# Praktikumsbericht

Michael Sandritter
AOE GmbH

24. August 2015

## Praktikumsbetrieb

Mein Praktikum habe ich bei der AOE GmbH in Wiesbaden absolviert. Die AOE GmbH hat seinen Hauptsitz in Wiesbaden und ist Dienstleister für Open Source Enterprise Lösungen. AOE wurde 1999 unter dem Firmennamen AOE media gegründet und arbeitete anfänglich als TYPO3 Dienstleister. Inzwischen umfassen die Serviceleistungen Open Source Web Portale, E-Commerce und mobile Anwendungen, die für globale agierende Unternehmen entwickelt werden. Das Kollegium umfasst 180 Entwickler und Consultants an 8 Standorten, davon 120 Mitarbeiter in der Zentrale in Wiesbaden. Die Unternehmenskultur setzt auf die Kernelemente: Open Source, objektorientierte Programmierung und Methoden wie Agile Software-Entwicklung und Test-Driven-Development.

Dabei pflegt die AOE GmbH eine dezentrale Unternehmensstruktur. Jedem Kunden wird ein Team aus Entwicklern zur Seite gestellt. Das Team arbeitet von dort ab komplett selbstständig und setzt alle Projektphasen in engem Kontakt mit dem Kunden um.

## Arbeitsumfeld

In meinem Praktikum wurde ich vorwiegend als Backend-Entwickler im Congstar-Team eingesetzt. Das Team bestand dabei aus 19 Entwicklern und acht Testern, die sich ihrerseits wieder in 3 kleinere Teams aufteilen. Die Frontend bzw. Backend-Kompetenzen waren dabei gleichmäßig auf die verschiedenen Teams verteilt, so dass jedes Team, für sich, in der Lage war sowohl Backend- als auch Frontend-Tasks zu übernehmen.

Jedes Team arbeitet nach der agilen Softwareentwicklungsmethodik Scrum. In täglichen Daily Meetings tauschen sich die Entwickler in ihren kleinen Teams über den aktuellen Entwicklungsstand aus, indem jeder Entwickler und Tester, dem Team erzählt, woran er gerade arbeitet und gegebenenfalls schildert welche Probleme bei der Umsetzung existieren. Zusätzlich finden zwei mal wöchentlich Weekly Meetings statt, bei denen alle Entwickler und Tester zu einem übergreifenden Update des Entwicklungsstand zusammenkommen. In der großen Runde werden Themen besprochen, die das komplette Team betreffen. Dazu gehört zum Beispiel, das Einführen und Verwenden neuer Technologien, das Entwickeln oder ändern der Software Architektur und generelle organisatorische Themen.

Der zeitliche Rahmen gibt vor, dass dem Kunden alle zwei Monate ein neues Softwarepaket mit neuen Features ausgeliefert wird. Innerhalb dieser Zeit absolviert, jedes Team drei Sprints. Die ersten beiden Sprints sind Feature Sprints, die jeweils 3 Wochen Arbeitszeit umfassen. In diesen beiden Sprints werden neue Stories umgesetzt, die davor von den Product Ownern (PO) priorisiert worden sind. Das Team entscheidet jedoch für jeden Sprint neu, wie viele Stories realistisch umsetzbar sind und arbeitet zu jeder Story kleinere Tasks aus, die von den Entwicklern geclaimed und umgesetzt werden. Der letzte Sprint innerhalb der zwei Monate ist ein Release Sprint. Dieser erstreckt sich über die beiden verbleibenden

Wochen. Zu dieser Zeit finden Verbundtests statt, bei denen die neu entwickelten Features im Zusammenspiel mit der $Aax^2$ getestet wird.

Die $Aax^2$ ist die Software aus dem Hause Compax. Diese hält den congstar-spezifischen Produktkatalog. Darin werden alle Congstar Produkte abgebildet, wie zum Beispiel die Post- und Prepaidtarife, mit den dazugehörigen Buchungsoptionen. Die $Aax^2$ bietet dafür Soap-Schnittstellen an, um die Produktdaten im, von AOE entwickelten, Backend beziehen zu können.

Infolge des Release Sprints, besteht für die Entwickler ein Entwicklungs-Stop, so dass das neue Softwarepaket im Verbundtest getestet werden kann. Anstatt an neuen Features weiter zu arbeiten, wird die Zeit einerseits zum refactorn der Software und zum anderen zum update von Frameworks und anderer Software genutzt.

