Guten Tag,

Bevor ich zuerst auf Joomla! und dann später auf Codeception gestoßen bin, konnte ich mir nicht vorstellen, dass die Hindernisse, die mir beim Testen oft im Wege standen, tatsächlich etwas zur Seite gerückt wurden.

Ich habe sehr viel Zeit mit dem Testen von Software - und früher noch mehr mit den Problemen die aufgrund von fehlenden Tests entstanden sind - verbracht!

Nun bin ich davon überzeugt, dass

- Tests, die möglichst zeitnah zur Programmierung,
- automatisch,
- häufig – idealerweise nach jeder Programmänderung

durchgeführt werden mehr einbringen als kosten. Und ich gehe sogar noch weiter: Testen kann sogar Spaß machen.

Ich musste mir auf meinem Lernweg alle Informationen an verschieden Stellen zusammen suchen und selbst eine Menge nicht immer schöner Erfahrungen sammeln. Deshalb habe ich mich entschieden das Buch zu schreiben, welches ich auf meinem Weg gerne zur Verfügung gehabt hätte.

> **RANDBEMERKUNG: Test-Methoden sind nachhaltig**
> Es lohnt sich Testmethoden zu erlernen, denn diese können nicht nur mit jeder Programmiersprache genutzt werden, sie sind anwendbar auf so gut wie jedes Menschenwerk. Sie sollten fast alles beizeiten einmal Testen :)
> Testmethoden sind unabhängig von bestimmten Softwarewerkzeugen.
> Das was Sie in diesem Buch lesen ist, im Gegensatz zu Programmiertechniken oder Programmiersprachen die oft Modeerscheinungen sind, zeitlos.

Welche Themen behandelt dieses Buch?

Das Kapitel *Softwaretests - eine Einstellungssache?* behandelte die Frage, warum man Zeit in Softwaretests investieren sollte. Dabei erläutere ich auch die wichtigsten Testkonzepte.

Praxisteil: Die Testumgebung einrichten Todo

Codeception – ein Überblick

Unittests

Testduplikate

Funktionstest

Akzeptanztests

Analyse

Was Sie zur Bearbeitung dieses Buchs benötigen

Welche Ausstattung brauchen Sie? Sie müssen nicht sehr viele Voraussetzungen erfüllen, um die Inhalte dieses Buches bearbeiten zu können. Natürlich müssen Sie über einen heute üblichen Computer verfügen. Auf diesem sollte eine Entwicklungsumgebung und ein lokaler Webserver installiert sein. Weitere Informationen und Hilfen zur Einrichtung Ihres Arbeitsplatzes finden Sie im Kapitel *Praxisteil: Die Testumgebung einrichten.*

Was sollten Sie persönlich für Kenntnisse mitbringen? Sie sollten die grundlegenden PHP Programmiertechniken beherrschen. Idealerweise haben Sie bereits eine kleine oder mittlere Webapplikation programmiert. Auf jeden Fall sollten Sie wissen, wo Sie PHP Dateien auf Ihrem Entwicklungsrechner ablegen und wie Sie diese im Browser aufrufen. Auf einem lokal installierten Webserver geht dies meist über eine URL in der Form `http::://localhost/datei.php`.

Das Allerwichtigste ist aber: Sie sollten Spaß daran haben neue Dinge auszuprobieren.

Wer sollte dieses Buch lesen

Jeder, der der Meinung ist, dass Softwaretests reine Zeitverschwendung sind, sollte einen Blick in dieses Buch werfen. Insbesondere möchte ich die Entwickler einladen dieses Buch zu lesen, die schon immer Tests für ihre Software schreiben wollten – dies aber aus den unterschiedlichsten Gründen bisher nie konsequent bis zu Ende gemacht haben. **Codeception** könnte ein Weg sein, Hindernisse aus dem Weg zu räumen.

Infos zu Formatierungen und Verweisen

Bei Verweisen habe ich bemüht deutsche Quellen zu finden. Leider war dies nicht immer möglich. Oder die Inhalte, auf die ich in englischer Sprach verweisen konnte, waren inhaltlich passende. Deshalb finden Sie an manchen Stellen Verweise auf Internetseiten die in englischer Sprache verfasst sind.

Ein Buch im Bereich Programmierung enthält **Programmcode**. Um diesen Code vom normalen Fließtext abzugrenzen habe ich ihn in einer anderen Schriftart etwas eingerückt. Relevante Teile sind fett abgedruckt.

```
/**
* @dataProvider provider_credentials_emptypassword
*/
public function testonUserAuthenticate_EmptyPassword($credentials) {
        require_once JINSTALL_PATH . '/libraries/joomla/event/dispatcher.php';
        $subject = \JeventDispatcher::getInstance();
        $config = array(
                'name' => 'jooma"',
                'type' => 'authentication',
                'params' => new \JRegistry
        );
```

Kommandozeileneingaben sind ebenfalls in einer anderen Schriftart eingerückt. Zusätzlich habe ich diese grau hinterlegt und eingerahmt.

```
$ tests/codeception/vendor/bin/codecept generate:test unit
/suites/plugins/authentication/joomla/PlgAuthenticationJoomla
Test was created in
/var/www/html/gsoc16_browser-automated-tests/tests/codeception/unit//suites/plugins/authentication/
joomla/PlgAuthenticationJoomlaTest.php
```

Randbemerkungen ergänzen den eigentlichen Inhalt. Zum Verständnis der Inhalte dieses Buches sind sind diese Texte nicht notwendig. Ich habe Randbemerkungen eingerückt und mit einem Strich am linken Rand versehen.

Exkurs - Was bedeutet das Zeichen & vor dem Parameter $response
Mithilfe des vorangestellten &-Zeichens können Sie eine Variablen an eine Methode per Referenz übergeben, so dass die Methode ihre Argumente modifizieren kann. Beispiel:
function add (&$num) {
$num++;

```
}
$number = 0;
add($number);
echo $number;
```

Wichte Merksätze sind zum leichteren Wiederfinden grau hinterlegt.

WICHTIG!
final, private und static Methoden können nicht mit PHPUnit Stub Objekten genutzt werden. PHPUnit unterstützt diese Methoden nicht.

Softwaretests - eine Einstellungssache?

> Unter Testen versteht man den Prozess des Planens, der Vorbereitung und der Messung, mit dem Ziel, die Eigenschaften eines IT-Systems festzustellen und den Unterschied zwischen dem tatsächlichen und dem erforderlichen Zustand aufzuzeigen.
> [Koomen und Spillner]

(todo Einleitung)

In diesem ersten Kapitel erkläre ich Ihnen theoretische Grundlagen. Wenn Sie lieber praktisch starten, können Sie das erste Kapitel zunächst links liegen lassen und mit dem praktischen zweiten Kapitel beginnen. Ich verweise an passender Stelle immer mal wieder auf diesen ersten Theorie-Teil.

Ich habe bewusst ein fertiges System, hier konkret das Content Management System Joomla!, als Beispiele für Erklärungen gewählt. Die Nutzung einer fertigen Anwendung anstelle von ausschließlich kleinen selbst erstellen Codebeispielen hat Vorteile und Nachteile. Joomla! stellt einen Rahmen zur Verfügung, innerhalb dessen ein Programmierer eine Erweiterung programmieren kann. Aus diesem Rahmen kann ich Testbeispiele wählen. Ich muss also nicht immer das Rad selbst neu erfinden. Nachteilig ist, dass dieser Rahmen teilweise selbst erklärungsbedürftig ist.

Was ist das Ziel dieses ersten Kapitels? Ich hoffe, dass Ihnen nach der Lektüre dieses Kapitels klar ist, warum Softwaretests in einem Projekt eingeplant werden sollten.

Sicherlich wird Ihnen bewusst werden, welchen Einfluss Softwaretests auf Ihre Arbeit haben können. Mir ging es so,

- dass ich sicherer in meiner Arbeit wurde,
- Fehler in Spezifikationen eher gefunden und korrigiert habe,
- meine Vorgehensweise selbst im Vorhinein überdacht habe und
- so qualitativ bessere und fehlerfreie Programme erstellt habe.

Um dies mit praktischen Beispielen zu veranschaulichen tauchen wir am Ende dieses Abschnitts in die Techniken Testgetriebene Programmierung (Test-Driven-Development, kurz TDD) und die verhaltensgetriebene Softwareentwicklung (Behavior-Driven-Development, kurz BDD) ein.

Dieses Kapitel umfasst die Themen

- Warum sollten Sie Software testen?
- Projektmanagement
- Fehler: Referenz nicht gefunden

Warum sollten Sie Software testen?

Software zu testen ist auf den ersten Blick nichts Tolles. Viele sind der Meinung, dass dies eine langweilige Tätigkeit ist! Außerdem erscheint es auch nicht wichtig Software zu testen. Schon im Studium war dieser Themenbereich ganz am Schluss eingeordnet und in meinem Fall blieb dafür gar keine Zeit. Prüfungsrelevant waren Testmethoden nicht. Demotiviert hat mich zusätzlich die Tatsache, das Qualität nicht sicher mit Tests belegt werden kann. Das der ideale Test nicht berechenbar ist hat Howden schon 1977 bewiesen.

Dann habe ich aber das Gegenteil erfahren. Außerdem hat mich der Satz im Google Testing Blog

> „While it is true that quality cannot be tested in, it is equally evident that without testing it is impossible to develop anything of quality. “ [James Whittaker]

nachdenklich gestimmt. Obwohl es stimmt, dass Qualität nicht sicher mit Tests belegt werden kann, ist es ebenso offensichtlich, dass es unmöglich ist, ohne zu Tests etwas

qualitativ Gutes zu entwickeln. Es gibt sogar Entwickler, die noch weiter gehen und sage: „Softwareentwicklung ohne Tests ist wie Klettern ohne Seil und Haken".

> **RANDBEMERKUNG: Warum ist Software fehlerhaft?**
> Ursachen fehlerhafter Software sind menschliche Fehlleistungen. Die meisten Fehler entstehen beim Informationsaustausch - also der Kommunikation.
> - Intrapersonale Kommunikation
> Mögliche Fehlerursachen bei der Informationsverarbeitung innerhalb eines Menschen sind Irrtümer beim Denken und Wahrnehmen. Beispiel: Ein Programmierer weißt den Wert einer Variablen brutto statt netto zu.
> - Interpersonale Kommunikation
> Beim Informationsaustausch zwischen Gesprächspartnern kann ein Erklärungsirrtum auftreten. Jemand hat A gesagt, aber B gemeint.
> - Irrtum bei der Übermittlung
> Fehler treten auch bei der Übermittlung von Informationen auf. Zum Beispiel wenn ein Mitarbeiter Informationen aus einem entgegengenommen Anruf falsch weitergibt.
> - Irrtum beim Entschlüsseln
> Wenn Informationen falsch gelesen oder gehört werden kann es auch zu Fehlern kommen. Immer mehre kommunizieren Menschen nicht in ihrer Muttersprache. In manchen Unternehmen versteht man kein Wort, wenn man nicht die dortige Fachsprache beherrscht. Dies kann zu Missverständnissen führen.
> - Kognitive Einschränkungen
> Wir Menschen habe zu wenig RAM im Gehirn! Unser Langzeitgedächtnis kann zwar viele Informationen speichern. Im Kurzzeitgedächtnis ist aber nur wenig Speicherplatz. Oft reicht dieser nicht aus um zwei Schleifendurchläufe inklusive Kontext nachzuvollziehen! Genau wie ein Maurer ein Haus Stein für Stein baut schreiben wir Programme Zeile für Zeile. Wir betrachten und manipulieren Programme durch ein extrem kleines kognitives Fenster!
> - Nicht-kommunikative Fehlerquellen
> Fehlerhafte Software entsteht aber nicht nur aufgrund von Kommunikationsproblemen. Andere Fehlerquellen sind Mitarbeiter, die nicht über ein ausreichendes fachliche oder projektspezifische Wissen verfügend oder zu viel Stress ausgesetzt sind. Zudem machen Menschen Fehler wenn sie übermüdet, krank oder unmotiviert sind.

Probieren Sie es aus. Integrieren Sie Tests in Ihr nächstes Projekt. Vielleicht springt der Funkte auch bei Ihnen über, wenn Sie das erste Mal hautnah erlebt haben, dass ein Tests Ihnen eine mühsame Fehlersuche erspart hat. Mit einem Sicherheitsnetz von Tests können Sie mit weniger Stress hochwertige Software entwickeln.

Möchten Sie, dass die Software, die Sie programmieren, qualitativ gut ist und Sie selbst entspannter arbeiten können? Dieses Kapitel hat Sie sicher davon überzeugt, dass dies ohne Tests nicht möglich ist.

Testen ist aber auch kein Selbstzweck! Deshalb stellt sich die Frage, wie intensiv und auf welche Art und Weise Tests integriert werden sollten. Und dies ist die ideale Überleitung zum Thema Projektmanagement.

Projektmanagement

Das Magische Dreieck im Projektmanagement beschreibt den Zusammenhang zwischen den **Kosten**, der benötigten **Zeit** und der leistbaren **Qualität**. Ursprünglich wurde dieser Zusammenhang im Projektmanagement erkannt und beschrieben. Sie haben aber sicher schon in anderen Bereichen von diesem Spannungsverhältnis gehört. Es ist bei fast allen betrieblichen Abläufen in einem Unternehmen ein wichtiges Thema.

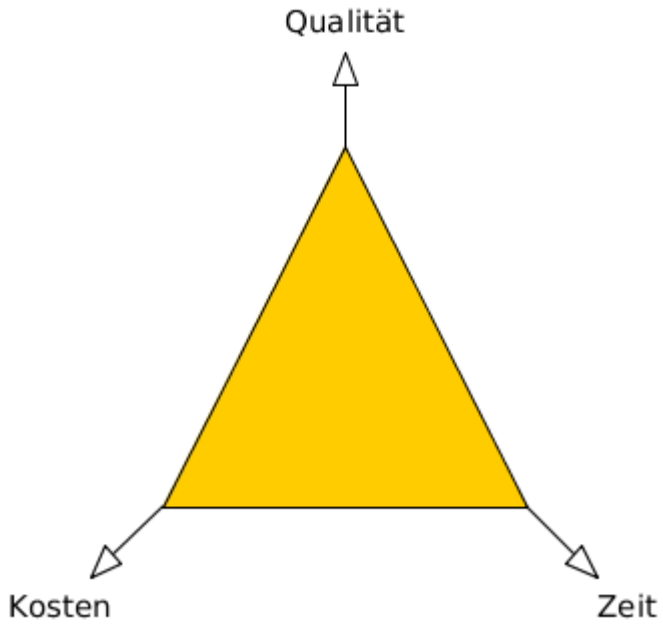


Illustration 1: Das Magische Dreieck im Projektmanagement 965.png

Zum Beispiel werden Überstunden geleistet, um einen Termin einzuhalten; dies erhöht die Kosten. Oder bei geforderte Kosteneinsparungen werden Leistungen gestrichen, um die Kosten zu halten; dies senkt die Qualität des Ergebnisses. Ein letztes Beispiel: Um die Qualität einer Software sicherzustellen, wird zusätzliche Zeit in Tests investiert und der Fertigstellungstermin nach hinten verschoben.

Nun kommt die Magie ins Spiel: Wir überwinden den Zusammenhangs aus Zeit, Kosten und Qualität! Denn, auf lange Sicht kann der Zusammenhang aus Kosten, Zeit und Qualität tatsächlich überwunden werden.

Vielleicht haben auch Sie schon in der Praxis selbst erlebt, dass eine Qualitätssenkung auf lange Sicht keine Kosteneinsparungen zur Folge hat. Die **technische Schuld**, die dadurch entsteht, führt oft sogar zu Kostenerhöhungen und zeitlichem Mehraufwand.

> **RANBEMERKUNG - Technische Schulden**
> Der Begriff Technische Schuld steht für die möglichen Konsequenzen technisch schlecht erstellter Software. Unter der technischen Schuld versteht man den zusätzlichen Aufwand, den man für Änderungen und Erweiterungen an mangelhaft geschriebener Software im Vergleich zu gut geschriebener Software einplanen muss. Martin Fowler unterscheidet folgende Arten von technischen Schulden: Diejenigen, die man bewusst aufgenommen hat und diejenigen, die man ungewollt eingegangen ist. Darüber hinaus unterscheidet er zwischen umsichtigem und risikofreudigem eingehen von technischer Schuld.

	bewusst	**ungewollt**
umsichtig	So sollte es sein :)	Nun haben wir etwas gelernt.
risikofreudig	Wir haben keine Zeit!	Was ist OOP?

Und nun sind wir bei einem Thema angekommen, dass sehr unterschiedlich diskutiert wird. Wie schaffen wir es, Kosten in der Planung genau zu berechnen und im zweiten Schritt, Kosten und Nutzen in idealer Weise zu verbinden?

