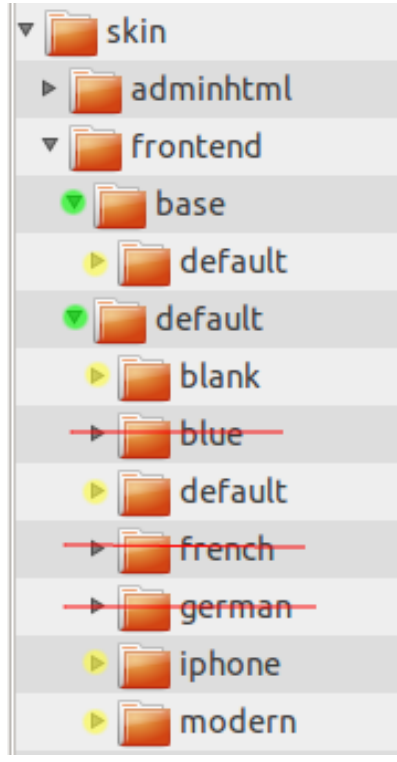
- XML-Dateien, die die Struktur der Seite angeben (im `layout` Ordner)
- *.phtml-Dateien, die eigentlichen Templates (im `template` Ordner)

Abbildung: Typischer Theme Ordner



Designs erstellen und bearbeiten
Interfaces (grün) und Themes (gelb)





- `base/default`: basic design
- `default/default`: Ordner für Designs von Extensions
- `default/MYTHEME`: hier legt man sein eigenes Theme an

Designs erstellen und bearbeiten
Theme Fallback

Nach Theme Dateien wird in der folgenden Reihenfolge gesucht (sowohl im `skin`, als auch im `app/design` Ordner):

1. `default/MYTHEME` (muss im Backend eingestellt werden)
2. `default/default`
3. `base/default`

Designs erstellen und bearbeiten
Wie mache ich denn jetzt ein eigenes Theme?

- NICHT durch kopieren des `base/default` Themes!
- Anlegen eines eigenen Themes innerhalb des `default` Interfaces
- für Template Änderungen: Kopieren einzelner Dateien aus dem `base/default` Theme in das eigene Theme und Bearbeiten dieser Dateien
- für Layout Änderungen: Anlegen und Bearbeiten der local.xml

Designs erstellen und bearbeiten
Layout Updates mit der local.xml

- Layout XML Dateien werden nach einander geladen und gemerged
- die local.xml Datei im layout Ordner wird als Letztes geladen
- damit lassen sich also fast alle Layout Updates bewerkstelligen!
- Grundstruktur wie alle anderen Layout XML Dateien:

```
<?xml version="1.0"?>
<layout version="0.1.0">
    <!-- Layout updates here -->
</layout>
```

Designs erstellen und bearbeiten
local.xml Beispiel: Blöcke verschieben

Verschieben der Suche aus dem Header in die rechte Spalte:

```
...
    <default>
        <reference name="header">
            <action method="unsetChild">
                <name>topSearch</name>
            </action>
        </reference>
        <reference name="right">
            <action method="insert">
                <name>top.search</name>
            </action>
        </reference>
    </default>
...
```

Designs erstellen und bearbeiten
local.xml Beispiel: Komplettes Entfernen von Blöcken

```
...
<checkout_cart_index>
    <remove name="checkout.cart.shipping" />
</checkout_cart_index>
...
```

Achtung: Durch ein `remove` werden die zugehörigen Klassen nicht mehr geladen und der Block kann auch nicht mittels `insert` wieder eingefügt werden. Man müsste ihn dann neu definieren.

Designs erstellen und bearbeiten
local.xml Beispiel: Entfernen und Einfügen von JavaScript

```
...
<default>
    <reference name="head">
        <action method="addItem">
            <type>skin_js</type>
            <name>js/my_js.js</name>
        </action>
        <action method="removeItem">
            <type>js</type>
            <name>some_ext/jquery-1.4.2.js</name>
        </action>
    </reference>
</default>
...
```