Am Ende jedes Sprints findet zusammen mit den Stakeholdern das Review statt, bei dem das Entwickler-Team die Ergebnisse des aktuell endenden Sprints präsentieren. Dafür fahren die Entwickler-Teams nach jedem Feature-Sprint nach Köln zu Congstar, während sich die Beteiligten nach Ende des Release-Sprints bei Aoe in Wiesbaden treffen. Hauptsächlich geht es um die Präsentation der Stories, die innerhalb des Sprints erfolgreich umgesetzt werden konnten. Das Entwickler-Team berichtet in diesem Zusammenhang was während des Sprints gut gelaufen ist, welche technischen Probleme und Hindernisse sich ergeben haben und wie diese Probleme gelöst wurden. Die Stakeholder haben in diesem Zuge die Möglichkeit offene Fragen mit den Entwicklern direkt zu klären. Alle Beteiligten erhalten somit einen Überblick über den aktuellen Stand des Projekts, der sich im Produkt-Backlog widerspiegelt. Das Review dient somit als Grundlage für das darauf folgende Sprint Planning.

Im Sprint Planning wird im ersten Teil besprochen, welche Stories und wie viele Stories im Laufe des nächsten Sprints umgesetzt werden können. Während im zweiten Teil besprochen wird, wie diese Stories umgesetzt werden sollen. Dies geschieht im Zusammenspiel von Entwickler-Team, PO und Scrum Master. Gemeinsam priorisieren sie das Sprintziel. Dafür werden die Stories im Backlog priorisiert und für jede Story eine Aufwandsschätzung durch die Entwickler abgegeben. Parallel wird grob berechnet wie groß die Velocity des Teams ist, um einordnen zu können wie viele Stories innerhalb eines Sprints tatsächlich umgesetzt werden können. Nachdem die Stories für den nächsten Sprint festgelegt sind, definieren die Entwickler für jede Story kleinere Tasks.

Backlog, Stories und Tasks werden im Projekt-Management-Tool zugänglich gemacht. Aktuell wird als solches Tool Kunagi verwendet. Künftig soll Kunagi jedoch durch JIRA ersetzt werden.

Im Laufe des Sprints werden die einzelnen Tasks von den Entwicklern abgearbeitet. Diese müssen dabei vermerken, wie viel Zeit sie für den Task geburned, sprich benötigt haben. Aus der geschätzen und der letztendlich benötigten Zeit lässt sich ein Burndown-Chart ermitteln, das den aktuellen Fortschritt des Sprints widerspiegelt.

Sind alle Tasks einer Story umgesetzt steht die Abnahme dieser Story an. Diese erfolgt durch einen Entwickler der an der Umsetzung der Story beteiligt war und durch den Product Owner. Der Entwickler präsentiert dem PO die Features, die durch die Story umgesetzt werden sollten. Erfüllt das Umgesetzte alle Akzeptanz-Kriterien, die für die Story definiert wurden, kann die Story durch den PO abgenommen werden und ist fertig.

## Projektbeschreibung

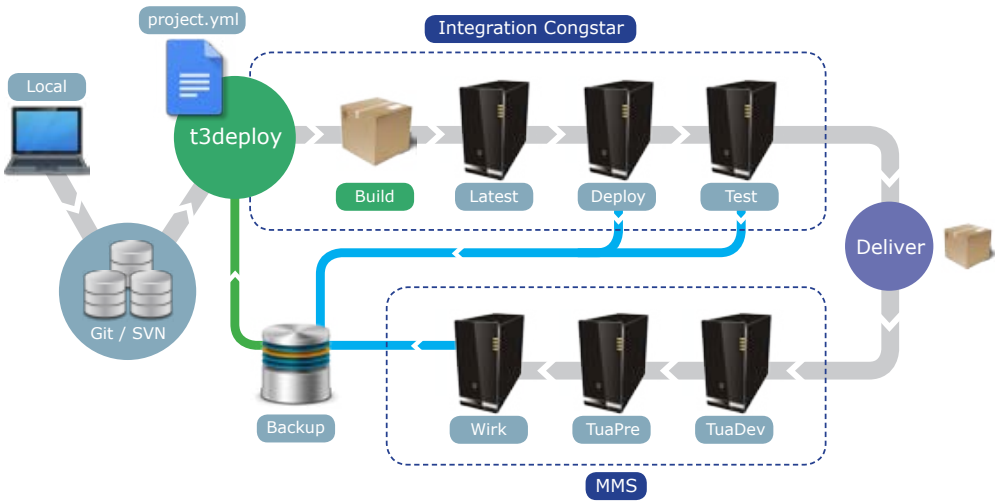
Entwickelt wird der Webshop der Firma Congstar. Congstar ist als Schwestergesellschaft der Telekom, deutschlandweiter Anbieter für Mobilfunk- und DSL-Produkte.