Kosten Nutzen Rechnung

In der Literatur finden Sie immer wieder niederschmetternde Statistiken über die Erfolgsaussichten von Softwareprojekten. Es hat sich wenig an dem negativen Bild geändert, das bereits eine Untersuchung von A.W. Feyhl in den 90er Jahren aufzeichnete. Hier wurde bei einer Analyse von 162 Projekten in 50 Organisationen die

Kostenabweichung gegenüber der ursprünglichen Planung ermittelt: 70 % der Projekte wiesen eine Kostenabweichung von mindestens 50 % auf!

Da stimmt doch etwas nicht! Das kann man doch nicht einfach so hinnehmen, oder?

Ein Lösungsweg wäre, ganz auf Kostenschätzungen zu verzichten und der Argumentation der #NoEstimates-Bewegung zu folgen. Diese vertritt die Meinung, dass Schätzungen in einem Softwareprojekt unsinnig sind. Ein Softwareprojekt beinhaltet die Erstellung von etwas Neuem. Das Neue ist nicht mit bereits existierenden Erfahrungen vergleichbar.

Je älter ich werde, desto mehr komme ich zu der Überzeugung, dass extreme Sichtweisen nicht gut sind. Die Lösung liegt fast immer in der Mitte. Vermeiden Sie Extreme und suchen Sie nach einem Mittelweg. Ich bin der Meinung, dass man keinen 100 % sicheren Plan als Ziel haben sollte. Man sollte aber auch nicht blauäugig an ein neues Projekt herangehen.

Obwohl das Management von Softwareprojekten und insbesondere die Kostenschätzung ein wichtiges Thema ist werde ich Sie in diesem Buch nicht länger damit langweilen. Der Schwerpunkt dieses Buches liegt darin, aufzuzeigen wie Softwaretests in den praktischen Arbeitsablauf bei der Entwicklung von Software integriert werden können.

Softwaretests in den Arbeitsablauf integrieren

Sie haben sich dazu entschieden Ihre Software zu testen. Schön! Wann tun Sie dies am besten? Schauen wir uns dazu die Kosten, die beim Auffinden eines Fehlers für dessen Behebung notwendig sind, an.

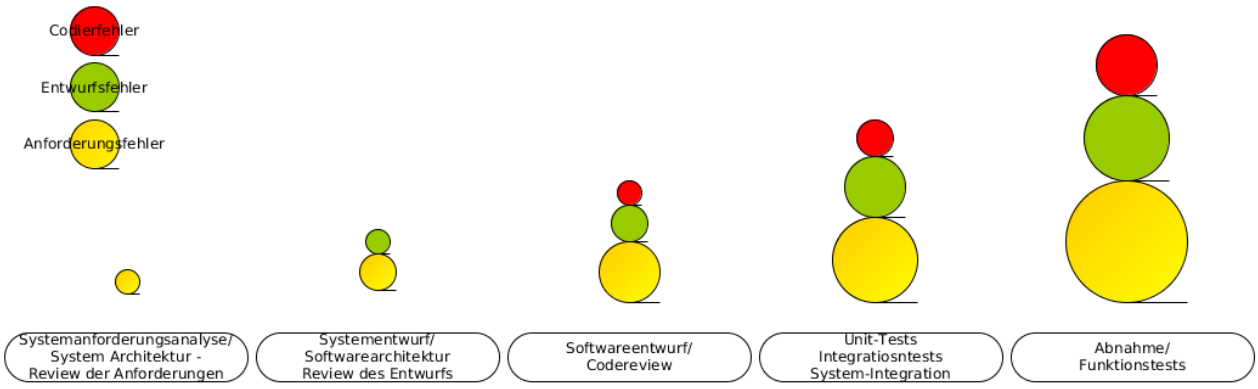
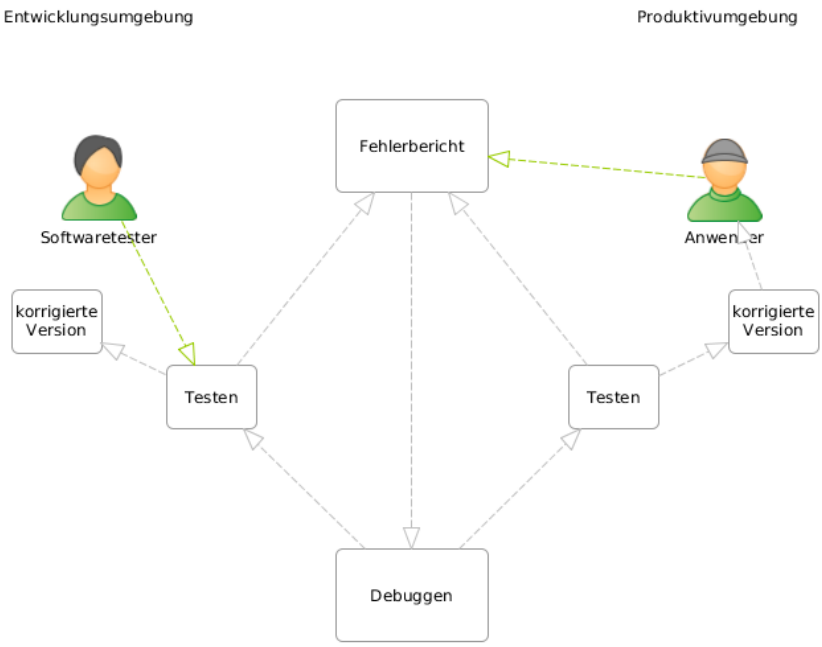


Illustration 2: Relative Kosten für die Fehlerbehebung in den Projektphasen 997.png

> **RANBEMERKUNG: Testen und Debuggen**
> Es gibt Worte die oft in einem Atemzug genannt werden und deren Bedeutung deshalb gleichgesetzt wird. Bei genauer Betrachtung stehen die Begriffe aber für unterschiedliche Auslegungen.Testen und Debuggen haben gemein, dass Sie Fehlfunktionen aufdecken. Es gibt aber auch Unterschiede in der Bedeutung.
> **Testmethoden** finden unbekannte Fehlfunktionen während der Entwicklung. Dabei ist das Finden der Fehlfunktion aufwendig und teuer, die Lokalisation und Behebung des Fehlers ist hingegen billig. Das Erkennen der Fehlfunktion ist quasi ein Nebenprodukt, dass sich aus den Testfällen ergeben hat.
> **Debugger** beheben bekannte Fehlfunktion nach Fertigstellung des Produktes. Dabei ist das Finden der Fehlfunktion gratis, die Lokalisation und Behebung des Fehlers aber teuer. Hauptaufwand: Reproduktion und Lokalisierung.



Je früher Sie einen Fehler finden, desto geringer sind die Kosten für die Fehlerkorrektur.

Kontinuierliches Testes

Kontinuierliche Integration von Tests

Stellen Sie sich folgendes Szenario vor. Die neue Version eines beliebten Content Management Systems soll veröffentlicht werden. Alles das, was die Entwickler des Teams seit dem letzten Release beigetragen haben, wir nun das erste Mal zusammen eingesetzt. Die Spannung steigt! Wird alles funktionieren? Werden alle Tests erfolgreich sein - falls das Projekt überhaupt Tests integriert. Oder muss die Freigabe

der neuen Version doch wieder verschoben werden und es stehen nervenaufreibende Stunden der Fehlerbehebung an? Ganz nebenbei ist das Verschieben des Veröffentlichungszeitpunkts auch nicht gut für das Image der Software!

Das eben beschriebene Szenario mag wohl kein Entwickler gerne miterleben. Viel besser ist es doch, jederzeit zu wissen, in welchem Zustand sich das Softwareprojekt gerade befindet? Weiterentwicklungen, die nicht zum bisherigen Bestand passen, sollten erst integriert werden, bevor diese „passend" gemacht wurden. Gerade in Zeiten, in denen es immer häufiger vorkommt, dass eine Sicherheitslücke behoben werden muss, sollte ein Projekt auch stets in der Lage sein, eine Auslieferung erstellen zu können! Und hier kommt das Schlagwort **Kontinuierliche Integration** ins Spiel.

Bei der kontinuierlichen Integration werden einzelne Bestandteile der Software permanent integriert. Die Software wird in kleinen Zyklen immer wieder erstellt und getestet. Integrationsprobleme oder fehlerhafte Tests finden Sie frühzeitig und nicht erst Tage oder Wochen später. Bei einer kontinuierliche Integration ist die Fehlerbehebung wesentlich einfacher, weil die Fehler zeitnah zur Programmierung auftauchen und in der Regel nur ein kleiner Programmteil betroffen ist.

(Todo Jenkins Kapitel 9?)

Und, damit Sie bei einer kontinuierlichen Integration auch jederzeit Tests für alle Programmteile zur Verfügung habe, sollten Sie testgetrieben entwickeln.

Testgetriebene Entwicklung (TDD)

Die Testgetriebene Entwicklung ist eine Technik, bei der in kleinen Schritten entwickelt wird. Dabei schreiben Sie als erstes den Testcode. Erst danach erstellen Sie den zu testenden Programmcode. Jede Änderung am Programm wird erst vorgenommen, nachdem der Test für diese Änderung erstellt wurde. Tests müssen also unmittelbar nach der Erstellung fehlschlagen. Die geforderte Funktion ist ja noch nicht im Programm implementiert. Nun erst erstellen Sie den Programmcode, der den Test erfüllt. Die Tests helfen Ihnen also zusätzlich dabei, das **Programm richtige** zu schreiben. Wenn Sie das erste Mal von dieser Technik hören, können Sie sich vielleicht nicht so recht mit diesem Konzept anfreunden. Mensch will doch immer erst etwas Produktives machen. Und das Schreiben von Tests wirkt auf den ersten Blick nicht wertvoll. Sehen Sie sich meine praktisches Beispiel im Kapitel Unittests an.

> **RANBEMERKUNG Regressionstest**
> Ein **Regressionstest** ist ein wiederholt durchgeführter Test. Durch die

> Wiederholung wird sicher gestellt, dass Modifikationen in bereits getesteten Teilen der Software keinen neuen Fehler oder keine neue Regression verursachen.
>
> Warum sollten Sie Regressionstests machen? Sie sollten Tests wiederholt durchführen weil bestimmte Fehlfunktionen manchmal plötzlich wieder auftauchen. Zum Beispiel
> - bei der Verwendung von Versionskontrollsoftware beim Zusammenführen mit alten defekten Versionen.
> - aufgrund von Maskierung: Fehlfunktion A tritt aufgrund der unkorrekte Programmänderung B nicht mehr auf, weil der neuer Defekt B die Fehlfunktion A maskiert. Nachdem B gefixt ist tritt Fehlfunktion A wieder auf.

Behavior-Driven-Development (BDD)

BDD ist keine weitere Programmiertechnik oder Testtechnik, sondern eine Art Best Practices für das Entwickeln von Software. BDD kommt idealerweise gemeinsam mit TDD zum Einsatz. Im Prinzip steht Behavior-Driven-Development dafür, nicht die Implementierung des Programmcodes, sondern das Verhalten des Programms zu testen. Ein Test prüfen, ob die Spezifikation, also die Anforderung des Kunden, erfüllte ist. Wenn Sie Ihre Software verhaltensgetrieben entwickeln helfen Ihnen Tests nicht nur dabei, Ihr Programm richtig zu schreiben. Sie unterstützen Sie auch dabei, das **richtige Programm** zu schreiben. Beim Behavior Driven Development werden die Anforderungen an die Software mittels Beispielen, sogenannten Szenarios beschrieben.

Merkmale des Behavior Driven Development sind

- eine starke Einbeziehung des Endnutzers in den Entstehungsprozess der Software.
- die Dokumentation aller Projektphase mit Fallbeispielen in Textform - üblicherweise in der Beschreibungssprache Gherkin.
- die Automatisierung dieser Fallbeispiele.
- eine sukzessive Implementierung.

So kann jederzeit auf eine Beschreibung der umzusetzenden Software zugegriffen werden. Mithilfe dieser Beschreibung sollte fortwährend die Korrektheit des bereits

implementierten Programmcodes möglich sein. Wie Sie Texte in der Beschreibungssprache Gherkin in Akzeptanztests verwenden können, finden Sie im Kapitel *Akzeptanztests.*

> **RANBEMERKUNG: Grundlegende Teststrategien**
> Grundsätzlich können Sie **spezifikationsorientierte** und **implementationsorientierte** Testverfahren unterscheiden. In beiden Verfahren gib es **statische** und **dynamische** Prüfverfahren. Statische Prüfverfahren prüfen das Programm ohne es auszuführen. Dynamische prüfen das Programm während es ausgeführt wird.
> Bei **spezifikationsorientierten Tests** werden die Testfälle durch Analyse der Spezifikation gewonnen. Sicherlich haben Sie schon einmal den Begriff **Black Box Tests** gehört. Unter diesem Namen ist diese Testvariante bekannter. Black Box Tests werden bewusst in Unkenntnis der Programminterna durchgeführt.
> Ein Vorteile von spezifikationsorientierten Tests sind, dass fehlende Programmteile entdeckt werden – vorausgesetzt die Spezifikation ist vollständig.
> **Implementationsorientierte Tests** gewinnen Testfälle durch strukturelle Analyse des Programms. Diese Testvariante kennen Sie vielleicht unter dem Namen **White Box-Tests**. Implementationsorientierte Tests sind auch bei fehlender Spezifikation möglich, dabei können aber vergessene Funktionen nicht entdeckt werden.

	implementationsbezogen	spezifikationsbezogen
statisch	Statische Code-Analyse	Entwurfsphase
dynamisch	Profiler	**Diese Tests sind unser Thema.**

Tests planen

Alles beginnt mit einem Plan. Sollte es zumindest. Ein Testplan sollte allen Projektbeteiligten Klarheit darüber verschaffen, was wie intensiv getestet werden soll.

Der Testplanungsprozess kann sehr komplex sein. In der ISO 29119-2 umfasst er neun umfangreiche Aktivitäten.

Es gibt aber auch andere Vorgehensweisen. Whittaker beschreibt in seinem Buch How Google Tests Software die sehr an der Praxis orientierte und leicht auf dem aktuellen Stand zu haltende Methode **Attributes-Componentes-Capabilities**, kurz ACC. ACC ordnet jeder Komponente verschiedene Attribute wie Benutzerfreundlichkeit, Geschwindigkeit oder Sicherheit zu. Diese Komponente-Attribut-Kombination wird in

einer Matrix zusammen mit einem Wert für die Wichtigkeit dieser Komponente-Attribut-Kombination aufgenommen.

Egal wie Sie Ihren Testplan erstellen. Wichtig ist meiner Meinung nach das Klarheit darüber herrscht, welche Programmbestandteile wie wichtig sind. Daraus ergibt sich dann was wie intensiv getestet werden sollte. (Todo Testabdeckung)

> **RANDBEMERKUNG Testfälle**
> Das Testen aller möglichen Eingabeparameter ist in der Realität unmöglich. Ein systematisches stichprobenartiges Testen ist die einzig praktikable Lösung!
>
> Nehmen wir an die Menge der möglichen Testfälle ist D.
> Um ein Beispiel zu nenne: Bei der Prüfung eines Textes auf ein bestimmtes Muster das in einen anderen Text umgewandelt werden soll, kann dieses Muster genau einmal, keinmal oder mehrmals im Text vorkommen.
> **D = Eingabebereich.**
> Nehmen wir weiter an, dass es eine Menge an möglichen Ausgabemöglichkeiten gibt und nennen diese R. Soll ein Muster in einem Text in einen anderen Text umgewandelt werden könnte eine Ausgabemöglichkeit der Eingabetext mit korrekt umgewandeltem Muster sein. Eine andere Ausgabemöglichkeit ist der Eingabetext mit Muster, das fehlerhaft nicht umgewandelt wurde.
> **R = Ausgabemöglichkeiten**.
> Während der Programmausführung wandelt das System die Menge D in die Menge R um. Im folgenden beschreibe ich die Programmausführung formal mit P.
> **Programmausführung P: D -> R**
> Ein Spezialfall von P liegt vor, wenn das Programm die Daten so verarbeitet, wie es in der Spezifikation festgelegt wurde. Im folgenden Nenne diesen Spezialfall F.
> **F: D -> R (Ausgabe-Anforderung für P ist erfüllt)**
> Das Programm arbeitet also formal gesehen korrekt und fehlerfrei, wenn die Menge P gleich der Menge F für alle möglichen Eingaben ist.
> $d \in D : P(d) = F(d)$
> Eine endliche Teilmenge $T \subseteq D$ ist eine Menge von Testfällen.
>
> **Der ideale Test:** Wenn ein Programm an einer Stelle nicht korrekt ist, dann

gibt es einen Testfall der dieses unkorrekte Verhalten erzeugt. Ideal ist dieser Testfall, wenn man ihn findet, ohne alle möglichen Testfälle durchprobieren zu müssen. Den idealen Test t in einem fehlerhaften Programm P(d) ≠ F(d) findet man, indem man P(t) ≠ F(t) bestimmt.
∃d∈D: P(d) ≠ F(d)
∃t∈T: P(t) ≠ F(t)

Die gute Nachricht: Für jedes Programm gibt es einen idealen Test, der sogar nur einen Testfall enthält.
- Wenn die Programmausführung fehlerhaft ist gilt: ∃d∈D: P(d) ≠ F(d) - Für den idealen Test gilt T={d}
- Wenn die Programmausführung fehlerfrei ist gilt: T={}- Die Menge der idealen Tests ist leer. Es gibt keinen Fehler.