Designs erstellen und bearbeiten
Weitere local.xml Beispiele

- http://www.webguys.de/magento/turchen-11-templating-mittels-local-xml-in-der-praxis/
- http://inchoo.net/ecommerce/magento/using-local-xml-for-overriding-or-updating-xml-structu
- http://magebase.com/magento-tutorials/5-useful-tricks-for-your-magento-local-xml/
- http://classyllama.com/development/magento-development/the-better-way-to-modify-magento-layout/

Designs erstellen und bearbeiten
Tipps und Tricks

- unter System - Konfiguration auf Website-Ebene umschalten und unter Entwickleroptionen - Debug die Pfadhinweise einschalten
- mit Firebug z. B. einen Klassennamen suchen und in Dateien unter `app/design/frontend/default` danach suchen (z. B. mit `grep -R classname /var/www/shop/*`)

Designs erstellen und bearbeiten
Learning by doing

1. lege ein eigenes Theme an
2. entferne das PayPal Logo in der rechten Spalte
3. entferne den Back-to-school-Banner in der rechten Spalte
4. entferne den “Help Us to Keep Magento Healthy” Hinweis ganz unten
5. ändere die Hintergrundfarbe jedes zweiten Produkts in der Kategorieansicht in eine Farbe deiner Wahl
6. füge ein Block in der rechten Spalte (auf der Kategorie- und Produktseite) ganz oben ein, der einigermaßen schön anzeigt, dass der Shop ab einem Bestellwert von 100 Euro versandkostenfrei liefert (Blocktyp: `core/template`)

Module schreiben

- Was ist ein Modul?
- Codepools
- Von der URL zum Controller
- Woraus besteht ein Modul?
- Module anpassen
- Eigene Elemente schreiben
- ...

Module schreiben
Was ist ein Modul?

- Sammlung von Funktionalität mit dem gleichen Thema, z. B.
  - `Mage_Catalog`
  - `Mage_Checkout`
  - `Mage_Cms`
  - `Mage_Sales`
  - `Spranks_Easylog`
  - `...`

Module schreiben
Codepools

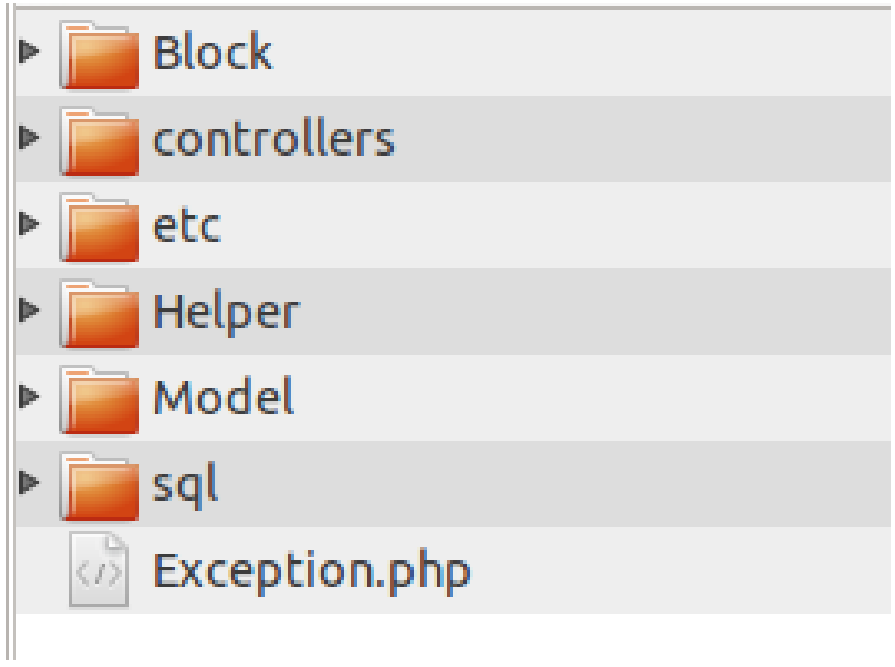
Wie wird nach einem Modul / einer Datei gesucht?
Ähnlich zum Theme Fallback ...