Dem Kunden soll zum einen abhängig von seinem Endgerät alle relevanten Informationen zum Congstar Portfolio und Service dargestellt bekommen und unabhängig von seinem Endgerät, neue Verträge abschließen und bestehende Verträge verwalten können.

Für die Redakteure von Congstar sollen spezielle Informationen des Webshops redaktionell und zentral pflegbar sein, um inhaltliche Inkonsistenzen und redundante Informationen zu vermeiden.

## Deployment Pipeline

Im folgenden wird der Ablauf der Deployment-Pipeline erläutert.




Der grobe Ablauf des Deployments sieht so aus, dass ein aktuelles Backup aller System, auf denen relevante Daten gepflegt werden müssen, erstellt und in das Backupstorage geschrieben werden. Dabei werden die Systeme vor einem Deployment gesperrt um zu verhindern, dass die Daten nach dem Backup noch verändert werden.

Die verschiedenen Systeme verteilen sich auf zwei Hosts:

- Integration Congstar
- MMS (T-Systems Multimedia Solution)

Wird ein Softwarepaket komplett neu gebaut, wird zum einen der Quellcode aus den getagten Git-Repositories gezogen und zum anderen die Datenbank und andere Bewegungsdateien aus dem Backupstorage (jeweils von den relevanten Systemen) zusammengeführt.

Die Verheiratung findet mit der Deploymentsoftware T3deploy statt. Mit Hilfe der Konfigurationsdatei project.yml baut die T3Deploy Software ein fertiges, installierbares Softwarepaket. Die Konfigurationsdatei enthält dabei alle Infos über Extensions, Libraries und Datenbanken, die zum Bau eines Pakets notwendig sind.

War der Bau eines Pakets erfolgreich, wird dieses zur erst auf den aoe-internen Umgebungen des Integration Congstar installiert und durchläuft mehrere Teststufen.

Auf der Latest Umgebung werden alle automatisierten Tests durchgeführt. Dazu gehören Unit-Tests, Smoke Tests, sämtliche Code Metriken und Static Cache Tests. Laufen dort alle Tests durch kann das Paket auf der Deploy Umgebung installiert werden. Dort haben die Redakteure von Congstar die Möglichkeit redaktionelle Änderungen vorzunehmen. Zusätzlich werden auf dieser Umgebung neue Nutzer für das Typo3-Backend angelegt, die über das Backupstorage in jedem gebauten Paket zugänglich gemacht werden. Die letzte Umgebung des Integration Congstar bildet die Test Umgebung. Dort laufen hauptsächlich Java Sellenium Tests.

Sind alle drei aoe-internen Teststufen durchlaufen wird einmal täglich nachts ein fertiges Paket an die MMS geliefert und dort auf der TuADev (Test und Abnahme) Umgebung installiert. Auf der TuADev Umgebung werden Abnahmetests seitens Congstar Testern durchgeführt, die hauptsächlich Sellenium Tests umfassen.

Sobald das Paket alle Teststufen durchlaufen hat, wird es letztendlich auf der Wirk-Umgebung installiert. Die Wirk Umgebung ist das Live-System, von der aus die Webseite für alle Besucher erreichbar ist.

## Task 01: Extension zur Reisekostenkalkulation

*TODO* Da viele Softwaremodule des Congstar Webshops also Typo3 Extension umgesetzt sind, habe ich im Zuge meiner Einarbeitung eine eigene Extension entwickelt, die als Reiseplanungstool eingesetzt werden kann. Ziel der Extension ist es, Personen eine grobe Kalkulation der Kosten offenzulegen, über Reiseziele, die sie gerne bereisen würden, von denen sie aber nicht wissen, welche Kosten auf sie zukommen. Informationen über Reiseziele und mögliche Kosten sollen über entsprechende APIs angesprochen werden und wurden bei der

Umsetzung dieser Extension vernachlässigt. Der Schwerpunkt der Umsetzung lag in der Erstellung und Konfiguration der Extension, sodass Sie über das Typo3 Backend eingebunden werden kann und mit zugewiesener URL im Browser aufrufbar ist.

# Task 02: Rollen-basierte Zuordnung der Bestandskundenverwaltung (CD 6929)

Ziel war es, dass die Bestandskundenfunktionalität rollen-basiert zugeordnet werden kann, damit sie nur den Sales-Partnern zur Verfügung steht, die die Funktion auch nutzen dürfen.

Dafür wurde im Typo3 Backend eine neue Gruppe für die Bestandskunden-Buchung angelegt. Redakteure können dadurch die Zuordnung der Rechte im Typo3 für verschiedene Partner freischalten oder zusätzlich zuweisen.