Leider ist die Menge T der idealen Tests nicht einfach zu bestimmen. Wenn das so wäre gäbe es sicherlich ausschließlich korrekte Programme. Das der ideale Test nicht berechenbar ist hat Howden schon 1977 bewiesen.

Der gleichförmig ideale Test: Ein idealer Test gilt für ein konkretes Programm mit einer konkreten Fehlfunktionen. Ein gleichförmig idealer Test findet Fehlfunktionen in allen Programmen P die die Ausgabe-Anforderung F berechnen. Nun ist es aber so, dass es für jeden Testfall d ein Programms P gibt, für das d nicht die korrekte Ausgabe berechnet. ∀d∈D gibt es P, das nur für d falsch ist. Dies hat zur Folge, dass nur der erschöpfende Test gleichförmig ideal ist. Und das heißt wiederum, dass nur ein Test bei dem die Menge der Testfälle gleich der möglichen Eingaben ist, also T = D gilt, ein erschöpfend idealer Test ist.

Was sollten Sie beim generieren von Tests beachten?

Sie haben nun sehr viel Theorie zum Thema Softwaretests gelesen. In diesem Buch werde ich ihnen einige Werkzeuge erklären, die Sie in der Praxis beim Generieren von Tests unterstützen. Wenn Sie diese Tools einsetzten, werden Sie selbst erfahren, wie Sie Ihre Tests am besten schreiben. Da jeder Entwickler individuelle Vorgehensweise hat, gibt es viele Dinge, die man nicht allgemein als Regel mitgeben kann. Es gibt aber drei Regeln, die sich allgemein durchgesetzt haben:

1. Ein Test sollte **wiederholbar** sein.
2. Ein Test sollte **einfach** gehalten sein.
3. Ein Test sollte **unabhängig** von anderen Tests sein.

> RANDBEMERKUNG
> Mit Docker können Sie Tests unabhängig voneinander ablaufen lassen. Eine kurze Beschreibung dazu, wie Sie Docker mit Codeception zusammen nutzen können, finden Sie in der Dokumentation von Codeception.

Kurzgefasst

Das 1. Kapitel beschreibt, was Softwaretest sind und warum Software Tests wichtig sind. Dann plane ich verschiedene Konzepte zu erklären. Unter anderem die Begriffe Test-Driven-Development und Behavior-Driven-Development. Ich möchte auch darauf eingehen, wie Tests kontinuierlich integriert werden können/sollen.

Außerdem möchte ich hier wichtige Prinzipien wie Einfachheit, Wiederholbarkeit und Unabhängigkeit eines Tests erläutern.

Praxisteil: Die Testumgebung einrichten

> Program testing can be used to show the presence of bugs, but never show their absence!
> [Edsger W. Dijkstra]

(todo Einleitung)

In diesem Kapitel arbeiten wir endlich praktisch. Da dieses Buch die Anwendung von Software zum Thema hat, ist es nicht dazu geeignet auf dem Sofa oder am Strand gelesen zu werden. Das Erlernen von Software funktioniert meiner Meinung nach am besten, wenn alle Beispiele am Computer selbst nachvollzogen werden. Am Ende dieses Kapitels werden Sie Joomal! und eine erste kleine eigene Erweiterung auf Ihrem lokalen Webserver installiert haben. Diese Erweiterung wird dann auch die Grundlage für den Aufbau der Beispeiltests sein.

Joomla! ist ein Content Management System, mit dem Sie eine Website erstellen und deren Inhalte mithilfe eines WYSIWYG Editors pflegen können. Die Erweiterung, die

wir beispielhaft erstellen, soll einen Benutzer dabei unterstützen einen Jetzt-kaufen-Button von Paypal in einen Beitrag zu integrieren. Der Benutzer muss hierzu nur ein bestimmtes Textkürzel kennen und dies in einen Beitrag einfügen. Wenn dieser Beitrag dann von Joomla! für die Anzeige auf der Internetseite präpariert wird, kommt die Erweiterung zum Einsatz. Sobald diese während der Beitragserstellung auf das definierte Textkürzel stößt, wandelt sie dieses in ein HTML-Element um, welches einen Jetzt-kaufen-Button anzeigt.

Entwicklungsumgebung und Arbeitsweise

Entwickeln Sie bereits Software? Dann arbeiten Sie sicherlich in Ihrer persönlichen Entwicklungsumgebung, in der Sie sich sicher und wohl fühlen. Falls dies nicht so ist, sollten Sie sich diese Entwicklungsumgebung aufbauen. Ein Computer, auf dem Werkzeuge installiert sind, die Sie bei der Arbeit unterstützen ist eine Voraussetzung für die Erstellung guter Software. Ohne eine solche Umgebung werden Sie sicherlich keinen Spaß an Ihrer Arbeit haben.

Als Leitfaden beschreiben ich Ihnen hier kurz die wichtigsten Teile meiner Entwicklungsumgebung.

- Ich arbeite mit dem Betriebssystem Ubuntu. Aktuell verwende ich die Version 16.04 LTS
- Zum Bearbeiten des Programmcodes verwende ich die integrierte Entwicklungsumgebung (IDE) Netbeans. Theoretisch können Sie einen einfachen Texteditor verwenden. Eine IDE bietet Ihnen jedoch eine Menge mehr Komfort. Mit Netbeans können Sie beispielsweise Ihre Software Debuggen, Programmcode automatisch vervollständigen oder Werkzeuge, die eine Versionskontrolle unterstützen, nutzen.
- Um die Installation und Konfigurieren des Webservers Apache mit der Datenbank MySQL und der Skriptsprachen PHP so einfach wie möglich zu machen verwende ich die Programmkombination LAMP. LAMP ist ein Akronym. Die einzelnen Buchstaben des Akronyms stehen für die verwendeten Komponenten Linux, Apache, MySQL und PHP. Eine Anleitung, die die Installation von LAMP unter Ubuntu beschreibt, finden Sie unter der Adresse https://wiki.ubuntuusers.de/LAMP/ im Internet.

Sie können natürlich die Beispiele im Buch auch mit alternativer Software durchführen. In diesem Fall kann es sein, dass etwas nicht so wie beschrieben läuft und Sie selbst Anpassungen vornehmen müssen.

Joomla! herunterladen und auf einem Webserver installieren

Sie werden feststellen, dass die Installation von Joomla! sehr einfach und intuitiv ist. Zumindest dann, wenn alle Systemvoraussetzungen erfüllt sind.

Ich beschreibe hier die Installation unter Ubuntu Linux inclusive einer Standard LAMP Installation. Falls Sie mit einem anderen Betriebssystem arbeiten passen Sie die Beschreibung bitte an Ihre Systemumgebung an.

Was ist Joomla! Eigentlich genau?

Bevor Sie Joomla! installieren möchten Sie sicherlich erfahren, worauf Sie sich mit dieser Installation möglicherweise einlassen? Joomla! Ist ein Content Management System, kurz CMS, mit dem Sie nicht nur eine Website erstellen und pflegen können. Sie können mit Joomla! leistungsstarke Webanwendungen programmieren.

Wenn Sie Joomla! nutzen möchten, müssen Sie kein Geld dafür zahlen. Außerdem können Sie den Quellcode einsehen. Joomla! ist eine Open Source Software die unter der Lizenz GNU General Public License Version 2 or later veröffentlicht ist.
GNU ist die am weitesten verbreitetste Softwarelizenz. Diese erlaubt Ihnen, Joomla! auszuführen, für Lernzwecke zu verwenden, zu ändern und zu kopieren. Software, die diese Freiheitsrechte gewährt, wird Freie Software genannt. Joomla! unterliegt weiterhin einem Copyleft. Das heißt, Sie müssen die eben beschriebenen Rechte bei Weitergabe beibehalten. Auch dann, wenn Sie Ihrer Meinung nach verbessernde Änderungen an Joomla! Vorgenommen haben. Sie dürfen für Ihre Verbesserungen aber Geld berechnen. Freie Software kann - wie im Falle von Joomla! - kostenlos sein. Freie Software muss aber nicht kostenlos sein!

Voraussetzungen

Joomla!s Anforderungen bezüglich PHP-Version, unterstützter Datenbanken und unterstützter Web-Server sind nicht sehr hoch. Wahrscheinlich werden Sie keine Probleme haben. Da sich die Mindesterfordernisse von Version zu Version ändern nenne ich Ihnen hier nur einen Link. Die aktuellen Systemvoraussetzungen können Sie unter der Adresse https://downloads.joomla.org/de/technical-requirements-de einsehen.

Download des Joomla! Installationspaketes

Besorgen sie sich als erstes das aktuelle Joomla! Installationspaket. Die neueste Version findest Sie immer unter der Adresse https://www.joomla.org/download.html. Eine Installationsdatei, die die deutschen Sprachpakete enthält finden Sie auf der Website https://www.jgerman.de/. Da ich ein deutsches Buch schreibe habe ich das

deutsche Installationspaket in der Version 3.6.5 - also die Datei Joomla_3.6.5-Stable-Full_Package_German.zip - installiert. Sie werden wahrscheinlich eine aktuellere Version herunterladen können.

Upload der Joomla! Installationsdateien auf den lokalen Webserver

Verschieben Sie das heruntergeladene Installationspaket auf Ihren lokalen Webserver und entpacke es dort. Wenn Sie wie ich die Standardinstallation von LAMP nutzen, sollten Sie das Installationspaket also in das Verzeichnis /var/www/html kopieren. Nach dem Entpacken sehen Sie dann das Unterverzeichnis /var/www/html/**Joomla_3.6.5-Stable-Full_Package_German.** Der Einfachheit halber benennen Sie dieses Verzeichnis bitte in /var/www/html/**joomla** um.

Ab nun können Sie Joomla! in Ihrem Internetbrowser über die URL http://localhost/joomla/ aufrufen. Probieren Sie es aus! Wenn alles richtig läuft werden Sie mit der Hauptkonfigurationsseite begrüße und können sofort mit dem nächsten Kapitel fortfahren.

Hauptkonfiguration

Abbildung 3: Joomal! Hauptkonfiguration 992

Der weitere Ablauf der Konfiguration ist meiner Meinung nach sehr intuitiv und selbsterklärend. Falls Sie doch Fragen haben, hilft Ihnen vielleicht die ausführlichere Installationsanleitung in der Joomla! eigenen Dokumentation weiter. Gerne werden auch Fragen im deutschen Joomla Forum beantwortet.

Damit wir gleiche Voraussetzungen haben wäre es gut, wenn Sie im letzten Schritt der Installation auf Beispieldaten verzichten.

Abbildung 4: Zusammenfassung der Joomal! Installation - Hier bitte auf Beispieldaten verzichten 991.png

Ich bin mir sicher, dass Sie Joomla! erfolgreich installiert und konfiguriert haben und den Administrationsbereich in Ihrem Browser nun über die Adresse http://localhost/joomla/administrator/ und das Frontend in Ihrem Browser über die Adresse http://localhost/joomla/ aufrufen können.

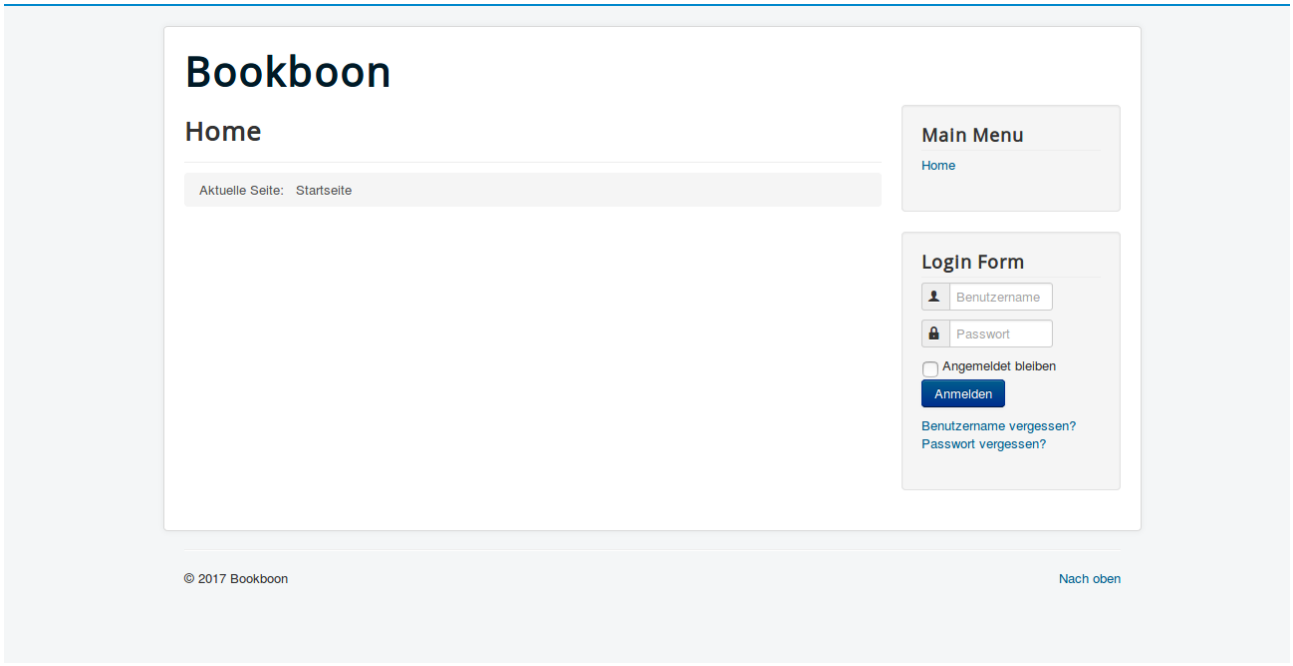


Abbildung 5: Joomla! Frontend unmittelbar nach der Installation – erreichbar über die Adresse http://localhost/joomla/ 988.png

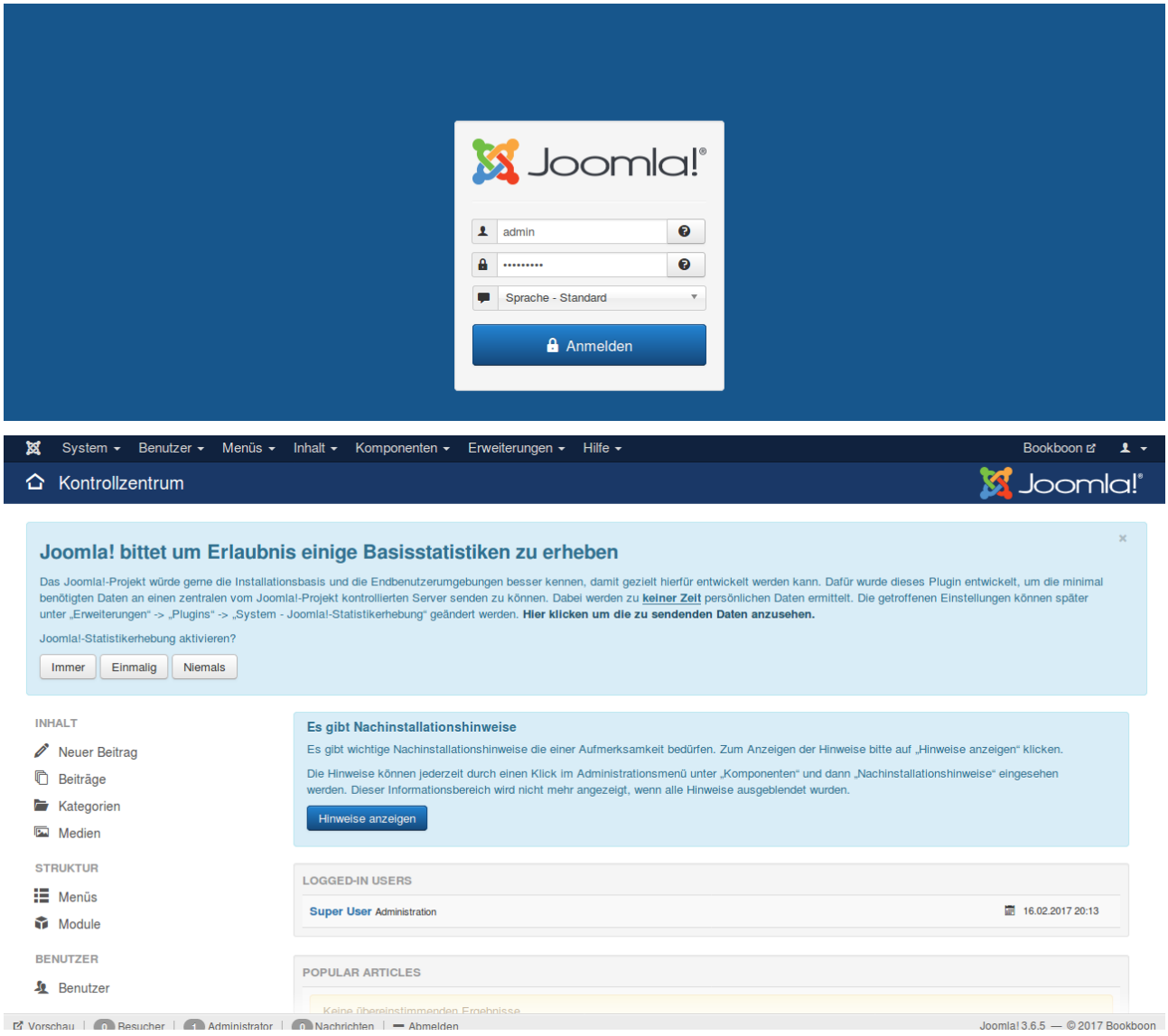


Abbildung 7: Joomla!: Die erste Anmeldung im Administrationsbereich 987.png

In Ihrem Dateisystem finden Sie die Joomla!-Dateien im Verzeichnis /var/www/htm/joomla. Genau finden Sie hier folgende Verzeichnisstruktur vor:

```
/var/www/html/joomla$
- administrator
- bin
- cache
- cli
- components
- images
- includes
- language
- layouts
- libraries
- media
- modules
- plugins
- templates
- tmp
 LICENSE.txt
 README.txt
 configuration.php
 htaccess.txt
 index.php
 robots.txt
 web.config.txt
```

RANDBEMERKUNG

Bitte haben Sie im weiteren Verlauf des Buches immer im Hinterkopf, dass Joomla! ein aktives und lebendes Projekt ist. Es wird ständig weiterentwickelt und verbessert. Das ist auch sehr gut so. Nachteilig ist nur, dass ich nicht sicherstellen kann, dass in allen zukünftigen Versionen alles genauso ist, wie ich es Ihnen hier erkläre. Sie sollten aber trotzdem immer die neueste Version von Joomla! verwenden – schon alleine aus Sicherheitsgründen.