1. `app/code/local`
2. `app/code/community`
3. `app/code/core`
4. `lib`

Module schreiben
Von der URL zum Controller

- frontname/actionname/method wird gemapped auf `Company_Modul_ActionnameController#methodAction()`, die sich in `app/code/[codepool]/Company/Modul/controllers/ActionnameController` befindet
- der `[codepool]` wird dynamisch durch die Codepool-Hierarchie bestimmt, `Company` und `Modul` sind bekannt aus XML-Konfiguration
- Parameter können angehängt werden mit /key1/value1/key2/value2
- catalog/product/view/id/6720?
- `Mage_Catalog_ProductController#viewAction()` mit Parameter id = 6720

Module schreiben
Woraus besteht ein Modul? 1/5



Module schreiben
Woraus besteht ein Modul? 2/5

- app/etc/modules/Company_Module.xml meldet Modul in System an [3]

```
<?xml version="1.0"?>
<config>
  <modules>
    <Company_Module>
      <active>true</active>
      <codePool>local</codePool>
      <depends>
        <Company2_Module/>
      </depends>
   </Company_Module>
  </modules>
</config>
```

[3] kann man unter System - Konfiguration - Erweitert - Modulausgaben deaktivieren prüfen!

Module schreiben
Woraus besteht ein Modul? 3/5

- Blocks: Block-Klassen, die in den layout XML-Files verwendet werden können - für Anzeige verantwortlich
- Controller: Klassen, die auf Aufrufe bestimmter URLs reagieren
- XML Files (Verzeichnis `etc`): Definieren das Modul (`config.xml`) und ggf. Einstellungen im Admin-Bereich (`system.xml`) oder Berechtigungen im Admin-Bereich (`adminhtml.xml`)

Module schreiben
Woraus besteht ein Modul? 4/5
Eine minimale Company/Module/etc/config.xml

Das wichtigste File eines Moduls! Hier wird **ALLES** definiert. XML Code auf den folgenden Folien wird immer hier eingefügt.
Magento ist ein konfigurationsbasiertes MVC System!

```
<?xml version="1.0"?>
<config>
  <modules>
    <Company_Module>
      <version>0.1.0</version>
    </Company_Module>
  </modules>
</config>
```

Module schreiben
Woraus besteht ein Modul? 5/5

- Helper: Weiß eigentlich auch keiner so Recht... Passt nirgendwo hin? Hier rein. Wenn Magento meckert, legt man halt einen an.
- Models: Klassen, die die eigentliche Arbeit verrichten und auf Daten zugreifen und sie verändern
- `sql` Verzeichnis: Skripte, um Datenbank anpassen zu können
- ggf. Layout Dateien in `app/design/frontend/default/default` bzw. `app/design/adminhtml/default/default`

Module schreiben
Module anpassen

Wie kann ich ein bestehendes Modul verändern?

- Schlechteste Lösung: Kopieren des Moduls / der entsprechenden Datei in höheren Codepool
- Bessere Lösung: Models/Blocks/Helper/Controller umschreiben
- Beste Lösung: Event-Observer-Pattern anwenden (nicht immer möglich)

Module anpassen
Kopieren von Modulen / Dateien

- schlechteste Lösung
- Kopieren von z. B. `app/code/core/Mage/Catalog/Product.php` nach `app/code/local/Mage/Catalog/Product.php`
- Konsequenz: `app/code/core/Mage/Catalog/Product.php` wird nicht mehr benutzt
- wenn diese Datei geupdated wird, gibt es Probleme ...
- schwer wartbar, da keiner durchblickt, was eigentlich angepasst wurde

Module anpassen
Models/Blocks/Helper/Controller umschreiben

- bessere Lösung
- Überschreiben der gesamten Klasse und Anpassen einzelner Methoden durch Überschreiben

- Codeausschnitte der nächsten Folien sind aus dem Developer Cheatsheet vom “Extension Papst” Vinai Kopp
- dass man den Code nicht kopieren kann, ist so gewollt - sowas muss man ein paar Mal selbst geschrieben haben :-)

Module anpassen
Rewrite von Models / Blocks / Helpern

Überschreiben der gesamten Klasse und Anpassen einzelner Methoden durch Überschreiben

```
<config>
  <global>
    <models> [1]
      <sales> [2]
        <rewrite>
          <order_item>Namespace_Module_Model_Sales_Order_Item</order_item> [3]
        </rewrite>
      </sales>
    </models>
  </global>
</config>
```

[1] Depending on object type „models“, „helpers“ or „blocks“

[2] Modul shorthand

[3] The tag is the class to override, the content is the full name of the replacement class.