Zusätzlich wurde die Funktionalität des Sales-Partner-Objektes erweitert, sodass dieser Auskunft geben kann, welche Gruppe er inne hat. Dadurch wird für den jeweiligen Sales-Partner im Congstar Vertriebsportal, nur die Funktionalität zur Verfügung gestellt, die ihm zugeordnet ist.

# Task 03: Madeira POS Kartenaktivierung (CD 7227)

*TODO* - Flexform erweitern - Template erweitern

# Task 04: Webservice implementieren (CD 7126)

*TODO* Infos aus dem JIRA extrahieren

# Task 04: Kunden-Authentifizierung durchführen

*TODO* Infos aus dem JIRA extrahieren mTan-Verfahren

# Task 05: Konfiguration eines Jenkins Job

## 10.1 Schritt 1: build.xml mit Ant ohne Jenkins erstellen

Zunächst einmal war es Ziel zu verstehen, wie eine build.xml aufgebaut ist und aus welchen Befehlen sie besteht, um im Nachgang nachvollziehen zu können, auf welchen Grundlagen die Einstellungen im Jenkins aufbauen.

*TODO* einfach eine build.xml einer Extension öffnen und die wichtigsten Schritte beschreiben

### 10.2 Schritt 2: Build-Job mit Jenkins konfigurieren

*TODO* Einrichtung eines Build-Jobs der für den Bau und das Testen der Extension $congstar_eft_rest_api$ zuständig ist. - Dabei muss konfiguriert werden unter welchen Repository, die Extension zu finden ist. - PHPMD

## Task 06: Erstellung von Databuildern

*TODO* Im Zuge dieses Tasks wurden DataBuilder erzeugt, die das Generieren von Mock-Objekten in Unit-Tests erleichtert.

*TODO* Beispiel-Code *TODO* Erläuterung wieso DataBuilder in dem Fall sinnvoll sind

## Task 07: Durchstich OptionsGroups

*TODO* Beschreibung als $eft_core$ extrahieren

## Task 08: Durchstich ContractDuration

*TODO* Beschreibung aus $eft_core$ extrahieren CR 5539 - Madeira - Implementierung REST Endpoint "getOptions"(BE) (13P) Description

Als Entwickler möchte ich den REST Endpoint "getOptionsïmplementieren.

Akzeptanzkriterien:

Controller in der neuen Extension Abstimmung Datenstruktur (auch mit Hinblick auf Optionswechsel) Optionsanreicherungen sind erweitert SSchwestern-Option"(Referenz auf andere Option) ïst-Turbo-Option"(Flag) Hochreichen der Optionen in ein für das Frontend fertiges Format zur Darstellung Folgende Elemente des Mixers sollen redaktionell pflegbar sein (siehe sto191): Texte in mehr DetailsLayer NEU: Beschreibung zum Datenturbo inkl. Preis-Info (ausgewählt und nicht ausgewählt) Mein MixBeschreibung zum Datenschieber auf "0Mein MixBeschreibungen zu Telefonie und SMS auf "0Äbstimmung mit AAX (Null-Option!?) (PO) API Dokumentation erstellt Profiling erstellt

## Task 09: Property hasMixableOptions an Produkt ergänzen

*TODO* nachschauen ob Konversion bei diesem Task mit inbegriffen war

## Task 10: Implementierung von CodeSniffern zur Überprüfung von Namespace-Deklaration, File Doc Comment und Use-Statements von PHP Dateien

Die Aufgabe dieses Tasks, war die Implementierung von CodeSniffern zur Überprüfung der eigens definierten Code-Style Richtlinien des Congstar Web-Teams. Die verschiedenen CodeSniffer überprüfen dabei, die Existenz und Reihenfolge von: Namespace-Deklaration, File Doc Comment und Use-Statements in PHP-Dateien. Wie im Code Block unten zu sehen, sollen folgende Richtlinien überprüft werden:

1. Namespace-Deklaration ist vorhanden
2. Namespace-Deklaration befindet sich direkt unterhalb des PHP-Opening-Tags
3. nach der Namespace-Deklaration folgt eine Leerzeile
4. nach 2. und 3. folgt das File Doc Comment
5. das File Doc Comment beinhaltet die von Aoe vorgegebene Copyright Notice
6. nach dem File Doc Comment folgt eine Leerzeile
7. nach 5. und 6 können Use-Statements folgen
8. wenn Use-Statements vorhanden sind, müssen diese unterhalb des File Doc Comment stehen
9. nach dem File Doc Comment oder nach den optionalen Use-Statements muss ein Leerzeile folgen

```
<?php
namespace Aoe\Checkout\Domain\Model;

/**************************************************************
 * Copyright notice
 *
 * (c) 2015 AOE GmbH <dev@aoe.com>
 *
 * All rights reserved
 **************************************************************/
```

```
use Aoe\Checkout\Domain\Model\Customer\Address;
use Aoe\Checkout\Domain\Model\Customer\LoginData;
use Aoe\Checkout\Domain\Model\Customer\PersonalData;
use Aoe\Checkout\Domain\Model\Payment;

class Customer
...
```