Die Joomla! Architektur verstehen

Wie jedes System besteht Joomla! aus mehreren Elementen. Und wie jedes System ist es mehr als die Summe der einzelnen Elemente!

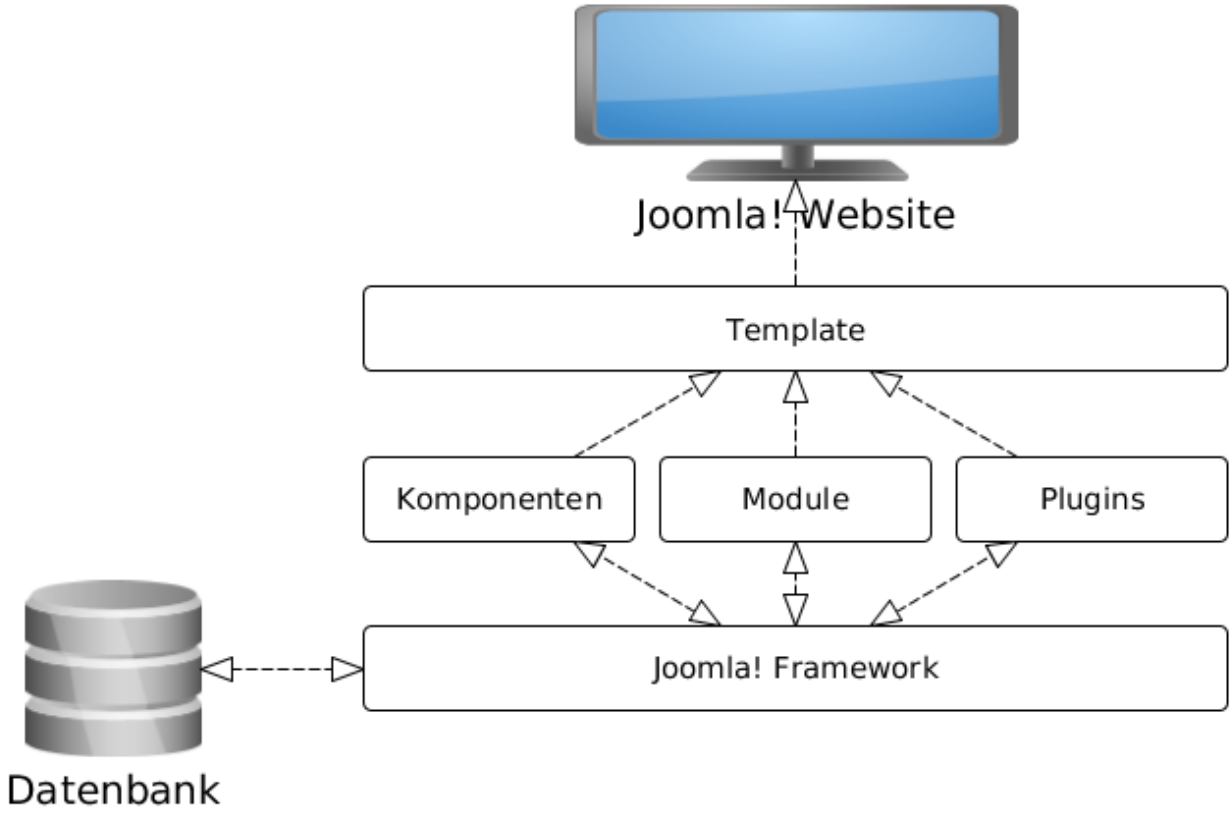


Abbildung 8: Joomla! Architektur 996

Joomla! ist ein modular aufgebautes System, das bereits in der Basisinstallation zahlreiche, nützliche Erweiterungen mitbringt.

Datenbank

Alle Inhalte Ihrer Joomla! Website, mit Ausnahme von Bildern und Dateien im Unterverzeichnis /images, werden in einer Datenbank gespeichert.

Joomla! Framework

Das Joomla! Framework ist eine Sammlung von Open Source Software auf der das Joomla! CMS aufbauen.

Erweiterungen

Erweiterungen erweitern, wie der Name schon sagt, die Basis von Joomla!, also das Joomla! Framework. Sie können zwei Arten und vier Typen von Erweiterungen unterscheiden:

Erweiterungsarten

Joomla! unterscheiden zum einen die

- geschützten Erweiterungen.

  Das sind Erweiterungen, die Sie bei der Standardinstallation von Joomla! mit installierst haben. Joomla! Bringt schon in der Basisinstallation zahlreiche, nützliche Erweiterungen mit.

- Erweiterungen von Drittanbietern.

  Sollte die Basisinstallation für Ihren Bedarf nicht ausreichen, lässt sich das System um beinahe jede erdenkliche Funktion ergänzen. In einem offiziellen Verzeichnis stehen Ihnen zahlreiche Erweiterungen zur Verfügung.

Erweiterungstypen

Joomla! unterscheidet verschiedene Arten von Erweiterungen.

- Komponenten

  Unter einer Komponente können Sie sich eine kleine Anwendung vorstellen. Diese Anwendung erfordert das Joomla! Framework als Grundlage, ansonsten können Sie diese aber eigenständig nutzen und mit ihr interagieren. Ein Beispiel für eine geschützte Komponente ist der Benutzermanager.

- Module

  Ein Modul ist weniger komplex als eine Komponente. Es stellt keinen eigenständigen Inhalt dar, sondern wird beim Aufbau der Seite auf einer festgelegten Position angezeigt. Mit dem Modulmanager, der ein Beispiel für eine geschützte Komponente ist, konfigurieren Sie ein Modul. Das wohl bekannteste Modul ist Eigenes HTML, mit dem Sie individuelle Texte mittels der Hypertext-Auszeichnungssprache HTML auf Ihrer Website anzeigen können.

- Plugins

  Plugins sind relativ kleine Programmcode Teile, die bei Auslösung eines bestimmten Ereignisses ausgeführt werden. Ein Beispiel für ein Ereignis ist die erfolgreiche Anmeldung eines Benutzers. Im Kapitel *Joomla! mit einem eigenen Plugin erweitern* werden wir das einfache Plugin *Agpaypal* schreiben. Das Ereignis, das wir in diesem Plugin ausnutzen werden, ist onContentPrepare. Genau beschreibt das Ereignis onContentPrepare den erste Schritt in der Aufbereitung der Anzeige eines Beitrags im Frontend. Ein Plugin ist eine einfache aber sehr effektive Art das Joomla! Framework zu erweitern.

- Templates

  Das Template bestimmt das Aussehen Ihrer Joomla! Website.

Joomla! mit einem eigenen Plugin erweitern

Zu Beginn dieses Kapitels haben Sie Joomla! installiert. Danach haben ich Ihnen die wichtigsten Bestandteile von Joomla! erläutert. In diesem Abschnitt werden wir nun eine einfache Erweiterung schreiben. Genau erstellen wir ein Content Plugin. Aufgabe dieser Erweiterung ist es, einen bestimmten Text in einen Jetzt-kaufen-Button umzuwandeln. Wir werden dazu das Ereignis onContentPrepare nutzen. Dieses Ereignis wird in Joomla! beim Vorbereiten eines Beitrags für die Anzeige im Browser ausgelöst.

Dieses Pluginin soll dann im weiteren Verlauf die Grundlage für unsere Tests sein.

Ein Joomla! Plugin muss im Grunde genommen nur aus zwei Dateien bestehen. Der XML-Installationsdatei oder Manifest Datei und dem eigentlichen Programmcode.

Die Dateien müssen in einem bestimmten Verzeichnis abgelegt sein. Plugins, die Inhalte von Beiträgen manipulieren, gehören in ein Unterverzeichnis des Verzeichnisses plugins\content. Legen Sie also als erstes im Verzeichnis plugins\content den Ordner agpaypal an. In diesem Ordner erstellen Sie als nächstes die Datei agpaypal.php. In meiner Entwicklungsumgebung lege ich also konkret die Datei /var/www/html/joomla/plugins/content/agpaypal/agpaypal.php mit folgendem Inhalt an.

```
<?php
defined('_JEXEC') or die;
class plgContentAgpaypal extends JPlugin {
        public function onContentPrepare($context, &$row, $params, $page = 0){
                $search = "@paypalpaypal@";
                $replace = '<form name="_xclick" action="https://www.paypal.com/de/cgi-
                        bin/webscr" method="post">
                        <input type="hidden" name="cmd" value="_xclick">
                        <input type="hidden" name="business" value="me@mybusiness.com">
                        <input type="hidden" name="currency_code" value="EUR">
                        <input type="hidden" name="item_name" value="Teddybär">
                        <input type="hidden" name="amount" value="12.99">
                        <input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-
                        but01.gif" border="0" name="submit" alt="Zahlen Sie mit PayPal – schnell,
                        kostenlos und sicher!">
                        </form>';
```

```
                if (is_object($row)){
                        $row->text = str_replace($search, $replace, $row->text);
                }
                else{
                        $row = str_replace($search, $replace, $row);
                }
                return true;
        }
}
```

Was tut unsere Plugin genau? Zunächst einmal werden die Variablen $search und $replace erstellt. Die Variable $search wird mit einem Suchtext belegt und der Variablen $replace wird ein Text, der in einem HTML-Dokument in einen Jetzt-kaufen-Button umgewandelt wird, zugeordnet. Als nächstes ist es wichtig, dass Sie wissen, dass die Variable $row den Text des Beitrags, der gerade von Joomla! angezeigt werden soll, enthält. Die Variabel $row ist entweder ein Objekt, dass den Beitragstext in der Eigenschaft text enthält oder eine einfache Variable. In jedem Fall ersetzten wir mithilfe der Funktion str_replace() den Suchtext mit dem Text für die Anzeige des Jetzt-kaufen-Buttons.

So, die Datei die die eigentlich Arbeit erledigt ist fertig. Nun erstellen Sie im gleichen Verzeichnis die Manifest-Datei agpaypal.xml. Diese Datei ist für die Installation, also das Bekanntmachen des Plugins in Joomla!, wichtig.

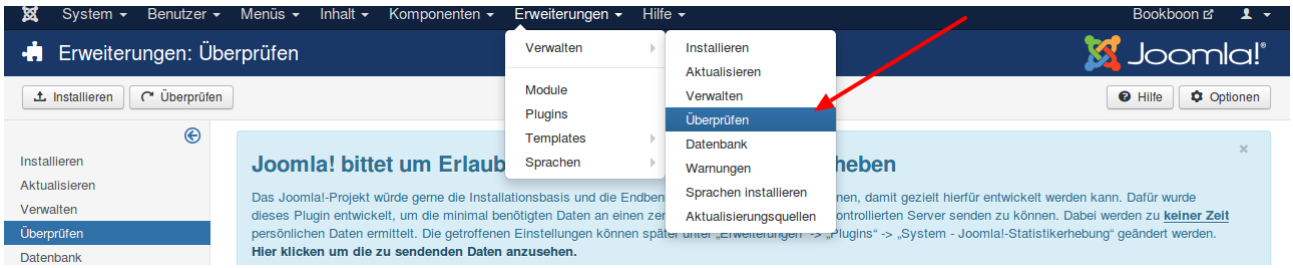
```
<?xml version="1.0" encoding="utf-8"?>
<extension version="3.6" type="plugin" group="content">
 <name>Paypal Schaltläche</name>
 <creationDate>[DATE]</creationDate>
 <author>[AUTHOR]</author>
 <authorEmail>[AUTHOR_EMAIL]</authorEmail>
 <authorUrl>[AUTHOR_URL]</authorUrl>
 <copyright>[COPYRIGHT]</copyright>
 <license>GNU General Public License version 2 or later; see LICENSE.txt</license>
 <version>1.0</version>
 <description>Das Plugin erzeugt eine Paypal "Kaufe jetzt" Schaltfläche.</description>
 <files>
        <filename plugin="agpaypal">agpaypal.php</filename>
```

```
</files>
</extension>
```

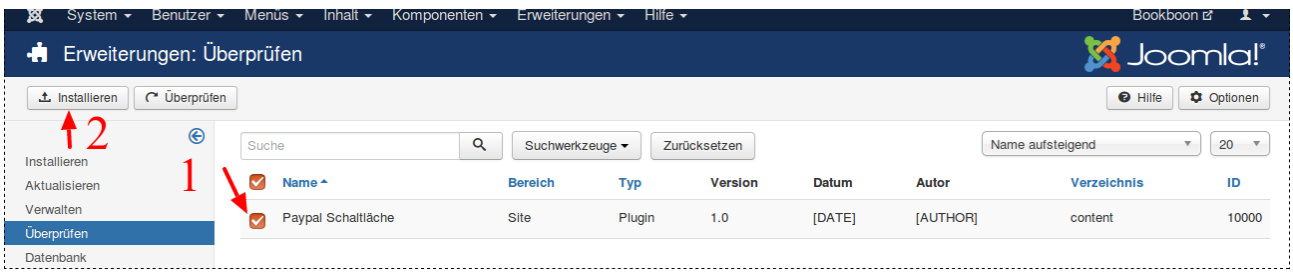
Dies ist kein Lehrbuch zum Thema Plugin-Programmierung für Joomla!. Deshalb hier nur das Wesentliche ganz kurz. Wichtig ist, dass die XML-Datei genauso heißt wie die PHP-Datei. Am Anfang der XML_Datei steht immer das XML-Tag <?xml version="1.0" encoding="utf-8"?>. In der zweiten Zeile <extension version="3.6" type="plugin" group="content"> geben Sie wichtige Informationen zur Erweiterung an. In unserem Fall handelt sich um ein Content Plugin für die Joomla! Version 3.6. Im Anschluss können Sie optional weitere Informationen zu Ihrem Plugin angeben. Zuletzt geben Sie im Tag <files> alle Dateien die zum Plugin gehören an. Das ist in unserem Fall einfach. Unser Plugin besteht bisher nur aus einer einzigen Datei.

Als nächste werden wir die Erweiterung in Joomal! ausprobieren. Wenn wir sicher sind, dass alles klappt werden wir die Erweiterung testgetrieben weiter bearbeiten. Irgendwann wollen Sie sicherlich einmal etwas anderes als einen Teddy für 12,99 Euro verkaufen, oder?

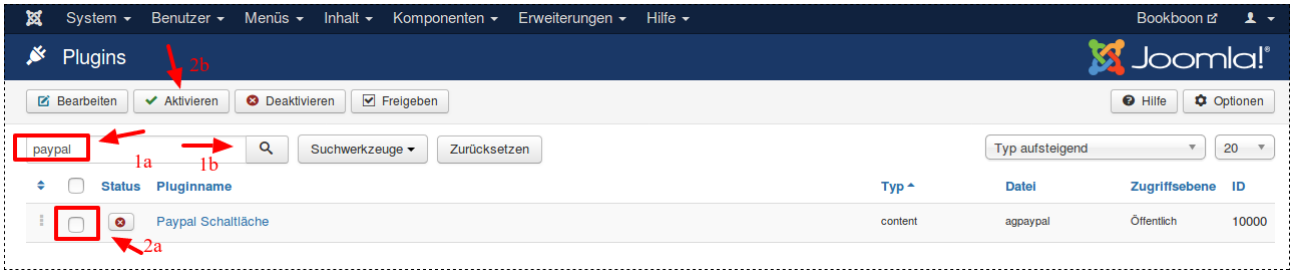
Damit Joomla! Ihre Erweiterung kennenlernt, muss das Content Management System diese noch entdecken. Öffnen Sie dazu bitte im Administrationsbereich das Menü Erweiterungen|Verwalten|Überprüfen und klicken dann links oben auf die Schaltfläche Überprüfen.



