I. Inhaltsverzeichnis

II. Tabellenverzeichnis

III. Abbildungsverzeichnis

IV. Abkürzungsverzeichnis

Abkürzung	Langschreibweise
API	**A**pplication **P**rogramming **I**nterface
evtl.	eventuell
ggf.	**gg**egebenen**f**alls
i.d.R.	**i**n **d**er **R**egel
o.g.	oben genannte
PHP	**H**ypertext **P**reprocessor (rekursives Akronym)
sog.	**sog**enannte
z.B.	**z**um **B**eispiel

V. Glossar

Begriff	Erklärung
Eclipse	Entwicklungsumgebung für PHP (u.a.). Ursprünglich für Java, Quelle: http://www.eclipse.org/
Java	Java ist eine Objekt orientierte, plattformunabhängige Programmiersprache, Quelle: http://www.sun.com/java/about/
Magento	Magento ist ein Open-Source Shopsystem, das eine einfache Erweiterung durch Modul ermöglicht, Quelle http://www.magentocommerce.com/company/
PHP	Eine speziell für Web Entwicklung nutzbare Scriptsprache, die an die Programmiersprache C angelehnt ist, Quelle: http://php.net/
Token	Der Token ist eine bestimmte Zeichenkette, die bei der Verifizierung von Twitter genutzt wird, Quelle: http://dev.twitter.com/pages/basic_to_oauth
Web Application Framework	Ein Programmiergerüst, dass für Webanwendungen ausgelegt ist
Zend Framework	Ein Programmiergerüst für PHP, welches einige nützliche Klassen, z.B. für Datenbank Zugriff oder Email Funktionen aufweist, Quelle: http://framework.zend.com/about/overview

1 Zielstellung und Vorgehen

Das Ziel dieser Praxisarbeit ist es einen Überblick über Modultest Frameworks[1] für PHP und deren Aufgabe zu vermitteln. Es wird der Nutzen einer Einbeziehung von Tests in die Programmentwicklung, unabhängig von der genutzten Programmiersprache, untersucht. Außerdem wird die Möglichkeit einer Automatisierung in Form von Modultests, speziell bei der Objekt orientierten Programmierung aufgezeigt und die Arbeit mit PHPUnit erläutert.

Nachdem ein allgemeiner Überblick über die Funktion und den Nutzen von Tests und speziell von Modultests aufgezeigt wurde, wird in dieser Arbeit eine Auswahl von verschiedenen PHP Modultest Frameworks vorgestellt. Dabei werden die Eigenschaften und Funktionen der Frameworks diskutiert und deren praktische Verwendungsmöglichkeiten untersucht. Im Anschluss wird am Beispiel von dem Modul „Sctwitter“, dass beim Preisbock verwendet wird, das Test Framework PHPUnit vorgestellt.

[1]Framework – Programmiergerüst (ermöglicht ggf. vereinfachten Zugriff auf bestimmte Funktionen einer Programmiersprache)

2 PHPUnit Tests beim Preisbock

2.1 Über dotSource

> *"Die dotSource GmbH ist eine inhabergeführte Full-Service-Internetagentur. Als zertifizierter Partner für die Open Source Shopsoftware Magento sowie die Enterprise E-Commerce-Lösung Intershop Enfinity Suite 6 realisieren wir anspruchsvolle E-Commerce-Plattformen - von der Analyse/Konzeption über Design/Implementierung bis zum laufenden Betrieb. Schulungen und Workshops runden das Angebot ab.*
>
> *Darüber hinaus hat sich dotSource als Social Commerce-Agentur im deutschsprachigen Raum einen Namen gemacht: Unsere Social-Shopping-Lösungen erschließen bundesweit sowie international ausgerichteten Versandhändlern und Markenherstellern die Potenziale des Social Commerce."*[2]

Die dotSource GmbH wurde 2006 gegründet und ist in diesem Jahr auf 50 Mitarbeiter gewachsen. Außer dem Preisbock wurden noch eine Reihe von anderen Projekten verwirklicht, wie z.B. Handelskraft[3].

2.2 Über Preisbock

Abbildung 1: Preisbock (offizielles Logo)

PREISBOCK®
JEDEN TAG EIN NEUES PRODUKT.

Quelle: http://www.preisbock.de

> *„Preisbock.de ist eines der größten und zugleich am schnellsten wachsenden Social-Commerce-Angebote in Deutschland. Unter dem Slogan »Jeden Tag ein neues Produkt.« vermarktet das Liveshopping-*

[2]Quelle: [DOT10]

[3]Handelskraft http://www.handelskraft.de/ 30.09.2010

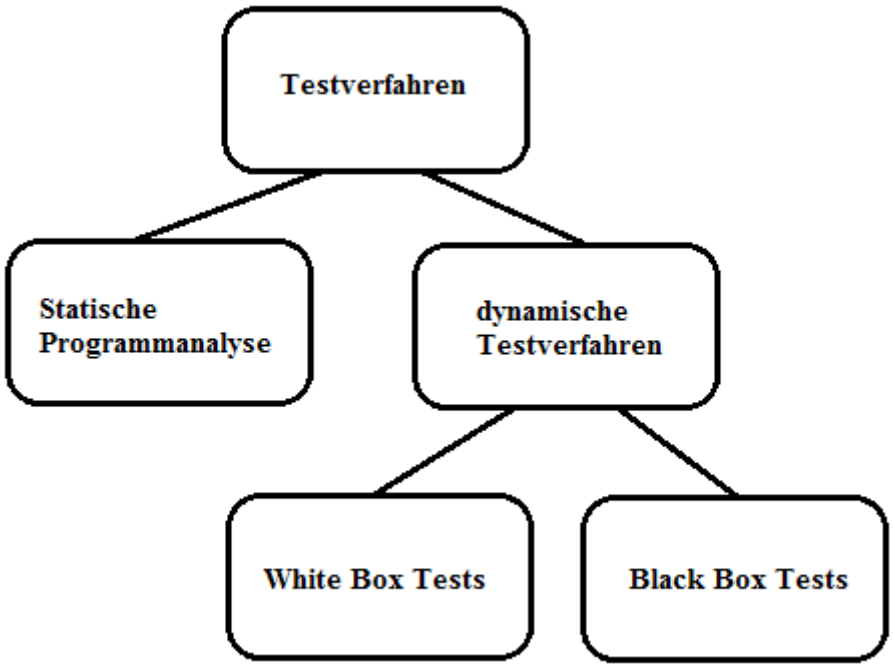
Portal seit 2007 tagtäglich einen Artikel - immer zum aktuell günstigsten Preis im Web.“[4]

Preisbock hat mittlerweile über 50.000 registrierte Kunden und bietet mehrere Live Aktivitäten. Außerdem hat sich um den Preisbock eine freundliche und sehr lebhafte Community gebildet, die rege im Forum des Preisbocks kommuniziert.