Die Umsetzung der CodeSniffer beruht auf Grundlage des PHP CodeSniffer Scripts von Squizlab [**?**]. Dieses Script scannt zum Beispiel eine PHP-Datei und liefert eine Liste aus Tokens, die den jeweiligen Inhalt der Datei beschreibt. Jedes Token beschreibt dabei, beispielsweise einen PHP-Tag, Leerzeichen, Strings und vieles mehr. Aufbauend darauf wurden die CodeSniffer implementiert. Jeweils für die Namespace-Deklaration, File Doc Comment und Use-Statements wurde ein eigener CodeSniffer in Form einer PHP-Klasse implementiert. Jede CodeSniffer-Klasse implementiert dabei das PHP CodeSniffer Sniff Interface, dass vorschreibt die Methoden register() und process() zu implementieren. Die register Methode wird dazu verwendet zu definieren, auf welchen Tokens der Sniff losgelassen werden soll. Die process Methode wird anschließend jedes mal aufgerufen, wenn das Script auf einen registrierten Token stößt. Ausgehend von der process Methode wurde die eigentliche Logik und Überprüfungen der Code-Style Richtlinien implementiert. Falls ein Verstoß gegen die Richtlinien aufgetreten ist wird ein Fehler oder Warnung geschmissen. Diese Information bei einem auftretenden Fehler oder einer Warnung, wird in eine Output-Datei geschrieben.

Nach der Erstellung der CodeSniffer mussten diese noch ins Projekt integriert werden. Für jeden Extension beziehungsweise Komponente, für die die CodeSniffer eingesetzt werden sollen, wurde der jeweilige Build-Job erweitert. Dafür musste die Einstellung des Jenkins Job der jeweiligen Komponente erweitert werden. Bei den Einstellungen für die verschiedenen Code Checkstyle Analysen, wurde das Ausführen des PHP CodeSniffer Scripts phpcs unter Angabe des Pfads zu den entsprechenden zu prüfenden Dateien angegeben. Zusätzlich wurde die Einstellung getroffen, dass das Ergebnis der CodeSniffer in die Ausgabedatei checkstyle.xml hinzuzufügen. Dadurch ist es möglich innerhalb des Jenkins Job visualisiert zu bekommen, wo eine Warnung oder Fehler aufgetreten ist.

## Task 11: Implementierung eines CodeSniffers zu Überprüfung von Code Coverage Ignore Annotationen

Umsetzung eines weiteren CodeSniffers der den PHP Code auf Code Coverage Ignore Annotations prüfen soll. Mit Hilfe der Code Coverage Ignore Annotations ist möglich gezielt Codezeilen, Methoden oder gar ganze Klassen, von den PHP Metriken unbeachtet zu lassen, sodass die Code Coverage nicht sinkt, falls für diesen Code keine Unit Tests existieren.

Es kann auch manchmal der Fall sein, dass solche Annotations Sinn machen, wenn Code beispielsweise nicht testbar ist. Es war eine Teamentscheidung zu sagen, dass sie die Code Coverage für Unit Tests transparent halten und somit auf Code Coverage Ignore Annotations verzichten möchte.

Folgende Codebeispiele zeigen die Annotations auf die geprüft werden soll.

```
/**
 * @codeCoverageIgnore
 */
```

```
// @codeCoverageIgnoreStart
... hier steht ein bisschen Code
// "codeCoverageIgnoreEnd
```

Sobald eine solche Annotation im Code entdeckt wurde, wird eine Warnung geworfen, die später bei den Ergebnissen des Build-Jobs angezeigt wird.

```
FILE: /path/to/code/yourfile.php
--------------------------------------------------------------------------------
FOUND 1 ERROR(S) AND 1 WARNING(S) AFFECTING 1 LINE(S)
--------------------------------------------------------------------------------
 21 | ERROR   | PHP keywords must be lowercase; expected "false" but found
    |         | "FALSE"
 21 | WARNING | Equals sign not aligned with surrounding assignments
--------------------------------------------------------------------------------
```

## Behobene Issues