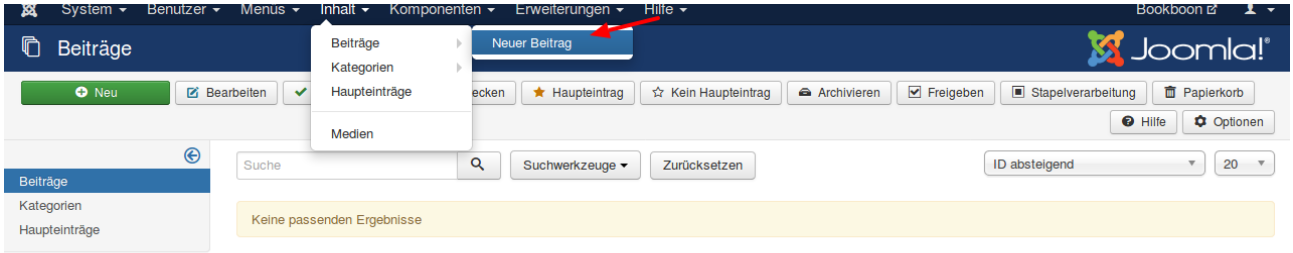
Wenn Sie die Plugin Dateien, genauso wie ich es Ihnen beschrieben habe, erstellt haben sehen Sie nun im Hauptbereich einen Eintrag, der Ihr eben erstelltes Plugin beschreibt. Wählen Sie diesen Eintrag aus und klicken danach links oben auf die Schaltfläche Installieren.



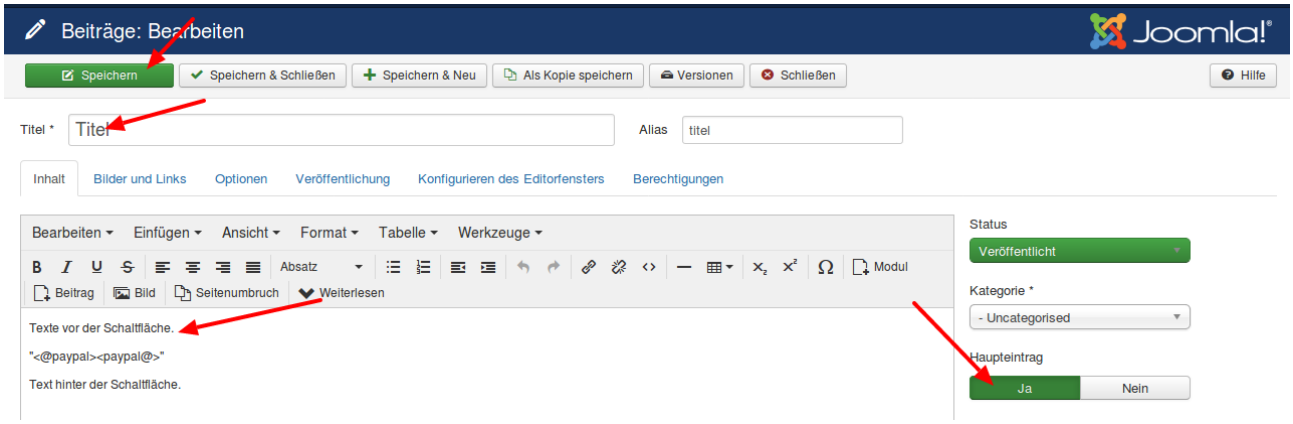
Wenn alles richtig läuft wird Ihnen nun gemeldet, dass das Installationspaket installiert wurde. Überprüfen Sie über das Menü Erweiterungen|Plugins ob Joomla! Ihr Plugin nun wirklich kennt und aktivieren Sie es im nächsten Schritt, indem Sie die Checkbox vor dem Plugin Eintrag selektieren und in der Werkzeugleiste auf die Schaltfläche Aktivieren klicken.



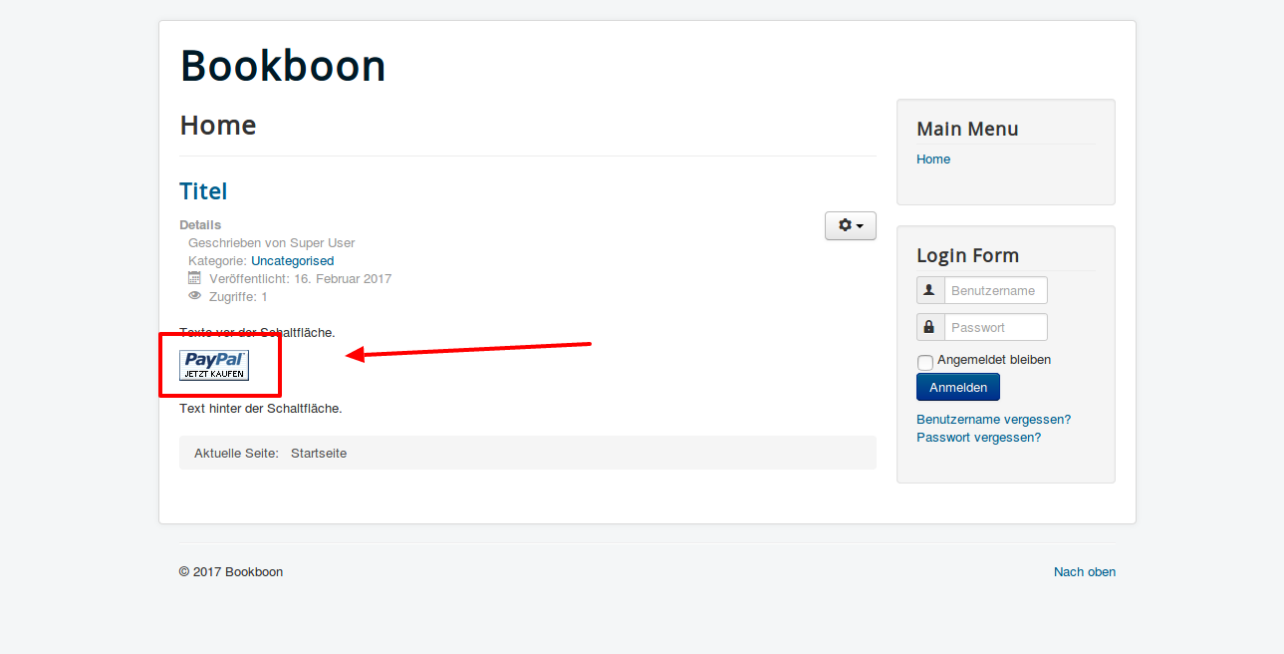
Ihr Plugin ist nun aktiv und wenn Sie einen neuen Beitrag mit dem Text "@paypalpaypal@" erstellen, erscheint im Frontend anstelle des Textes "@paypalpaypal@" ein Jetzt-kaufen-Button. Probieren Sie es aus. Erstellen Sie als erstes einen Beitrag, indem Sie im Administrationsbereich das Menü Inhalt|Beiträge|Neuer Beitrag öffnen.



Geben Sie hier nun einen Text ein, der das Muster "@paypalpaypal@" enthält. Geben dem Beitrag einen Titel und setzen Sie den Parameter Haupteintrag auf Ja, damit der Beitrag als Haupteintrag auf der Startseite angezeigt wird. Zuguterletzt speichern Sie den Beitrag.



Rufen Sie im Browser nun das Frontend auf. Sehen Sie den Jetzt-kaufen-Button? Ich denke ja. Falls nicht sollten Sie Ihr Plugin noch einmal mit der von mir beschriebenen Datei vergleichen.



So die Erweiterung funktioniert. Da wir testgetrieben entwickeln wollen, installieren wir als nächstes Codeception. Dies tun wir im nächsten Kapitel. Danach sehen wir uns die Testmöglichkeiten mit Codeception und die Philosophie die dahinter steckt kurz theoretisch an.

Unsere Tests mit Codeception planen

Die Spezifikation für das Plugin, das wir erstellen wollen, ist definiert. Ein Textmuster soll in einen Jetzt-kaufen-Button umgewandelt werden. Zusätzlich sollen bestimmte Parameter variabel im Suchtext mitgegeben werden können. Wie integrieren wir nun am besten welche Tests? Was bietet Codeception uns für Möglichkeiten?

Testtypen

Codeception unterstützt Sie beim Erstellen von

- Unittests

  Ein Unittest ist ein Test, der kleinste Programmeinheiten unabhängig voneinander testet.

- Integrationstests

  Ein Integrationstests ist ein Test, der das Zusammenspiel der einzelnen

Einheiten testet. In der Codeception Terminologie heißt dieser Test Funktionstest.

- Akzeptanztests
  Ein Akzeptanztest ist ein Test, der überprüft ob das Programm seine, zu Beginn festgelegte, Aufgabe erfüllt.

Die Bausteine des Testsystems

Da zu Beginn eines Softwareprojektes noch nicht sicher ist, wie das Programm am Ende genau aussieht, fällt das Planen von Tests schwer. Man tappt sozusagen im Dunkeln. Sinnvoll ist es, die Aufgabenstellung in einzelne, unabhängige Baustein zu unterteilen.

Für das Plugin, dass wir in diesem Buch als Beispieltestobjekt verwenden, könnten wir folgende Bausteine unterscheiden:

- Das Content Management System Joomla!, das überwiegend nach dem Entwurfsmuster Model View Controller, kurz MVC, aufbaut ist.
- Unser Plugin als kleinste Einheit innerhalb der Programmsteuerung – also innerhalb eines Controllers.
- Die Schnittstelle zum Internetbrowser, über den der Benutzer mit dem System agiert.
- Die Datenbank.
- Der Internetbrowser, über den der Benutzer die Webanwendung Joomla! aufruft.

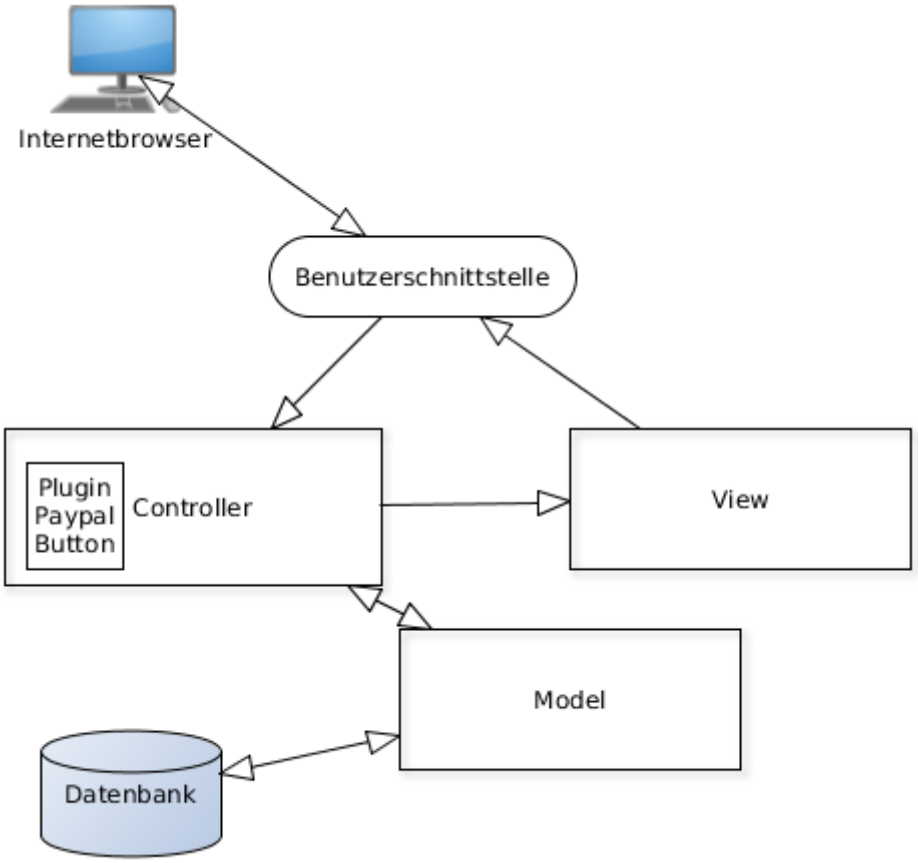


Abbildung 9: Eine Webanwendung mit Datenbank und Browerausgabe 990.png

Unittests testen einen Baustein

Ein Unittest testet kleinste Programmeinheiten unabhängig voneinander. In unserem Beispiel könnte ein Unittest die Klasse des Plugin als Gegenstand haben. Abhängigkeiten zu anderen Programmteilen müssen dabei aufgelöst werden. Zum Beispiel prüfen wir beim Testen nicht, ob der Plugin Klasse auch der richtige Text zur Umwandlung übergeben wird – also ob die anderen Klassen im System richtig arbeiten. Hier setzten wir einfach ein Testduplikat ein. Mit Testduplikat meine ich in diesem Fall einen extra für den Test erstellten Text.

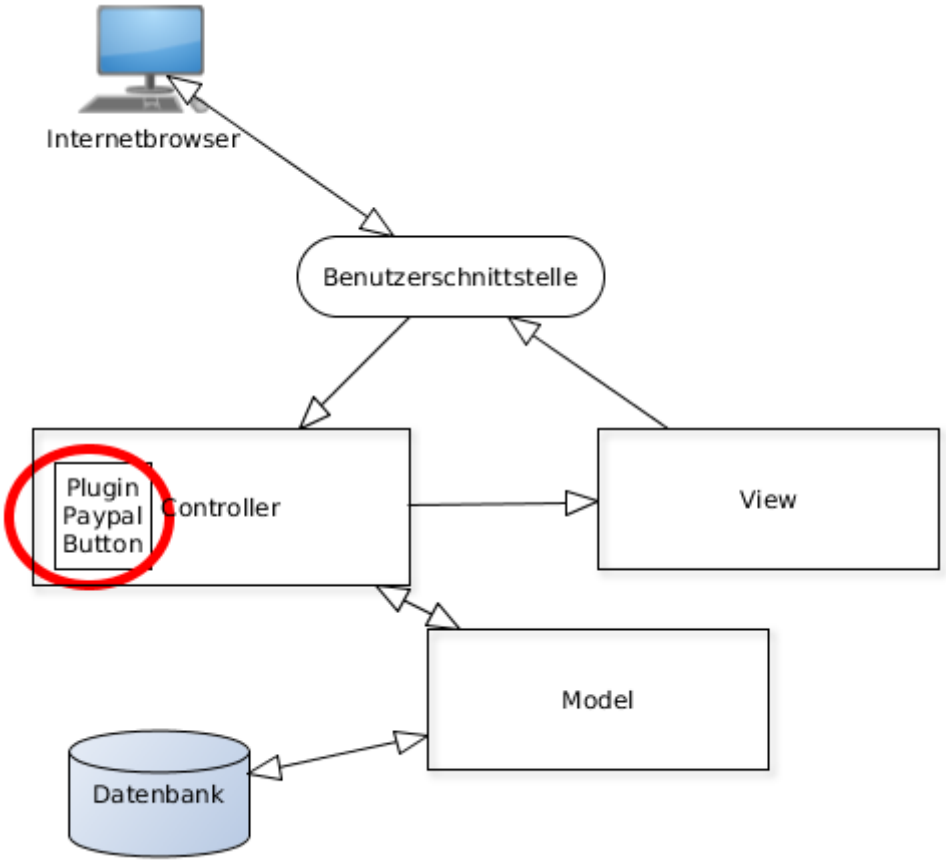


Abbildung 10: Unittest - Eine kleine Einheit innerhalb eines Systems testen. 990aUnittest.png

Integrationstests testen das Zusammenspiel der Bausteine

Ein Integrationstests testet das Zusammenspiel der einzelnen Bausteine. In unserem Beispiel könnte ein Integrationstest das gesamte System zum Gegenstand haben. Wobei der Browser, mit dem der Endbenutzer arbeitet, ausgenommen ist. Um sicher zu stellen, dass unsere Webanwendung richtig arbeitet reicht es aus, dass alle Daten richtig an die Schnittstelle zum Browser übergeben werden. Den Browser selbst testen wir mit einem Integrationstest nicht. Er gehört nicht zu unserer Anwendung.