2.3 Testverfahren

Das Testen von Software ist ein wichtiger Bestandteil des Entwicklungsprozesses. Mit Hilfe von Tests können Fehler in der Programmierung aufgedeckt und die Software auf ihre Funktionalität geprüft werden. Regelmäßige Tests tragen zur Sicherung der Qualität der Software bei. Das Testen von Software garantiert

Abbildung 2: Testverfahren



Quelle: Eigenes Bild

zwar keine Fehlerfreiheit. Es sollte jedoch trotzdem so intensiv wie möglich betrieben werden.[5] Die Formatierung der Tests unterliegt den selben Regeln wie die Programmierung selbst, denn auch hier ist der Aufbau und die Übersichtlichkeit sehr wichtig.

[4]Quelle: [PRE10]

[5]Vgl. [THA00], S.43

Tests können unter anderem darin unterschieden werden, die zu testende Software auszuführen (dynamische Testverfahren), oder den Quellcode zu analysieren. Wird eine analytische Betrachtung des Quellcodes der Software durchgeführt, so spricht man von einer statischen Programmanalyse. Die statische Analyse ist bei korrekter und vollständiger Ausführung sehr effektiv und bietet eine hohe Vollständigkeit in Bezug auf die Prüfung der Korrektheit des getesteten Programmcodes. Allerdings ist diese Analyse extrem aufwendig und kann in der Softwareentwicklung aus diesem Grund nur sehr selten genutzt werden.[6]

Dynamische Testverfahren hingegen prüfen die Software in dem sie sie ausführen. Sie sind Stichprobenverfahren und können die Korrektheit des getesteten Programms nicht beweisen. Bei diesen Tests wird die Software mit einer direkten Anwendung und je nach Test einer Reihe von möglichen Testfällen geprüft. Die Erstellung und die Ausführung der Tests ist in den meisten Fällen sehr einfach. Die dynamischen Tests decken häufig nicht alle Fehler der getesteten Software auf, können aber erheblich zur Qualitätssicherung beitragen. Aus diesem Grund werden sie in der Softwareentwicklung sehr häufig angewendet.[7]

2.4 White Box und Black Box

Dynamische Tests können unter anderem auch in White Box und Black Box Tests unterteilt werden.

Bei White Box Tests ist der Aufbau der zu testenden Software bekannt. Hier steht im Vordergrund möglichst alle Anweisungen der Software auszuführen und alle Pfade innerhalb der Software abzudecken.[8] Auf diese Weise kann allerdings nur die korrekte Arbeitsweise der Software getestet werden und nicht die Funktionalität, welche in den Spezifikationen festgelegt wurde. Deshalb können mit White Box Tests einzelne Teile der Software getestet werden, die Zusammenarbeit zwischen mehreren Teilen oder Modulen ist damit aber noch nicht gewährleistet. Dies ist der größte Nachteil, der bei White Box Tests

[6] Vgl. [SNE02], S.6

[7] Vgl. [SNE02], S.8 ff.

[8] Vgl. [THA00], S.62 / 63

besteht. Jedoch haben die White Box Tests auch ihre Vorteile. Sie können in der Regel sehr leicht und schnell angefertigt werden, da sich die Tests direkt am Quellcode der Software orientieren.[9] Dazu sind sie individuell einsetzbar, um z.B. einzelne Module, bzw. Teile der Software zu testen und können so auch sehr leicht verwaltet und organisiert werden.[10] Ein weiterer Vorteil ist die große Zahl von Entwicklungsumgebungen für eine Reihe von Programmiersprachen, mit deren Hilfe sich (ggf. durch Plugins[11]) solche Tests sehr einfach erstellen lassen. Ein Beispiel dafür ist Eclipse. Hier kann mit sehr wenig Aufwand für einen beliebigen Teil der Software ein Modultest erstellt werden.

Abbildung 3: Eclipse (offizielles Logo)

Quelle: http://eclipse.org

Bei Black Bock Tests ist der Aufbau der Software nicht bekannt, sondern nur die Spezifikationen der Software. Die Tests werden dazu genutzt, um die erwartete Funktionalität der Software zu prüfen. Bei dieser Art von Test wird mit Testfällen gearbeitet, die die möglichen Abläufe in der Software aufrufen. Wobei hier eine möglichst große Testabdeckung stattfinden sollte. Mit der Größe der Testabdeckung steigt in der Regel auch die Fehlerfreiheit der getesteten Software. Dabei werden aber keine einzelnen Funktionen angesprochen, sondern die Tests orientieren sich an den Spezifikationen der Software und Prüfen die Software durch die direkte Verwendung. Auf diese Weise ist es natürlich nicht möglich einzelne Teile der Software oder Module zu testen, da bei den Black Box Tests zwischen solchen Komponenten nicht unterschieden werden kann.[12] Dadurch ist es nur sehr schwer die Tests zu kategorisieren und einheitlich anzulegen und zu nutzen. Diese Tests können nur bedingt automatisiert ausgeführt werden.[13]

---

[9]Vgl. [THA00], S.71 - 73

[10]Vgl. [THA00], S.53

[11]Plugin – Erweiterung für ein Programm welches eine Funktionserweiterung ermöglicht

[12]Vgl. [THA00], S.76 - 79

[13]Vgl. [THA00], S.125

Auch durch die Anwendung beider Testverfahren, White Box und Black Box Tests, können nicht alle Fehler ausgeschlossen werden. Es müssen aber beide Testarten angewendet werden, da nur so die Qualität der Software auf verschiedene Art und Weise überprüft und gesichert werden kann.[14]