Module anpassen
Rewrite von Controllern

```
<config>
  <frontend> [1]
    <routers>
      <checkout> [2]
        <args>
          <modules>
            <yourModule
              before="Mage_Checkout">Namespace_Module_Overwrite_Checkout</yourModule> [3]
          </modules>
        </args>
      </checkout>
    </routers>
  </frontend>
</config>
```

[1] Depending on the controllers area „frontend“ or „admin“

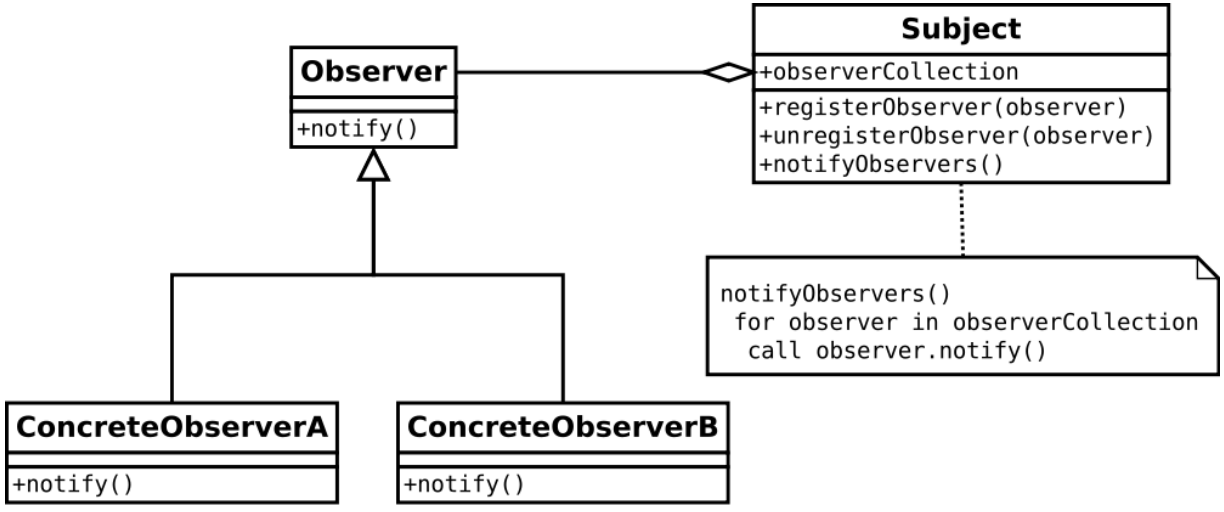
[2] Router node of the module to be overridden (look it up in the config.xml of that module)

[3] The tag <yourModule> can be anything, it must be unique within the <modules> node.
Your_Module_Overwrite_Checkout (is mapped to directory)
Your/Module/controllers/Overwrite/Checkout/

Module anpassen
Event-Observer 1/2

Wer kennt das Event-Observer-Pattern?

- **Events**: werden ausgelöst, wenn irgend etwas passiert, z. B. `checkout_cart_product_add_after`
- **Observer**: werden benachrichtigt, wenn ein Event eintritt
- Klassen können sich als Observer für bestimmte Events eintragen



Module anpassen
Event-Observer 2/2

```
<config>
  <frontend> [1]
    <events>
      <catalog_product_load_after> [2]
        <observers>
          <yourModule> [3]
            <type>singleton</type> [4]
            <class>module/observer</class> [5]
            <method>catalogProductLoadAfter</method> [6]
          </yourModule>
        </observers>
      </catalog_product_load_after>
    </events>
  </frontend>
</config>
```

[1] Area: „adminhtml“, „frontend“ oder „global“.
[2] Event Code
[3] Must be unique within the event observer.
[4] „model“ or „singleton“
[5] Your_Class_Name oder module/class
[6] The method to be called on the observer instance.

Module schreiben
Eigene Elemente schreiben

Wie schreibe ich ein eigenes Modul?

- `app/etc/modules/Company_Module.xml` anlegen
- `app/code/[codepool]/Company/Module/etc/config.xml` anlegen
- Beliebige Funktionalität implementieren (in `config.xml` definieren!)