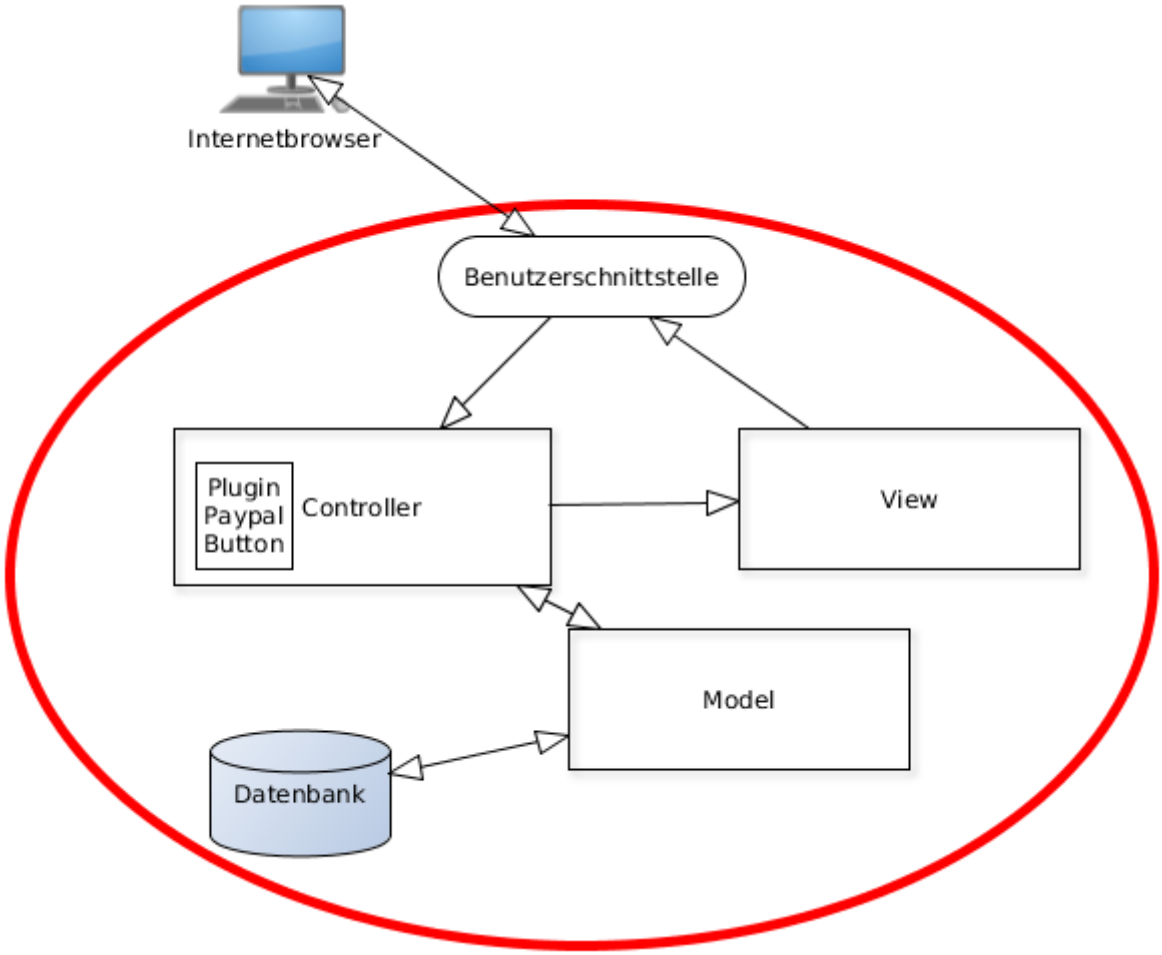


Abbildung 11: Funktionstests oder Integrationstests; Das Zusammenspiel der einzelnen Einheiten testen. Ein Anwender wird nicht simuliert.

Akzeptanztests testen das System mit wirklichen Anwendungsfällen

Ein Akzeptanztest testet, ob das Programm seine zu Beginn festgelegte, Aufgabe erfüllt. In unserem Beispiel werden nun also alle Bausteine mit in den Test einbezogen. Wir testen nicht nur, ob die Umwandlung unseres Suchtextes in einen Jetzt-kaufen-Button korrekt erfolgt. Bei einem Akzeptanztest ist es auch wichtig, dass der Button vom Benutzer wie gewollt, höchstwahrscheinlich zur Bezahlung einer Leistung, verwendet werden kann.

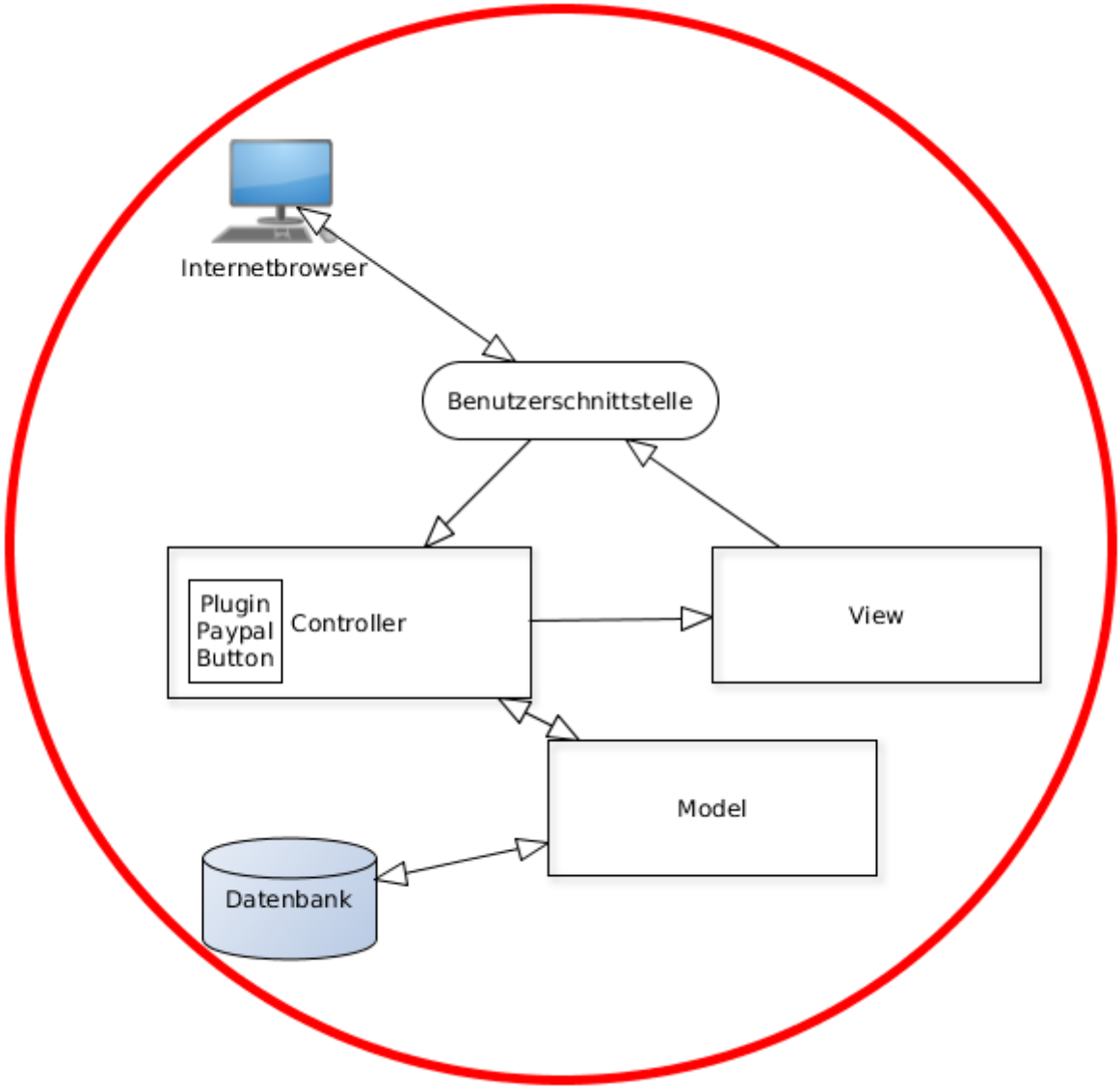


Abbildung 12: Akzeptanztests; Anwendungsfälle werden anhand von (automatisierten) Benutzereingaben getestet.

Teststrategie

Top-Down-Testen und Bottom-Up-Testen

An dieser Stelle ist es meiner Meinung nach wichtig, sich noch einmal in Erinnerung zu rufen, dass ein Element des Behavior Driven Development eine Beschreibung des Verhaltens der Software in Textform ist. Diese Beschreibung der Akzeptanzkriterien erleichtert gleichzeitig die Erstellung von Tests – insbesondere das Erstellen der Akzeptanztests auf der höchsten Ebene.

Die heute übliche Vorgehensweise beim Erstellen von Tests ist *von unten nach oben* - beziehungsweise von *innen nach außen*. Je weiter sich die verhaltensgetriebene Softwareentwicklung durchsetzt, desto mehr kommt diese Strategie in Wanken.

Immer mehr Entwickler vertreten die Sichtweise, dass von *oben nach unten* - beziehungsweise von *außen nach innen* getestet werden sollte. Denn, nur bei einer Top-Down Strategie ist einer falschen Annahme bei Auslegung einer Benutzeranforderungen schon zu Beginn des Projektes schnell der Gar ausgemacht.

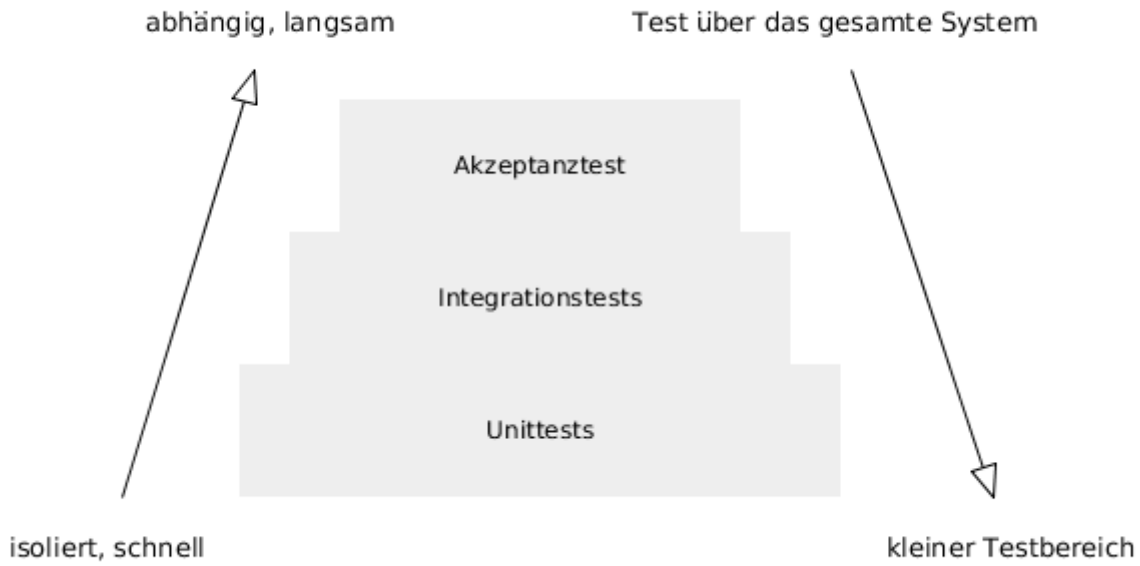


Abbildung 13: Teststrategien: Top-Down-Testen und Bottom-Up-Testen 989.png

Bottom-Up-Testen

Verfolgt man die Bottom-Up Strategie beginnt man mit den Unittests. Der Entwickler hat zwar zu Beginn das Endziel im Auge. Dieses Ziel zerlegt er aber in einzelne Komponenten. Das Problem beim Bottom-Up-Ansatz ist, dass es schwierig ist, wirklich zu testen, wie eine Komponente später verwendet wird. Der Vorteil von Bottom-Up-Tests ist, dass wir sehr schnell fertige und getestet Softwareteile haben. Wobei diese Teile mit Vorsicht zu genießen sind. Sie arbeiten zwar korrekt. Dies stellen die Unittests sicher. Ob das Endergebnis aber wirklich so ist, wie der Kunde sich die Software vorgestellt, ist nicht sichergestellt.

Top-Down-Testen

Verfolgt man die Top-Down Strategie beginnt man mit den Akzeptanztests. Das heißt, der Teil des Systems, der am engsten mit den Benutzeranforderungen verknüpft ist, wird als erstes getestet. Da Software oft für menschliche Benutzer geschrieben wird ist dies in der Regel die Benutzerschnittstelle. Der Schwerpunkt liegt darauf, wie ein Benutzer mit dem System interagieren. Ein Nachteil von Top-Down-Tests ist, dass sehr

Zeit für die Erstellung von Testduplikaten verwendet werden muss. Es gibt noch keinen echten Programmcode. Deshalb muss dieser künstlich erstellt werden. Nach und nach werden diese künstlichen Daten dann durch wirklich berechnete Daten aus implementieren Funktionen ersetzt.

(Todo Akzeptanztests testen nicht die Korrektheit des Programms sondern die Korrektheit des Verhaltens

Todo Akzeptanztest können Tests in Unittests verringeren verweis code coverage

todo Testplan?)

Kurzgefasst

Im 2. Kapitel möchte ich erklären, wie die Testumgebung am Beispiel des Content Management Systems CMS Joomla! eingerichtet werden kann (https://github.com/joomla/joomla-cms). Hier gebe ich auch eine kleine Einführung in den Pakte Manager Composer (https://getcomposer.org/). Außerdem erkläre ich die Struktur des CMS Joomla! kurz. Ein weiteres Thema wird die Planung der Tests sein: Was soll wie getestet werden.

Codeception – ein Überblick

> Testing is the process of operating a system or component under specified conditions, observing or recording the results and making an evaluation of some aspects of the system or component. [ANSI/IEEE Std. 610.12-1990, S. 76]

(todo Einleitung)

(todo http://codeception.com/install)

Wir möchten Tests für eine Joomla! Erweiterung mit Codeception Tests schreiben. Dafür installieren wir Codeception im Joomla! Projekt selbst. Alternativ könnte Codeception auch global für alle Anwendungen auf Ihrem Rechner installiert werden. Ich empfehle und zeige Ihnen hier aber die projektspezifische Installation. Um Codeception zu installieren benötigen Sie zunächst die Software Composer.

Composer

Wer oder was ist Composer und wofür wird Composer gebraucht? Um diese Frage zu beantworten, ist es wichtig sich die folgende Tatsache in Erinnerung zu rufen: Es gibt eine unübersehbare Menge von PHP-Bibliotheken, Frameworks und Bausteinen. Wenn Sie schön länger mit PHP arbeiten, werden Sie sicherlich in Ihren Projekt die eine oder andere externe Software einsetzen. Ihr PHP-Projekt ist somit abhängig von einem anderen Projekt. Diese Abhängigkeiten mussten lange Zeit manuell verwaltet werden. Zusätzlich mussten Sie mithilfe von Autoloading sicherstellen, dass die verschieden Bausteine sich auch gegenseitig kennen. Mit Composer ist dies Gott sei Dank Vergangenheit.

> Vielleicht kennen Sie **PEAR** und fragen Sie sich nun, ob dieser Paketmanager ein Pendant zu Composer ist. Nicht ganz. Verwenden Sie PEAR für die Verwaltung von Abhängigkeiten von PHP in einem **Gesamtsystem** wie Ihrem Computer und Composer für die Verwaltung von Abhängigkeiten in einem **einzelnen Projekt**.

Sie führen Composer Befehle über die Kommandozeile aus. In der Regel werden mithilfe von Composer andere PHP Programme, zu denen das aktuell zu installierende Pakete in einer Abhängigkeitsbeziehung steht, automatisch installiert. Welche PHP Anwendungen über Composer verfügbar sind, können Sie über die Plattform Packagist herausfinden. Composer ist noch recht neu. Die erste Version wurde im März 2012 veröffentlicht. An der Entwicklung von Composer können Sie sich beteiligen – Composer wird auf Github entwickelt.

Installation

Sie können Composer lokal in Ihrem aktuellen Joomla! Projekt installieren - oder global, etwa in im Verzeichnis /usr/local/bin. Dieses Verzeichnis ist von Haus aus in der Umgebungsvariable $PATH hinterlegt, und wird bei einem Programmaufruf nach dem entsprechenden Programm durchsucht.

Ich habe Composer unter Ubuntu 16.04 im Verzeichnis /usr/local/bin installiert. Dazu habe ich die Datei composer.phar mittels

```
$ wget https://getcomposer.org/composer.phar
```

heruntergeladen. Eventuell müssen Sie vorher die Paketquellen in der /etc/apt/sources.list aktualisieren.

```
$ sudo apt-get update
```

Falls wget noch nicht auf Ihrem Rechner installiert ist, können Sie es mit dem Befehl

```
$ sudo apt-get install wget
```

installieren.