2.5 Modultests

Zu den White Box Tests gehören die Modultests. Diese Tests dienen dem Test von einzelnen Modulen und können in der Regel aber auch zu Test Suiten zusammengefasst werden. Auf diese Weise ist dann ein Modul umfassender oder sogar übergreifender Test möglich. Modultests sind sehr gut für die Objekt orientierte Programmierung geeignet, da damit die einzelnen Module und Klassen sehr schnell und einfach getestet werden können.[15] Außerdem können in den Modultests Objekte oder ggf. auch Attrappen von Objekten erzeugt werden, was bei einigen Test Frameworks integriert ist und bei anderen durch andere Bibliotheken ergänzt werden kann.[16] Mit dem Aufruf einzelner Aussagen können Rückgabewerte von Funktionen geprüft werden. Dabei kann sowohl auf den jeweiligen Datentyp geprüft werden, als auch auf die korrekte Anwendung von Ausnahmebehandlungen. Die Tests können nach der Erstellung einzeln aufgerufen werden oder auch mit der umfassenden Test Suite. Auf diese Weise ist es möglich die Tests dauerhaft zu nutzen und damit zum Beispiel auch nicht nur festzustellen, ob der Datentyp des Rückgabewerts einer Funktion korrekt ist, sondern auch ob bei Änderungen an der Software sich dieser geändert hat.[17] Die Modultests sind i.d.R. sehr leicht zu handhaben. Sie können relativ einfach erstellt werden, sind oft sehr übersichtlich und auch einfach zu verstehen und bieten einen relativ großen Funktionsumfang.[18] Zu den bekanntesten PHP Modultest Frameworks gehören Lime, PHPUnit, SimpleTest, Testilence und das PHP Assertion Unit Framework.[19]
Diese werden im folgenden Kapitel vorgestellt.

---

[14] Vgl. [THA00], S.15

[15] Vgl. [SNE02], S.42

[16] Vgl. [VIG05], S.180 / 181

[17] Vgl. [WES06], S.20

[18] Vgl. [WES06], S.140 / 141

[19] Vgl. [OPS10] und [WIK10]

3 Vergleich verschiedener PHP Modultest Frameworks

3.1 Übersicht

Die o.g. Frameworks werden beschrieben und ihre Vor- und Nachteile werden aufgezeigt. Dadurch soll dem Leser ein Einblick in die Möglichkeiten gegeben werden, die von dem jeweiligen Framework bereitgestellt werden. Im Anschluss wird das beim Preisbock verwendete PHPUnit mit dem Modul Sctwitter als Beispiel genauer dargestellt.

Zuvor eine Übersicht der Frameworks:

Name	Internetadresse
Lime	http://www.symfony-project.org/..
PHPUnit	http://www.phpunit.de/
SimpleTest	http://www.lastcraft.com/simple_test.php
Testilence	http://www.testilence.org/
PHP Assertion Unit Framework	http://jsassertunit.sourceforge.net/docs/phpassertunit.html

Tabelle 1: Übersicht verschiedener PHP Modultest Frameworks

3.2 Lime

Abbildung 4: Symfony (offizielles Logo)

Quelle: http://www.symfony-project.org/

Lime ist ein Modultest Framework, dass ursprünglich für das Web Application Framework Symfony[20] geschrieben wurde. Es kann allerdings auch unabhängig von Symfony genutzt werden. Lime basiert auf der Test::More Perl[21] Bibliothek. Das Modultest Framework Lime beschränkt sich auf einen Funktionsumfang von 16 Funktionen.[22](Im Vergleich

[20]Symfony: http://www.symfony-project.org/ - 29.09.2010

[21]Perl: http://www.perl.org/ - 29.09.2010

[22]Vgl. [SYM10] – Abschnitt „Unit Testing Methods“

dazu hat das Framework PHPUnit 36 Funktionen[23]) Die Ausgabe der Analysedaten erfolgt mittels TAP[24], einem Protokoll für standardisierte Ausgabe.[25]

Obwohl der Funktionsumfang wie oben beschrieben klein ist, können zum Modultest benötigte Grundfunktionen ausgeführt werden. Zum Beispiel gibt es die Möglichkeit bei der Initialisierung des Test Objekts die Anzahl der Testanweisungen zu spezifizieren. Bei der Testausführung wird dann geprüft, ob die Anzahl der Tests absolviert wurde und es wird ggf. ein Hinweis angezeigt.[26]

Für größere Projekte ist das Framework nicht geeignet, da die Anzahl der verfügbaren Methoden des Frameworks begrenzt ist (s.o.). Bei anderen Frameworks wie z.B. PHPUnit gibt es einige spezielle Methoden mehr (z.B. „assertStringMatchesFormat“, welche bei PHPUnit dazu dient String Inhalte auf bestimmte Formate zu prüfen). Für eine solche Aufgabe muss bei dem Lime Framework eine eigene Methode angefertigt werden, was sich dann auch auf die Übersichtlichkeit der Testklasse auswirkt. Denn dadurch muss die Zahl der Methoden erhöht werden. [27]
Für den Privatgebrauch oder für kleinere Projekte ist das Test Framework Lime geeignet, denn es bietet für Modultests benötigte Grundfunktionen (s.o.). Für größere Projekte und Anwender, die oft mit Modultests arbeiten, ist dieses Framework allerdings nicht zu empfehlen, da der Funktionsumfang in anderen Frameworks vielfältiger ist.(s.o.)

