Dr. Nikolai Krambrock
Andreas von Studnitz

Codequalität mit Magento

Inhalt

- Vorstellung
- Motivation
- Code-Qualität Allgemein
- Code-Qualität Magento-spezifisch
- Fragen und Antworten

Magento Certified DEVELOPER

Magento Certified DEVELOPER

Dr. Nikolai Krambrock
Diplom-Informatiker
Magento seit 4 Jahren
code4business GmbH
Aachen

Andreas von Studnitz
Diplom-Informatiker
Magento seit 6 Jahren
integer_net GmbH
Aachen

Wofür Codequalität?

Für Kunden

- Funktionalität
- Gebrauchstauglichkeit
- Sicherheit
- Verfügbarkeit und Zuverlässigkeit

Für Entwickler

- Code einfach zu lesen
- Einfach zu verstehen
- Einfach anzupassen
- Einfach zu erweitern

Code-Qualität

ALLGEMEIN

Coding Conventions (1)

- Dead Code und nicht gelesene Variablen vermeiden; mglw. statische Analyse
- Doppelten Quellcode vermeiden – Code Smell #1 nach Martin Fowler
- Einfache Lösungen
- Indentation > 3 sollte überarbeitet werden
- Magic numbers: Woher kommt die 7?

Coding Conventions (2)

- Mittlere Größe: Klassen 100-200 Lines of Code (LoC), Methoden 3-20 LoC
- Sprechende Variablennamen:
  Sehr schlecht: $c = Mage::getModel("...");
  Schlecht: $customer = getModel("...");
  Gut: $customerToAdd = getModel("...");
- Sprechende Methoden und Klassenamen:
  „deepClone()“ besser als „copy()“

Objektorientierung

- Zusammengehörige Attribute (=Eigenschaften, Status) und Methoden in Klassen kapseln
- Minimale Sichtbarkeit;
  in PHP meist protected

Werkzeuge

- Es gibt **keinen Grund ohne Versionsmanagement** zu arbeiten
- Single-Branch wenn alle in einem Raum, Multi-Branch bei verteilter Entwicklung
- Automatisierte Tests einsetzen
- Black-Box (Selenium) einfach und effizient aber lange Laufzeit, White-Box (PHPUnit) in großen Projekten hilfreich

Code-Qualität

MAGENTO-SPEZIFISCH

Kapselung von Modulen

- Module sollten eigenständig lauffähig sein
- Wenig Abhängigkeiten von anderen Modulen
- Keine Abhängigkeiten vom Template
- Layout-Updates (base/default) statt Anpassung von Template-Dateien
- Ein Modul, das Template-spezifische Zusatzfunktionen abdeckt

Updatefähigkeit (1)

- Keine Core-Hacks
- Keine Hacks von externen Modulen
- ➡ Stattdessen: Rewrites, besser Observer
- Keine Änderungen von Sprachdateien
- ➡ Stattdessen: Eigene Übersetzungsdateien

Updatefähigkeit (2)

- Korrekte Verwendung von Code Pools
  - **local**: projektspezifische Module
  - **community**: veröffentlichte / geteilte Module
  - **core**: Core
  - **lib**: Externe Bibliotheken
- Kopien von Template-Dateien minimieren
  - local.xml
  - Fallback nutzen

Minimale Änderungen

- Minimale Anzahl LoC für Anforderung – „nur **kein** Code ist guter Code“
- Bestehende Funktionen nutzen
- Beispiel: Preisregeln generieren statt Preislogik überschreiben
- Beispiel: Tier Prices for Custom Options durch generieren weiterer Custom Options
- Beispiel: Free Product durch Preisregel

Model View Controller (MVC)

- Code und Template strikt trennen
  - Kein (SQL-)Code im Template!
  - Kein HTML im Code!
- MVC nach Möglichkeit einhalten
  - Anreicherung von Daten in Models
  - Logik in Controllern und Observern
  - Hilfsfunktionen für das Template in Blocks

Setup-Skripte

- Setup-Skripte statt direkter Änderungen an der Datenbank oder im Admin-Bereich
  - Datenbank-Tabellen und -Spalten
  - Attribute
  - Konfiguration
  - Inhalte (CMS, Kategorien, Produkte, …)
- ➡ Zuverlässigeres Deployment

Fremdmodule

- Nur Module von bekannt zuverlässigen Quellen verwenden
- Alternativ gründliche Code-Analyse
- Module mit verschlüsseltem Code meiden

Coding-Guidelines

- Zend Framework Coding Standard
  - http://framework.zend.com/manual/1.12/de/coding-standard.html
- Magento Extension Developers Guide
  - http://info.magento.com/rs/magentocommerce/images/Magento-Extension-Developers-Guide-v1.0.pdf
- Am Magento-Kern orientieren

Fragen?

Dr. Nikolai Krambrock
code4business GmbH

krambrock@code4business.de
Xing: Nikolai_Krambrock
Twitter: @nkrambrock

Andreas von Studnitz
integer_net GmbH

avs@integer-net.de
Xing: Andreas_vonStudnitz2
Twitter: @avstudnitz