Module schreiben
Model / Block / Helper definieren

```
<config>
  <global>
    <models> [1]
      <yourModule> [2]
        <class>Namespace_Module_Model</class> [3]
      </yourModule>
    </models>
  </global>
</config>
```

[1] Depending on object type „models“, „helpers“ or „blocks“

[2] The module shorthand, must be unique for the module

[3] Classname prefix, will replace the shorthand [2] when resolving

Module schreiben
Controller definieren

```
<config>
  <frontend> [1]
    <routers>
      <yourModule> [2]
        <use>standard</use> [3]
        <args>
          <module>Namespace_Module</module> [4]
          <frontName>yourmod</frontName> [5]
        </args>
      </yourModule>
    </routers>
  </frontend>
</config>
```

[1] Depending on the controllers area „frontend“ or „admin“
[2] Must be unique
[3] „admin“ for Adminhtml controllers, otherwise „standard“
[4] Namespace and Name of the Module as defined in the <config><modules> node
[5] The URL part to point to the controller directory (e.g. http://example.com/yourmod)
Magento will look in Namespace/Module/controllers/ for the called controller.

Module schreiben

Problem: Zwei Module überschreiben die selbe Klasse

Beispiel: Die Klasse `MyCompany_Module_Model_Class` und die Klasse `OtherCompany_Module_Model_Class` überschreiben beide die Klasse `Mage_Module_Model_Class`.

1. Lösung: `<depends><OtherCompany_Module/></depends>` in app/etc/modules/MyCompany_Module.xml einfügen
2. Lösung:
   - 2.1 Entfernen des Klassen Rewrites in `MyCompany/Module/etc/config.xml`
   - 2.2 Klassen Definition von `MyCompany_Module_Model_Class` ändern
     alt: `MyCompany_Module_Model_Class extends Mage_Module_Model_Class`
     neu: `MyCompany_Module_Model_Class extends OtherCompany_Module_Model_Class`

Hier ist das Modul MDN_ExtensionConflict äußerst hilfreich...

Module schreiben
Tipps und Tricks

- Model instanziieren:
  `Mage::getModel('catalog/product');`
- @var Statement für Auto-Vervollständigung:
  `/* @var $product Mage_Catalog_Product */`
- Logeintrag in einer Datei in `var/log` erzeugen:
  `Mage::log('Logentry', null, 'logfile.log');`

Module schreiben
Learning by doing

- Selber ein Modul schreiben, welches bei Aufruf der URL http://wwww.shop.de/schulung/special/machwas einen Logeintrag erstellt
- Gemeinsam Modul für Newsletter Subscription schreiben: User kann irgendwo auf Seite Daten eingeben für Newsletter, Adresse wird an Shopbetreiber gemailt
- Gemeinsam Modul schreiben: Neuer Tab im Admin-Bereich "Schulung", in dem irgendwas angezeigt wird
- Bestehende Module zusammen anschauen

Viel Erfolg ...

Magento Resourcen 1/2

- Magento Commerce http://www.magentocommerce.com
  - Forum http://www.magentocommerce.com/boards
  - Knowledge Base
    http://www.magentocommerce.com/knowledge-base
  - Wiki http://www.magentocommerce.com/wiki
  - PHP Docs http://docs.magentocommerce.com/
  - Answers
    http://www.magentocommerce.com/answers/get-started
  - IRC Chat: auf chat.freenode.net #magento-de
- Magento Design Guide http://info.magento.com/rs/magentocommerce/images/MagentoDesignGuide.pdf
- Modulentwicklung (Magento for PHP MVC Developers)
  http://alanstorm.com/category/magento /
  http://t3n.de/magazin/workshop-module-open-source-shop-system-programmieren-

Magento Resourcen 2/2

- Magento Cheatsheets
  `http://devcheatsheet.com/tag/magento` (hier vor allem der “Magento Developer Cheatsheet” vom “Extension-Papst” Vinai Kopp :-))
- Magento Podcast `http://magentopodcast.de`
- Magento Expert `http://magentoexpert.com`
- Stackoverflow
  `http://stackoverflow.com/questions/tagged/magento`
- `http://www.coderblog.de` :-)
- Tausende Magento Blogs

Fragen?

Vielen Dank für eure Aufmerksamkeit!