[23]Vgl. [PHP10] – Abschnitt „PHPUnit API“

[24]TAP: Test Anything Protocol http://testanything.org/wiki/index.php/Main_Page – 29.09.2010

[25]Vgl. [SYM10]

[26]Vgl. [SYM10] – Abschnitt „Testing Parameters“

[27]Vgl. [SYM10] – Abschnitt „Unit Testing Methods“ und [PHP10] – Abschnitt „PHPUnit API“

3.3 PHPUnit

PHPUnit gehört zu den xUnit Frameworks, eine Reihe von standardisierten Frameworks für verschiedene Programmiersprachen. Das erste Framework dieser Reihe war SUnit für die Programmiersprache Smalltalk. Außerdem gibt es beispielsweise noch DUnit für Delphi und JUnit für Java. PHPUnit ist plattformunabhängig und wird in verschiedenen bekannten Projekten genutzt. Zum Beispiel im Zend Framework, welches u.a. zur Entwicklung in der dotSource GmbH genutzt wird. PHPUnit bietet sehr viele Funktionen und ist auch für größere Projekte und Testreihen sehr gut geeignet, da die Übersichtlichkeit auch bei einer komplexen Testsuite gewährleistet ist. Außerdem gibt es die Möglichkeit Objekt Attrappen zu erstellen und so gewisse Funktionalitäten zu prüfen. Für die Analyse der Testdaten wird bei PHPUnit PHP Code Coverage[28] verwendet. Die Dokumentation für PHPUnit beschreibt alle beinhalteten Funktionen und zeigt deren Funktionsweise anhand von Beispielen.[29]

Abbildung 5: PHPUnit (offizielles Logo)

Quelle: http://www.phpunit.de

Die Vorteile von PHPUnit sind hauptsächlich der strukturierte Aufbau, die leichte Integration und der Funktionsumfang, der bei größeren Projekten und einer höheren Zahl von benötigten Funktionen deutlich wird, was am Beispiel vom Modul Sctwitter weiter unten verdeutlicht wird. Außerdem gibt es mit PHPDoc die Möglichkeit, die Testklassen zu kommentieren und zu dokumentieren. Da PHPUnit im Zend Studio[30] integriert ist, ist es sehr schnell möglich für vorhandene Klassen entsprechende PHPUnit Testklassen und Testsuiten zu erstellen.

---

[28]PHP Code Coverage http://www.phpunit.de/manual/current/en/code-coverage-analysis.html 30.09.2010

[29]PHPUnit Online Dokumentation http://www.phpunit.de/manual/current/en/writing-tests-for-phpunit.html 30.09.2010

[30]Zend Studio http://www.zend.com/de/products/studio/ - 30.09.2010

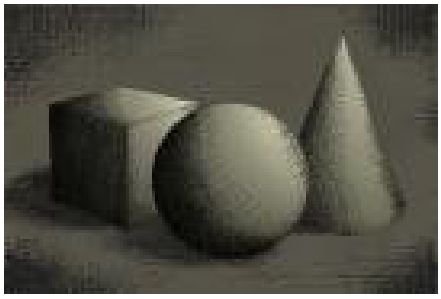
Für sehr kleine Projekte und Privatanwender bietet das Framework die gleichen Vorteile wie für größere Projekte. Allerdings entspricht der Funktionsumfang des Frameworks wie oben beschrieben 36 verschieden Funktionen und wird deshalb bei kleinen Projekten je nach Anforderungen evtl. nicht benötigt.

Für den Author überwiegen bei dem Modultest Framework PHPUnit eindeutig die oben beschriebenen Vorteile. Es bietet eine ganze Reihe von Grundfunktionen und erweiterten Funktionen. Außerdem ist die Dokumentation mit PHPDoc möglich. Gerade für große Projekte wie den Preisbock ist das Framework besonders gut geeignet, weshalb es wahrscheinlich auch im Zend Studio integriert ist.

3.4 SimpleTest

SimpleTest ist dem PHPUnit Framework sehr ähnlich. Zusätzlich zu den Modultests bietet das Framework auch die Möglichkeit Attrappen von Objekten zu erzeugen (sog. „Mock Objects“) und des Weiteren sogar einen HTTP Client zu simulieren und so ggf. auch komplexe Eingaben, wie das Absenden einer Form in die Tests mit einzubeziehen. Dadurch ergibt sich eine erhöhte Zahl von Funktionen innerhalb eines Frameworks. Allerdings kann dies die Anwendung des Frameworks, bei einer ausgiebigen Verwendung der Funktionen, auch sehr komplex gestalten.

Abbildung 6: SimpleTest (offizielles Logo)



Quelle: http://www.lastcraft.com/simple_test.php

Diese Funktionen sind nicht für Modultests geeignet, da die Arbeitsweise der Tests damit eher in Richtung Black Box Tests geht. Hier wird nicht mehr die Funktion einzelner Module und Klassen getestet, sondern die Funktionalität entsprechend der Spezifikationen. Das Test Framework ist ebenfalls plattformunabhängig und es gibt auch ein Plugin für Eclipse, mit dem die Modultests innerhalb der Entwicklungsumgebung erstellt werden können.

Die Kombination der Testmöglichkeiten bietet den Vorteil, dass eine Reihe von White Box und Black Box Tests erstellt werden kann. Außerdem bietet das Framework den nahezu gleichen Funktionsumfang wie PHPUnit.

Die Vereinigung von White Box und Black Box Tests innerhalb eines Frameworks hat den Nachteil, dass die Tests sehr schnell unübersichtlich werden können, sobald HTTP Simulation und Modultests nicht getrennt angelegt werden. Sonst ist es kaum möglich die Tests in größerem Umfang zu überblicken und richtig einzusetzen.

Das Framework bietet vielfältige Testmöglichkeiten, die das Testen vereinfachen kann. Werden die angebotenen Funktionen aber nicht benötigt, so ist ein großer Teil des Frameworks überflüssig. Das Framework sollte je nach Projekt und Anforderungen verwendet werden.

3.5 Testilence

Das Framework Testilence orientiert sich an JUnit und kennzeichnet sich durch ähnliche Funktionsweise. Allerdings ist es bei diesem Framework nur möglich eine Aussage („Assertion“) pro Testmethode durchzuführen. Im Gegensatz dazu können z.B. bei JUnit oder PHPUnit so viele Aussagen pro Testmethode wie gewünscht gemacht werden. Diese Eigenschaft bringt sowohl Vor- als auch Nachteile, welche im Folgenden beschrieben werden. Testilence ist unter anderem auch in der Werkzeug- und Bibliotheks- Kollektion BEER[31] beinhaltet, welche eine umfangreiche Testmöglichkeit für PHP bietet. Testilence benötigt zur Ausführung PHP Version 5.1.3 oder höher.
Das Modultest Framework kann in folgenden Betriebssystemen genutzt werden:

- BSD
- Linux
- UNIX

Außerdem kann es auch unter Windows genutzt werden, was allerdings teilweise zu Fehlern führen kann.