Danach habe ich die Datei composer.phar in composer umbenannt und ausführbar gemacht.

```
$ mv composer.phar composer
$ chmod +x composer
```

Nun kann ich composer lokal, also in dem Verzeichnis in dem die Datei abgelegt ist, ausführen.

```
$ ./composer
```

Um auch global auf den Paketmanager zugreifen zu können, habe ich die Datei in das Verzeichnis /usr/local/bin verschoben.

```
$ sudo mv composer /usr/local/bin
```

Nun ist Composer auf meinem Rechner überall verfügbar. Probieren Sie es aus. Die Eingabe des Befehls composer zeigt Ihnen, egal wo Sie sich gerade befinden, eine Liste mit allen möglichen Befehlen an.

```
$ composer
Composer version 1.2.4 2016-12-06 22:00:51
Usage:
  command [options] [arguments]
Options:
  -h, --help                     Display this help message
  -q, --quiet                    Do not output any message
  -V, --version                  Display this application version
      --ansi                     Force ANSI output
      --no-ansi                  Disable ANSI output
  -n, --no-interaction           Do not ask any interactive question
      --profile                  Display timing and memory usage information
      --no-plugins               Whether to disable plugins.
  -d, --working-dir=WORKING-DIR  If specified, use the given directory as working directory.
  -v|vv|vvv, --verbose           Increase the verbosity of messages: 1 for normal output, 2 for more
verbose output and 3 for debug
Available commands:
  about            Short information about Composer
  archive          Create an archive of this composer package
  browse           Opens the package's repository URL or homepage in your browser.
  clear-cache      Clears composer's internal package cache.
  clearcache       Clears composer's internal package cache.
  config           Set config options
  create-project   Create new project from a package into given directory.
  depends          Shows which packages cause the given package to be installed
  diagnose         Diagnoses the system to identify common errors.
  dump-autoload    Dumps the autoloader
  dumpautoload     Dumps the autoloader
  exec             Execute a vendored binary/script
  global           Allows running commands in the global composer dir ($COMPOSER_HOME).
  help             Displays help for a command
  home             Opens the package's repository URL or homepage in your browser.
  info             Show information about packages
  init             Creates a basic composer.json file in current directory.
  install          Installs the project dependencies from the composer.lock file if present, or falls back on
the composer.json.
  licenses         Show information about licenses of dependencies
  list             Lists commands
```

```
outdated        Shows a list of installed packages that have updates available, including their latest
version.
prohibits       Shows which packages prevent the given package from being installed
remove          Removes a package from the require or require-dev
require         Adds required packages to your composer.json and installs them
run-script      Run the scripts defined in composer.json.
search          Search for packages
self-update     Updates composer.phar to the latest version.
selfupdate      Updates composer.phar to the latest version.
show            Show information about packages
status          Show a list of locally modified packages
suggests        Show package suggestions
update          Updates your dependencies to the latest version according to composer.json, and
updates the composer.lock file.
validate        Validates a composer.json and composer.lock
why             Shows which packages cause the given package to be installed
why-not         Shows which packages prevent the given package from being installed
```

> **RANDBEMERKUNG**
> Composer selbst erklärt auf der eigenen Website die Installation für unterschiedliche Plattformen.

Die Dateien Composer.json und Composer.lock

Wenn sie den Befehle composer install ausführen, liest Composer die Datei composer.json im aktuellen Verzeichnis. Falls es diese Datei nicht gibt sieht die Ausgabe wie folgt aus.

```
$ composer install
Composer could not find a composer.json file in /home/astrid
To initialize a project, please create a composer.json file as described in the https://getcomposer.org/
"Getting Started" section
```

Im nächsten Kapitel werden wir eine composer.json, die die Installation von Codeception anforder, anlegen. Sie könnten diese Datei von Hand manuell erstellen. Der Inhalt der Datei sollte wie im folgenden Codeschnipsel aussehen.

```
{
   "require": {
      "codeception/codeception": "*"
   }
}
```

Sie schützen sich aber vor Tippfehlern, wenn Sie den folgenden Befehl verwenden. Dieser Befehl erstellt die Datei composer.json automatisch in der richtigen Syntax.

```
composer require codeception/codeception
```

Praktisch werden wir diese Befehle im nächsten Kapitel anhand der Installation von Codeception ausführen. Im nächsten Kapitel werden Sie dann auch praktisch sehen was passiert, wenn Sie den Befehl composer install in einem Verzeichnis, in dem eine Datei composer.json vorhanden ist, ausführen.

Nur so viel vorab: In diesem Fall werden alle Pakete, die zur Installation der in der Datei composer.json enthaltenen Programme notwendig sind, heruntergeladen und im Unterverzeichnis /vendor installiert. Außerdem wird die Datei composer.lock im Stammverzeichnis angelegt. Die Datei composer.lock dokumentiert die heruntergeladenen Versionsstände der einzelnen Pakete. Wenn Sie Ihr Projekt mit anderen teilen möchten, können Sie mithilfe der Datei composer.lock immer sicherstellen, dass alle Projektbeteiligten mit den gleichen Versionsständen arbeiten. Alle im Projekt enthaltenen Pakete werden immer in der Version, die in der Datei composer.lock festgehalten ist, zum Projekt hinzugefügt. Und das auch dann, wenn ein Paket in der Zwischenzeit aktualisiert wurden.

> RANDBEMERKUNG
> Wenn Sie in Ihrem eigenen Projekt die neuere Version eines zwischenzeitlich aktualisierten Paketes einsetzten möchten, laden Sie diese zunächst in Ihr lokales Projektverzeichnis und sehen sich in Ruhe an, ob die Änderungen im aktualisierten Paket negative Auswirkungen auf Ihr Projekt haben. Falls dies nicht so ist, können Sie die Versionsnummer in der composer.lock und der Datei composer.json herauf setzten. Dies bewirkt, dass in Ihrem Projekt zukünftig die neuere Version automatisch über Composer installiert wird.

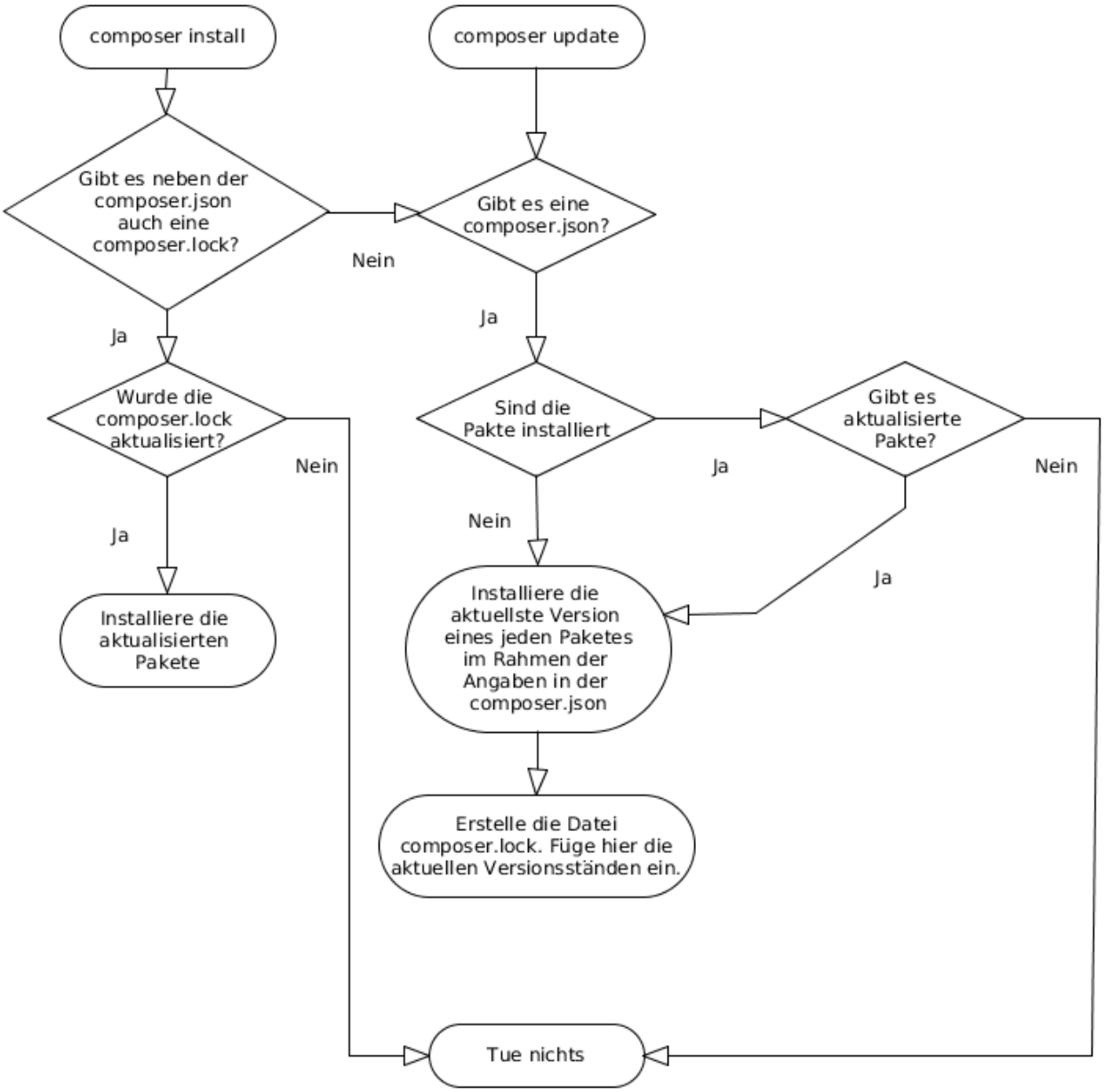


Abbildung 14: Die Dateien Composer.json und Composer.lock im Zusammenspiel mit den Befehlen composer install und composer update 960.png

Paketverwaltung

Die Datei composer.json, die ich im letzten Kapitel als Beispiel genannt hatte, hat es sich mit der Versionsnummer sehr einfach gemacht. Ein Sternchen - also das Symbol * - prüft die Konfiguration und die Abhängigkeiten und fügt dann die neueste mögliche Version des Paketes zu Ihrem Projekt hinzu.

```
{
   "require": {
      "codeception/codeception": "*"
   }
}
```

Sie können die Version eines anzufordernden Paktes genauer eingrenzen. Neben dem Sternchen (*), gibt es noch die Tilde (~), den Zirkumflex (^) und die Vergleichszeichen. Die genaue Bedeutung der Zeichen ist leichter mit Beispielen zu erklären. In der nachfolgenden Tabelle finden Sie Beispiele für unterschiedliche Anwendungsfälle.

Format	Trifft zu auf	Beispiel
^1.0	>= 1.0 < 2.0	1.0, 1.2.3, 1.9.9
^1.1.0	>= 1.1.0 < 2.0	1.1.0, 1.5.6, 1.9.9
~1.0	>= 1.0 < 2.0.0	1.0, 1.4.1, 1.9.9
~1.0.0	>= 1.0.0 < 1.1	1.0.0, 1.0.4, 1.0.9
1.2.1	1.2.1	1.2.1
1.*	>= 1.0 < 2.0	1.0.0, 1.4.5, 1.9.9
1.2.*	>= 1.2 < 1.3	1.2.0, 1.2.3, 1.2.9

Codeception

Nachdem Sie nun die Grundlagen von Composer kenne, können wir die Installation von Codeception in Angriff nehmen.

Installation

In diesem Kapitel zeige ich Ihnen wie Sie Codeception installieren können. Grundlage ist die Vorgehensweise, die Codeception selbst auf der eigenen Website vorschlägt. Besorgen wir uns also zunächst einmal mit dem Befehl `wget http://codeception.com/codecept.phar` die Datei codecept.phar.

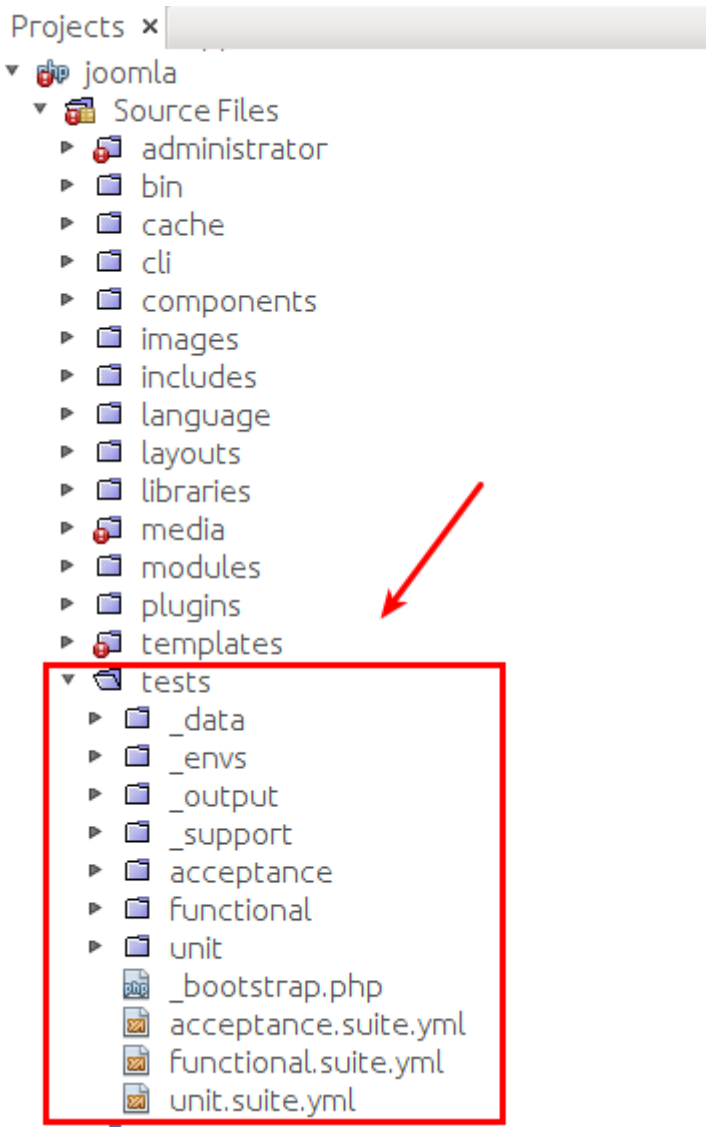
```
/var/www/html/joomla$ wget http://codeception.com/codecept.phar
--2017-02-17 22:06:39-- http://codeception.com/codecept.phar
Auflösen des Hostnamen »codeception.com (codeception.com)«... 192.30.252.154, 192.30.252.153
Verbindungsaufbau zu codeception.com (codeception.com)|192.30.252.154|:80... verbunden.
```

```
HTTP-Anforderung gesendet, warte auf Antwort... 200 OK
Länge: 10345974 (9,9M) [application/octet-stream]
In »»codecept.phar«« speichern.
codecept.phar       100%[===================>]   9,87M   122KB/s    in 56s
2017-02-17 22:07:36 (181 KB/s) - »codecept.phar« gespeichert [10345974/10345974]
```

Mit dem Befehl `php codecept.phar bootstrap` installiert Codeception sich quasi selbst. Probieren Sie es aus.

```
/var/www/html/joomla$ php codecept.phar bootstrap
 Initializing Codeception in /var/www/html/joomla
File codeception.yml created       <- global configuration
tests/unit created                 <- unit tests
tests/unit.suite.yml written       <- unit tests suite configuration
tests/functional created           <- functional tests
tests/functional.suite.yml written <- functional tests suite configuration
tests/acceptance created           <- acceptance tests
tests/acceptance.suite.yml written <- acceptance tests suite configuration
 ---
tests/_bootstrap.php written <- global bootstrap file
Building initial Tester classes
Building Actor classes for suites: unit, acceptance, functional
 -> UnitTesterActions.php generated successfully. 0 methods added
\UnitTester includes modules: Asserts, \Helper\Unit
UnitTester.php created.
 -> AcceptanceTesterActions.php generated successfully. 0 methods added
\AcceptanceTester includes modules: PhpBrowser, \Helper\Acceptance
AcceptanceTester.php created.
 -> FunctionalTesterActions.php generated successfully. 0 methods added
\FunctionalTester includes modules: \Helper\Functional
FunctionalTester.php created.
Bootstrap is done. Check out /var/www/html/joomla/tests directory
```

Wenn alles richtig gelaufen ist, haben Sie nun eine zusätzliches Verzeichnis in Ihrem Projekt. Dieses Verzeichnis heißt `tests`.