[31]BEER Werkzeug- und Bibliotheks- Kollektion http://beer.sigpipe.cz/ - 30.09.2010

Die Begrenzung der Anzahl der Aussagen pro Testmethode hat den Vorteil, dass die einzelnen Tests übersichtlich sind. Außerdem soll auf diese Weise vermieden werden, dass Aussagen übersprungen und nicht ausgeführt werden, was bei Mehrfachaussagen der Fall sein kann, sobald bei der ersten Aussage ein Abbruch erfolgt. Im Vordergrund steht hierbei aber die einfache Lesbarkeit des Quellcodes.[32]

Ein Nachteil dabei ist, dass dadurch die Anzahl der Testmethoden erheblich steigt. Zusätzlich muss natürlich für jede der Testmethoden ein eigener Name vergeben werden, was sehr aufwendig ist, sobald die Tests für ein größeres Modul oder Projekt erzeugt werden. Außerdem ist es auch wegen der Übersichtlichkeit vorteilhaft, wenn mehrere Aussagen in eine Testmethode eingefügt werden, z.B. wenn diese Aussagen das gleiche Objekt betreffen, oder eine Funktion mehrere mögliche Rückgabewerte besitzt, können hier verschiedene Tests angewendet werden.

Testilence hat eine ähnliche Funktionalität wie PHPUnit und besitzt daher auch einige Vorteile, die auch diesem Framework zuzusprechen sind. Die Tatsache, dass allerdings nur eine Aussage pro Testmethode gemacht werden kann (s.o.) bietet nicht nur Vorteile und so ist jedem Benutzer individuell überlassen, ob er daraus einen Vor- oder Nachteil zieht.

3.6 PHP Assertion Unit Framework

PHP Assertion Unit basiert auf dem Javascript Assertion Unit. Hier wird lediglich eine PHP Klasse genutzt um die Funktionen des Javascript Assertion Unit zu nutzen. Der Funktionsumfang weist die Grundfunktionen des PHPUnit Frameworks auf, allerdings wird hier im Hintergrund Javascript genutzt.

Die Integration von PHP in das Javascript Framework nicht perfekt. Es gibt keine Dokumentation, nur eine kleine Übersicht über Beispiele mit Javascript. Als Ausgabe der Analysedaten dient ein beliebiger Browser.[33]

[32]Vgl. http://www.testilence.org/ - 30.09.2010

[33]Vgl. http://jsassertunit.sourceforge.net/docs/phpassertunit.html - 30.09.2010

4 Einsatz von PHPUnit beim Preisbock

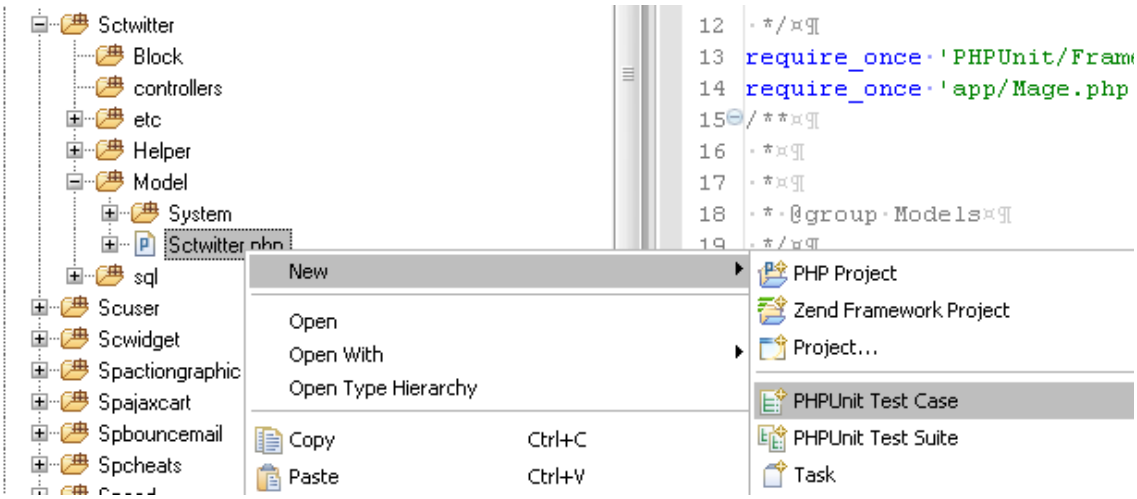
4.1 Allgemein

Abbildung 7: Zend Framework (offizielles Logo)

Quelle: http://framework.zend.com/

Der Preisbock Shop wird auf der Basis von Magento mit der Programmiersprache PHP entwickelt. Hierfür wird die Entwicklungsumgebung Zend Studio genutzt, was eine Reihe von Vorteilen mit sich bringt, da das Magento Shop System auf der Basis des Zend Frameworks entwickelt wurde und damit alle Vorzüge dieses Frameworks genutzt werden können. Das Zend Studio basiert auf Eclipse und bietet eine leichte Nutzung von PHPUnit. Hier können sehr einfach die Modultests aus den bestehenden Klassen gebildet werden, indem entweder selbst eine Klasse erstellt wird und die Testmethoden eingefügt werden, oder die folgende Funktion des Zend Studios genutzt wird:

Abbildung 8: Erzeugung einer PHPUnit Testklasse im Zend Studio



Quelle: Eigenes Bild

Über diese wird eine Testklasse erzeugt, welche alle benötigten Funktionen mit dem Hinweis auf Vervollständigung schon beinhaltet. Die Testfunktionen müssen dann nur noch mit den entsprechenden Funktionen gefüllt werden und damit ist die Testklasse schon fertig. Außerdem wird vom Zend Studio PHPDoc[34] unterstützt, was die Dokumentation der Software erheblich erleichtert. Die Modultests dienen beim Preisbock der Qualitätssicherung und werden i.d.R. vom jeweiligen Entwickler eines Moduls erstellt. Dies geschieht meistens im Zuge der Modulerstellung, kann aber auch im Nachhinein eingefügt werden.
Die Komplexität richtet sich nach dem jeweiligen Umfang des Moduls. Umso größer der Funktionsumfang eines Moduls ist, desto höher fällt auch die Anzahl der Testmethoden aus. Die Modultests werden innerhalb des Projekts in einer eigenen Verzeichnisstruktur (<Projekt>/tests/) abgelegt, um die Übersichtlichkeit zu optimieren:

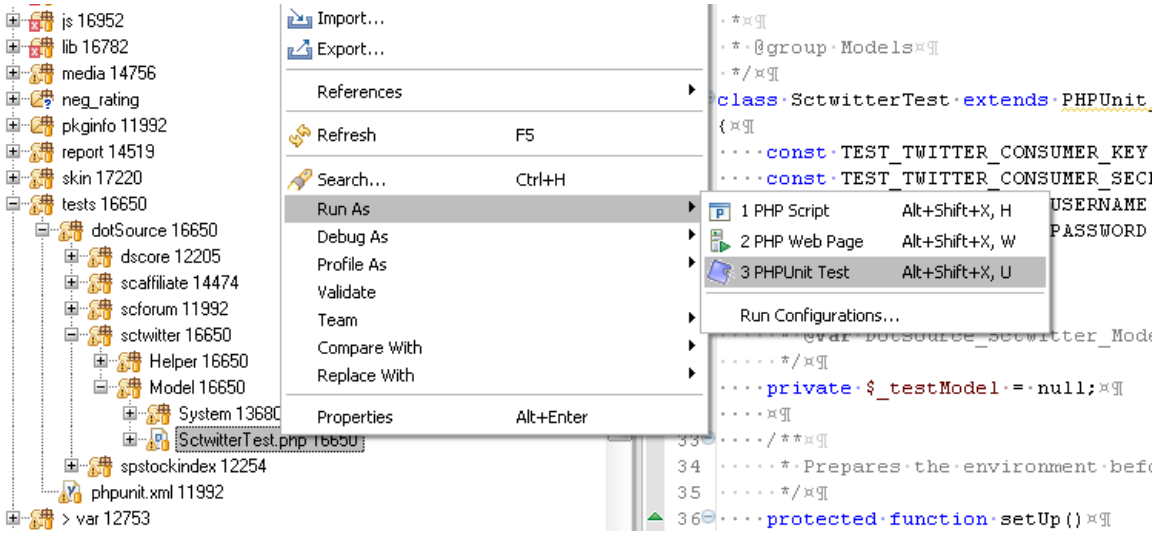
Abbildung 9: Auszug aus der Verzeichnisstruktur des Projekts

Quelle: Eigenes Bild

[34]PHPDoc Dokumentator für PHP http://www.phpdoc.org/ - 30.09.2010

Dadurch, dass die Module und die zugehörigen Modultests in verschiedenen Verzeichnissen untergebracht sind, können diese jeweils leichter verwaltet werden. Die PHPUnit Tests können direkt innerhalb des Zend Studios oder von der Konsole aus aufgerufen werden.

Abbildung 10: PHPUnit im Zend Studio



Quelle: Eigenes Bild

Abbildung 11: PHPUnit in der Konsole

Quelle: Eigenes Bild

Wobei die Ausführung mit der Konsole beim Preisbock vorgezogen wird, da für die Ausführung das Zend Studio nicht geöffnet werden muss und die Tests somit schneller verfügbar sind.
Modultests werden möglichst für jedes Modul angelegt, damit diese schnell und einfach auf Korrektheit geprüft werden können. Auch bei Änderungen kann geprüft kann, ob die Funktionsweise der einzelnen Teile der Module weiterhin gegeben ist und Fehler können sehr schnell aufgedeckt werden.
Außer den Modultests gibt es innerhalb des Entwicklungsprozesses auch Black Box Tests. Diese werden allerdings durch ein Test Team durchgeführt. Das Test Team hält sich dabei an die Spezifikationen der Software. Auch die Black Box Tests spielen eine wesentliche Rolle im Gesamtprozess. Durch die Zusammenwirkung von White Box und Black Box Tests wird die Qualität der Software enorm gesteigert, da sowohl Fehler innerhalb der Software, als auch Fehler in der Funktionalität der Software aufgedeckt werden können.

4.2 Beispiel am Preisbock Modul Sctwitter

4.2.1 Modulbeschreibung

Abbildung 12: Twitter (offizielles Logo)

Das Modul Sctwitter stellt eine Schnittstelle zwischen dem Preisbock Shop und dem Online Dienst Twitter[35] dar. Dazu gehört unter anderem der Verbindungsaufbau zur Twitter API und die Übermittlung von bestimmten Daten. Da sich vor kurzer Zeit die Authentifizierung bei Twitter geändert hat, musste einige Funktionen des Moduls angepasst werden. Twitter ermöglicht seit dem 16.08.2010 nur noch die Authentifizierung mit dem OAuth[36] Verfahren. Hierbei werden bestimmte Schlüssel und Token verwendet, was das Verfahren sicherer macht im Gegensatz zur bisherigen Authentifizierung.

Quelle: http://twitter.com

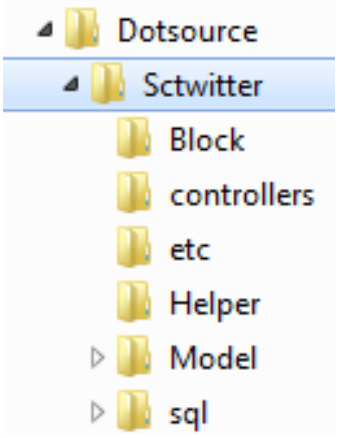
[35]Twitter – Online Dienst zur persönlichen Nachrichtenverteilung http://twitter.com/ 30.09.2010
[36]Twitter Oauth – Authentifizierungsverfahren http://dev.twitter.com/pages/oauth_faq 30.09.2010

Außerdem bietet das Modul einige Funktionen in Zusammenhang mit dem Produktangebot auf dem Preisbock. Hier werden für bestimmte Ereignisse Aktionen ausgeführt. Wenn z.B. ein neues Tagesprodukt beim Preisbock angeboten wird, wird über das Modul ein neuer Twitter Post erzeugt, der die Benutzer des Twitter Dienstes, welche den Twitter Benutzer des Preisbocks beobachten, über das neue Angebot informiert. Außerdem wird je nach Einstellung im Backoffice des Shops bei einem bestimmten Lagerbestand des Produkts ebenfalls ein neuer Twitter Post erstellt, der die Benutzer darüber informiert, dass das Produkt bald ausverkauft sein wird. Am Määhday[37] wird keine Angabe über den Lagerbestand der Produkte gemacht.

4.2.2 Anwendung der Modultests bei Sctwitter

Das Modul Sctwitter basiert, wie auch das gesamte Magento Shop System, auf der MVC[38] Architektur. Das Modul hat den folgenden Aufbau:

Abbildung 13: MVC Architektur im Modul Sctwitter



Quelle: Eigenes Bild

Im Modul werden allerdings keine Blöcke und Controller benötigt, sowie auch kein SQL Script. Es besteht nur aus den folgenden Komponenten.

[37]Määhday - Aktionstag beim Preisbock, an dem mehrere Produkte angeboten werden

[38]MVC – Model Controller View Architektur http://www.magentocommerce.com/knowledge-base/entry/magento-for-dev-part-1-introduction-to-magento/#2 30.09.2010

- config.xml – hier werden alle Modul relevanten Einstellungen gemacht, z.B. die Standard Umgebung (Live / Test):

Abbildung 14: config.xml - Auszug

```
<default>
    <!-- set the default environment value to test, to avoid live posts -->
    <sctwitter>
        <sctwitter_general>
            <environment>test</environment>
        </sctwitter_general>
    </sctwitter>
</default>
```

Quelle: Eigenes Bild

- system.xml – hier werden alle Backoffice[39] System Einstellungen des Moduls definiert, wie z.B. die Einstellung der Umgebung (Live / Test):

Abbildung 15: system.xml - Auszug

```
<environment translate="label">
    <label>Environment</label>
    <comment>either test or live - depending on that setting, you'll post with your test or live account credentials</comment>
    <frontend_type>select</frontend_type>
    <source_model>sctwitter/system_environment</source_model>
    <sort_order>1</sort_order>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>1</show_in_store>
</environment>
```

Quelle: Eigenes Bild

- Config.php (Helper) – für die Konfigurationsverwaltung zuständig:

[39]Backoffice – Administrationsbereich des Internet Shops

Abbildung 16: Config.php (Helper) - Auszug

```php
<?php
/**
 * Copyright (c) 2008-2010 dotSource GmbH.
 * All rights reserved.
 * http://www.dotsource.de
 *
 * Created:
 * 02.03.2010 20:00:00
 *
 * Contributors:
 * mhaupt - initial contents
 * mwetter - 2010-08-11 - from twitter Basic Auth to OAuth
 */
class Dotsource_Sctwitter_Helper_Config extends Mage_Core_Helper_Abstract
{
    // section of sctwitter configs
    const CONFIG_PATH_SECTION    = 'sctwitter';

    // groups of the sctwitter section
    const CONFIG_PATH_GROUP_CREDENTIALS             = 'sctwitter_credentials';
    const CONFIG_PATH_GROUP_BITLY_CREDENTIALS       = 'bitly_credentials';
    const CONFIG_PATH_GROUP_TEST_CREDENTIALS        = 'sctwitter_test_credentials';
    const CONFIG_PATH_GROUP_BITLY_TEST_CREDENTIALS  = 'bitly_test_credentials';
    const CONFIG_PATH_GROUP_GENERAL                 = 'sctwitter_general';

    // fields of the sctwitter groups
    const CONFIG_PATH_BITLY_USERNAME        = 'user_name';
    const CONFIG_PATH_BITLY_API_KEY         = 'api_key';
    const CONFIG_PATH_ENVIRONMENT           = 'environment';

    // fields of the twitter oauth
    const TWITTER_OAUTH_SIGNATURE_METHOD = 'HMAC-SHA1';
    const CONFIG_PATH_TWITTER_OAUTH_CONSUMER_KEY        = 'consumer_key';
    const CONFIG_PATH_TWITTER_OAUTH_CONSUMER_SECRET     = 'consumer_secret';
    const CONFIG_PATH_TWITTER_OAUTH_TOKEN               = 'oauth_token';
    const CONFIG_PATH_TWITTER_OAUTH_TOKEN_SECRET        = 'oauth_token_secret';
    //oauth urls
    const TWITTER_OAUTH_CALLBACK_URL        = 'http://www.preisbock.de/index.php';
```

Quelle: Eigenes Bild

- Environment.php – Bereitstellung der Umgebungsvariablen für die Backoffice Einstellungen

Abbildung 17: Environment.php (Model) - Auszug

```php
<?php
/**
 * Copyright (c) 2008-2010 dotSource GmbH.
 * All rights reserved.
 * http://www.dotsource.de
 *
 * Created:
 * 05.03.2010 09:50:30
 *
 * Contributors:
 * mhaupt - initial contents
 *
**/
?>
<?php
class Dotsource_Sctwitter_Model_System_Environment extends Varien_Object
{
    /**
     * returns the option array of all main categories in the system
     *
     * @return array
     */
    public function toOptionArray()
    {
        $options = array(
            Mage::helper('sctwitter')
                ->__(Dotsource_Sctwitter_Helper_Config::ENVIRONMENT_LIVE) =>
                    Dotsource_Sctwitter_Helper_Config::ENVIRONMENT_LIVE,
            Mage::helper('sctwitter')
                ->__(Dotsource_Sctwitter_Helper_Config::ENVIRONMENT_TEST) =>
                    Dotsource_Sctwitter_Helper_Config::ENVIRONMENT_TEST
        );

        return $options;
    }

}
```

Quelle: Eigenes Bild

- Sctwitter.php – Funktionalität der Twitter Schnittstelle:

Abbildung 18: Sctwitter.php (Model) - Auszug

```php
<?php
/**
 * Copyright (c) 2008-2010 dotSource GmbH.
 * All rights reserved.
 * http://www.dotsource.de
 *
 * Created:
 * 02.03.2010 20:00:00
 *
 * Contributors:
 * mhaupt - initial contents
 * mwetter - 2010-08-11 - from twitter Basic Auth to OAuth
 */
class Dotsource_Sctwitter_Model_Sctwitter extends Mage_Core_Model_Abstract
{
    /**
     * config helper
     * @var Dotsource_Sctwitter_Helper_Config
     */
    protected $_configHelper = null;

    /**
     * constructor of the class, inits the config helper
     */
    public function __construct()
    {
        $this->_configHelper = Mage::helper('sctwitter/config');
    }

    /**
     * replaces the var with a replacement and returns the content
     *
     * @param string $varName
     * @param string $replacement
     * @param string $content
     * @return string
     */
    public function replaceVarWithContent($varName, $replacement, $content)
```

Quelle: Eigenes Bild

Für die Helper und Model Klassen wurden Modultests angelegt, die alle Funktionen dieser Klassen testen. Dazu werden in den Testklassen ggf. Testobjekte und Testdaten angelegt. Außerdem befindet sich in jeder Testklasse eine setUp und eine tearDown Methode. Diese Methoden werden i.d.R. Bei PHPUnit Tests genutzt um vor dem Test allgemeine Vorbereitungen für die Umgebung zu machen und sobald der Test abgeschlossen ist, die Umgebung wieder aufzuräumen, indem z.B. Objekte wieder gelöscht werden.
In diesen Testklassen werden nun die Rückgabewerte der Funktionen der Sctwitter Klassen überprüft.

Die folgenden Aussagen werden hier getestet:

- Prüfen eines String als Rückgabewert

```
 /**
     * tests the getTwitterOauthSignatureMethod() method and
should return a string
     */
    public function testGetTwitterOauthSignatureMethod()
    {
        $this->assertType('string', $this->_testModel-
>getTwitterOauthSignatureMethod());
    }
```

- Prüfen eines Boolean als Rückgabewert:

```
    public function testInit()
    {
        $this->assertTrue(true);
    }
```

- Prüfen eines Array Inhalts:

```
    /**
     * test the getEnvironment() Method, should return a
string and should
     * be live or test
     */
    public function testGetEnvironment()
    {
        $live =
Dotsource_Sctwitter_Helper_Config::ENVIRONMENT_LIVE;
        $test =
Dotsource_Sctwitter_Helper_Config::ENVIRONMENT_TEST;

        $testArray = array($live, $test);
```

```
        $this->assertType(
            'string',
            $this->_testModel->getEnvironment()
        );

        $this->assertContains(
            $this->_testModel->getEnvironment(),
            $testArray
        );
    }
```

- Prüfen eines Array als Rückgabewert:

```
    /**
     * tests the return value of the toOptionArray() method
     */
    public function testToOptionArray()
    {
        // should return an array
        $this->assertType(
            'array',
            $this->_testModel->toOptionArray()
        );
    }
```

- Prüfen eines Zend_Service_Twitter Objekt als Rückgabewert (hier muss die login Funktion vorangehen, damit das o.g. Objekt bei der updateStatus Funktion zurückgegeben wird):

```
        $twitter = $this->_testModel->login(
            self::TEST_TWITTER_CONSUMER_KEY,
            self::TEST_TWITTER_CONSUMER_SECRET
        );
```

```
        $updateMessage = 'here\'s an update for you all on: '
. now();

     $this->assertType(
            'Zend_Service_Twitter',
            $this->_testModel->updateStatus($twitter,
$updateMessage)
        );
```

- Prüfen auf den gleichen Inhalt von zwei Variablen:

```
      $testVarName = '{{{testVar}}}';
        $testString  = 'my ' . $testVarName . ' variable';
        $replacement = 'replacement';
        $correctString = 'my ' . $replacement . ' variable';

        $testString = $this->_testModel-
>replaceVarWithContent(
            $testVarName,
            $replacement,
            $testString
        );

        $this->assertSame($testString, $correctString);
```

5 Fazit

Modultests sind ein sehr wichtiger Bestandteil der Qualitätssicherung und tragen erheblich zum Entwicklungsprozess bei. Hierfür wird ein geeignetes Werkzeug benötigt, das eine regelmäßige Testerstellung, eine gute Übersichtlichkeit, sowie eine sehr ausgeprägte Dokumentation gestattet. Mit PHPUnit wird diesen Ansprüchen gerecht und die dotSource nutzt dies aus diesem Grund im Preisbock Projekt. PHPUnit ist schon seit langer Zeit eines der populärsten Modultest Frameworks und wird dieses Ansehen auch noch weiter behalten, da die Entwicklung stets weiter getrieben wird. Zusätzlich erscheinen auch andere Programme und Plugins, welche PHPUnit nutzen und die Arbeit damit noch vereinfachen und für einige Fälle auch spezialisieren. Wie zum Beispiel Selenium IDE[40], mit welchem auch durch Zusammenarbeit mit PHPUnit und unter Verwendung von Selenium RC[41] Webanwendungen getestet werden können. Das Modultest Framework ist ein wesentlicher Bestandteil bei der Entwicklung mit PHP beim Preisbock.

[40]Selenium IDE http://seleniumhq.org/projects/ide/ 30.09.2010

[41]Selenium RC http://seleniumhq.org/projects/remote-control/ 30.09.2010

VI. Quellen

Kürzel	Quelle
[DOT10]	http://www.dotsource.de/ - Abruf: 30.09.2010
[OPS10]	http://www.opensourcetesting.org/unit_php.php - Abruf: 30.09.2010
[PHP10]	http://www.phpunit.de/manual/3.5/en/index.html - Abruf: 30.09.2010
[PRE10]	Http://www.preisbock.de/ - Abruf: 30.09.2010
[SNE02]	Sneed, Harry M., Winter, Mario „Testen objektorientierter Software“, 2002, Carl Hanser Verlag
[SYM10]	http://www.symfony-project.org/book/1_0/15-Unit-and-Functional-Testing - Abruf: 30.09.2010
[THA00]	Thaller, Georg Erwin „Software-Test Verifikation und Validation“, 2000, Verlag Heinz Heise
[VIG05]	Vigenschow, Uwe „Objektorientiertes Testen und Testautomatisierung in der Praxis“, 2005, dpunkt.verlag
[WES06]	Westphal, Frank „Testgetriebene Entwicklung mit JUnit & FIT“, 2006, dpunkt.verlag
[WIK10]	http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#PHP - Abruf: 30.09.2010

VII. Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich meine Praxisarbeit mit dem Thema

„Einsatz von PHPUnit am Beispiel von Preisbock.de“

ohne fremde Hilfe angefertigt habe,

dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und dass ich meine Praxisarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____ _____

Ort, Datum Unterschrift