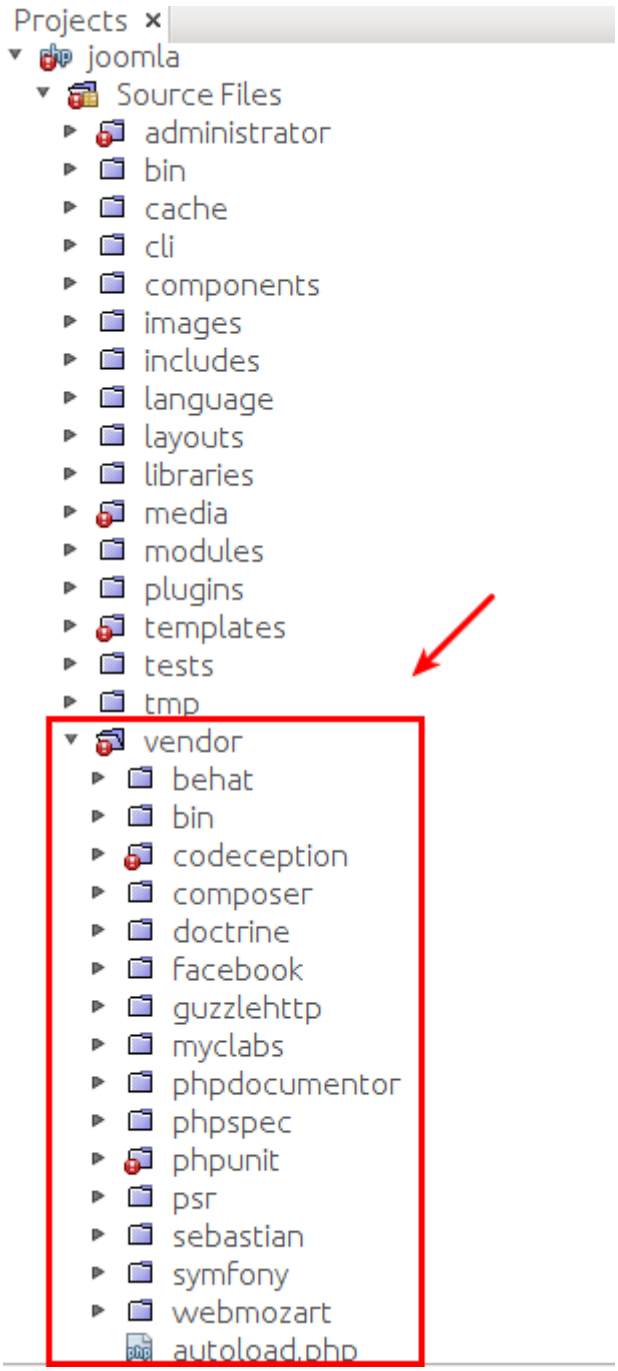


Das Grundgerüst von Codeception ist installiert. Besorgen wir uns nun alle Pakte, die in einem Abhängigkeitsverhältnis zu Codeception stehen. Geben Sie dazu den Befehl `composer require codeception/codeception` ein. Was dieser genau bewirkt, war Thema im letzten Kapitel.

```
/var/www/html/joomla$ composer require codeception/codeception
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
  - Installing symfony/yaml (v3.2.4)
    Downloading: 100%
  - Installing symfony/finder (v3.2.4)
    Loading from cache
```

```
 - Installing symfony/event-dispatcher (v3.2.4)
   Loading from cache
 - Installing symfony/polyfill-mbstring (v1.3.0)
   Loading from cache
 - Installing symfony/dom-crawler (v3.2.4)
   Loading from cache
 - Installing symfony/css-selector (v3.2.4)
   Loading from cache
 …
 - Installing facebook/webdriver (1.3.0)
   Loading from cache
 - Installing behat/gherkin (v4.4.5)
   Loading from cache
 - Installing codeception/codeception (2.2.9)
   Loading from cache
symfony/event-dispatcher suggests installing symfony/dependency-injection ()
symfony/event-dispatcher suggests installing symfony/http-kernel ()
symfony/console suggests installing symfony/filesystem ()
sebastian/global-state suggests installing ext-uopz (*)
phpunit/phpunit-mock-objects suggests installing ext-soap (*)
phpunit/phpunit suggests installing phpunit/php-invoker (~1.1)
facebook/webdriver suggests installing phpdocumentor/phpdocumentor (2.*)
codeception/codeception suggests installing codeception/specify (BDD-style code blocks)
codeception/codeception suggests installing codeception/verify (BDD-style assertions)
codeception/codeception suggests installing flow/jsonpath (For using JSONPath in REST module)
codeception/codeception suggests installing phpseclib/phpseclib (for SFTP option in FTP Module)
codeception/codeception suggests installing league/factory-muffin (For DataFactory module)
codeception/codeception suggests installing league/factory-muffin-faker (For Faker support in
DataFactory module)
codeception/codeception suggests installing symfony/phpunit-bridge (For phpunit-bridge support)
Writing lock file
Generating autoload files
```

Das war ihnen nun schon klar. Ich schreibe es aber der Vollständigkeit halber trotzdem noch einmal. In ihrem Projekt hat Composer nun das Verzeichnis `vendor` angelegt. In diesem Verzeichnis finden Sie alle Softwarepakete, die Codeception benötigt.



Sie können nun sogar schon einen Testlauf starten. Ich weiß, wir haben noch keine Tests erstellt. Es kann also auch nichts wirklich getestet werden. Starten Sie trotzdem einmal mit dem Befehl `vendor/bin/codecept run unit` einen Test. Wenn Ihnen daraufhin keine Fehler gemeldet wird, können Sie sicher sein, dass Codeception richtig installiert und konfiguriert ist.

```
/var/www/html/joomla$ vendor/bin/codecept run unit
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
```

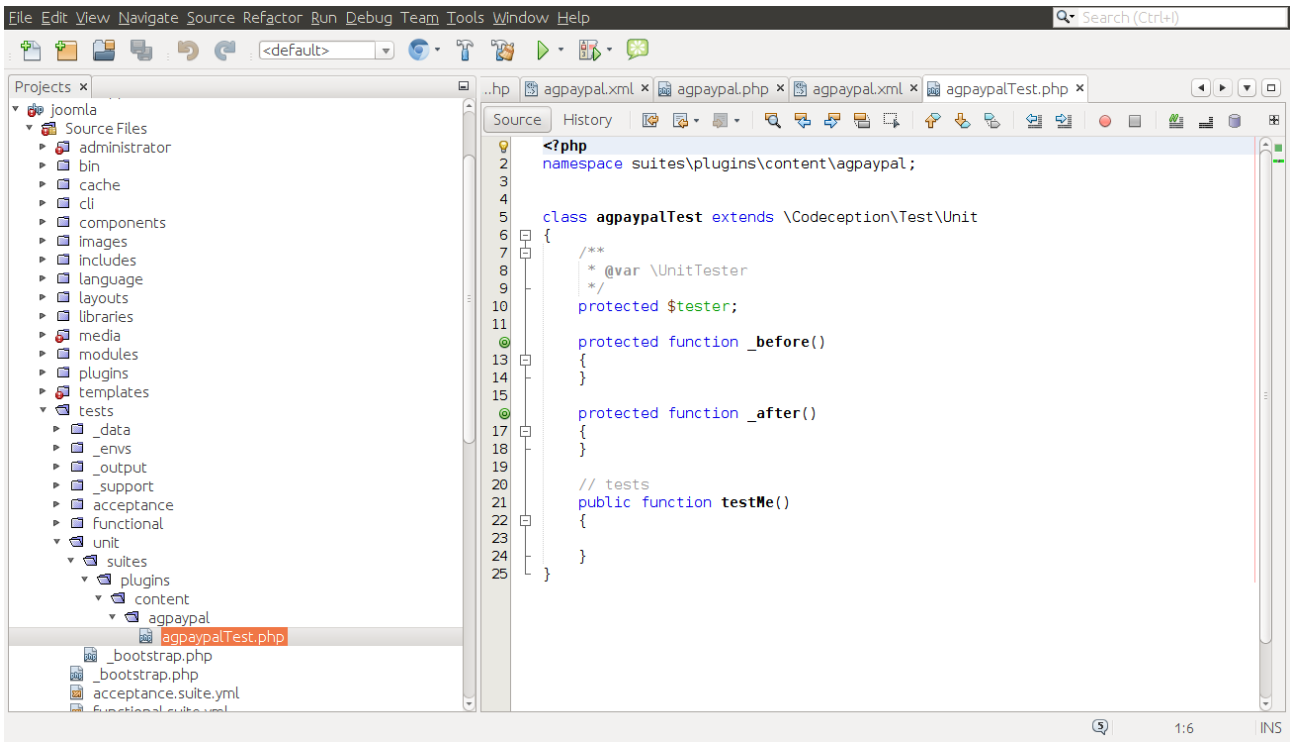
```
Unit Tests (0) ------------------------------------------------------------
------------------------------------------------------------------------
Time: 95 ms, Memory: 8.00MB
No tests executed!
```

Codeception läuft fehlerfrei. Wir können den ersten Test erstellen. Am einfachsten ist dies über den Testgenerator. Geben Sie den Befehl `vendor/bin/codecept generate:test unit` ein. Sie werden daraufhin die `agpaypalTest.php` im Verzeichnis `/var/www/html/joomla/tests/unit/suites/plugins/content/agpaypal` vorfinden.

```
/var/www/html/joomla$ vendor/bin/codecept generate:test unit
/suites/plugins/content/agpaypal/agpaypal
Test was created in
/var/www/html/joomla/tests/unit/suites/plugins/content/agpaypal/agpaypalTest.php
```

In der automatisch generierten Version enthält die Datei nur leere Methoden. Codeception kann einem nur die Routinearbeiten abnehmen, indem es das Boilerplate erstellt. Im weiteren Verlauf dieses Buches werden wir die Methoden genauer ansehen und mit Inhalt füllen.

> **RANDBEMERKUNG**
> Die Testklassen, im Beispiel hier die Klasse `agpaypalTest`, werden bewusst getrennt von der Produktionsklassen `plgContentAgpaypal` in einem separaten Verzeichnis abgelegt. Diese Klassen werden Sie später, wenn Ihr Programm fertig ist, nicht mit beim Kunden installieren.



Codeception – ein erster Rundgang

Nachdem nun alles installiert und lauffähig ist werden wir uns mit der Implementation der ersten realen Tests befassen. Da eine gute Vorbereitung die halbe Miete ist, sehen starten zunächst das, was Codeception uns bietet, genauer an. Ich führe Sie als erstes durch die Verzeichnisstruktur von Codeception. Wenn Sie sich in dem Programm dann zurecht finden, ist der zweite Teil ein Kinderspiel. Sie wissen dann, wie Codeception intern Daten verarbeitet, welche Erweiterungen es gibt, welche Module es gibt und kennen die grundlegende Syntax.

Was steckt in Codeception?

Falls Sie bereits Tests mit einem Testwerkzeug geschrieben haben, kennen Sie die Problematik sicherlich. Jedes Werkzeug hat seine Vorteile, seine Nachteil und seine Tücken. Das eine Tool bietet Ihnen einen guten Support, dafür können Sie die Tests nicht automatisieren. Bietet ein Testtool effizientes und komfortables Arbeiten, ist die Lernkurve vielleicht sehr steil - oder umgekehrt.

Es gibt nur wenige Unternehmen, die intensiv Tests einsetzen. Diese Firmen habe in der Regel Wissen im Bereich Softwaretests selbst aufgebaut oder verfügen über das notwendige Geld um das Wissen einzukaufen.

Für die Programmiersprache PHP ist das bekannteste Werkzeug zweifelsfrei PHPUnit. PHPUnit ist ein PHP-Framework, mit dem Sie PHP-Skripte testen können. Es eignet

sich besonders für automatisierte Unittests. PHPUnit basiert auf Konzepten, die zuvor in dem Java Pendant JUnit umgesetzt wurden. Aber, auch als bekanntestes Tool hat PHPUnit seine Grenzen. Wenn Sie Integrationstests oder Akzeptanztests schreiben möchten, werden selbst an diese Grenzen stoßen. Wie der Name schon vermuten lässt ist das Framework für diese Aufgaben nicht gemacht. Hierfür gibt es andere Werkzeuge. Jedes Tool für sich hat seine Berechtigung. Als Anwender müssen Sie sich aber immer wieder für jedes Programm neu einrichten. Jedes Tool hat seine eigenen individuellen Besonderheiten. Die Bedienung des Programms, die Konfiguration der Software, die Syntax und nicht zuletzt die Regeln beim Erstellen der Tests müssen neu gelernt werden. Codeception will hier Abhilfe schaffen.

> **RANDBEMERKUNG**
> Möchten Sie sich einen Überblick über die verschieden Testframeworks - nur im Bereich Unittests – verschaffen? Wikipedia bietet eine umfangreiche Liste.

Codeception ist mehr als nur ein ein weiteres Werkzeuge

Codeception ist kein weiteres Werkzeug. Warum sollte auch ein weiteres Werkzeug geschaffen werden? Es gibt ja genug Hilfsmittel. Eine Neuschaffung wäre keine Lösung. Am Ende würde ein Tester wieder vor dem gleichen Problem stehen, nämlich unterschiedliche Programme nutzen, aufeinander abstimmen und erlernen zu müssen. Denn auch ein neues Werkzeuge wäre höchstwahrscheinlich nicht die Eierlegende Wollmilchsau die alles gleich gut kann.

Deshalb bündelt Codeception vielmehr verschiedene Tools. Es ist so etwas wie ein *Framework für Frameworks.*

Codeception bietet eine einheitliche Art Tests zu schreiben. Dabei unterstützt es unterschiedliche Testtypen - nämlich Unittests, Integrationstest/Funktionstests und Akzeptanztest. Die Logik und die Vorgehensweise beim Erstellen der unterschiedlichen Tests ist mithilfe von Codeception für alle Testtypen einheitlich. Damit wirkt die ganze Testinfrastruktur stimmig und auf schlüssige Weise zusammenhängend.

Codeceptions Konzepte stellen sich vor

Codeception schreibt auf der eigenen Website selbst über sich: Codeception wurde entwickelt um ein Werkzeug zu bieten, dass möglichst einfach zu bedienen ist. Schon die Installation ist einfach. Das haben Sie im vorausgehenden Kapitel selbst erfahren.

Codeception legt wert darauf, dass alle Tests **leicht zu lesen** sind. Ziel ist es, dass alle Projektbeteiligten, auch die die nicht selbst programmieren, einen Test lesen können und ohne weitere Erklärungen den Testgegenstand verstehen.

Ich hatte etwas weiter vorne schon geschrieben, dass ein Problem bei der Verwendung unterschiedlicher Testtools die unterschiedliche Syntax ist. Jedes Programm hat in der Regel seine eigenen Syntax-Regeln. Codeception unterstützt beim Erstellen von Tests durch die Vereinheitlichung der Syntax. Tests sollen **einfach zu schreiben** sein!

Ein weiteres Ziel von Codeception ist es, Fehler leichter auffindbar zu machen. Dafür ist es wichtig, jederzeit die aktuelle Variablenbelegung im Testcode ablesen zu können. Mit Codeception sind Test **leicht zu Debuggen**.

Und dabei ist Codeception modular aufgebaut, so dass es **leicht zu erweitern** ist.

Zusätzlich unterstützt Codeception mithilfe von Entwurfsmustern die **Wiederverwendbarkeit** von Programmcode eines Tests in anderen Tests.

Sind Sie nun neugierig geworden? Dann tauchen wir doch tiefer in die Details ein.

Testtypen in Codeception

Ich habe im Kapitel *Praxisteil: Die Testumgebung einrichten* | *Unsere Tests mit Codeception planen* | *Testtypen* die wichtigsten Testtypen in Codeception, nämlich Unittests, Funktionstests (Integrationstests) und Akzeptanztests erläutert.

Jeder Testtype hat in Codeception seinen eigenen Bereich, sprich Ordner.

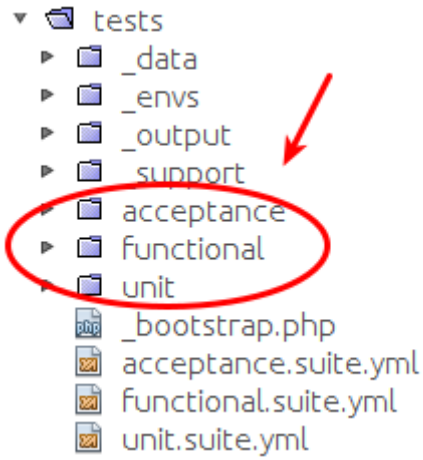


Abbildung 15: Verzeichnisse für die Testtypen innerhalb von Codeception. 976.png

Bevor Sie mit dem Schreiben von Tests beginnen, sollten Sie sicherstellen, dass alle notwendigen Codeception Klassen vorhanden sind. Anlegen können Sie dieser jederzeit mithilfe des Befehls vendor/bin/codecept build. Geben Sie den Befehl nun bitte selbst ein.

```
/var/www/html/joomla$vendor/bin/codecept build
Building Actor classes for suites: unit, acceptance, functional
 -> UnitTesterActions.php generated successfully. 0 methods added
\UnitTester includes modules: Asserts, \Helper\Unit
 -> AcceptanceTesterActions.php generated successfully. 0 methods added
\AcceptanceTester includes modules: PhpBrowser, \Helper\Acceptance
 -> FunctionalTesterActions.php generated successfully. 0 methods added
\FunctionalTester includes modules: \Helper\Functional
```

Was ist genau passiert? Drei Dateien, nämlich die Dateien UnitTesterActions.php, AcceptanceTesterActions.php und FunctionalTesterActions.php, wurden erfolgreich erstellt. Sehen Sie sich diese Klassen genauer an. Sie sind prall gefüllt mit hilfreichen Funktionen die Sie in Ihren Tests später verwenden können. Sie finden die Klassen im Verzeichnis /var/www/html/joomla/tests/_support/_generated/

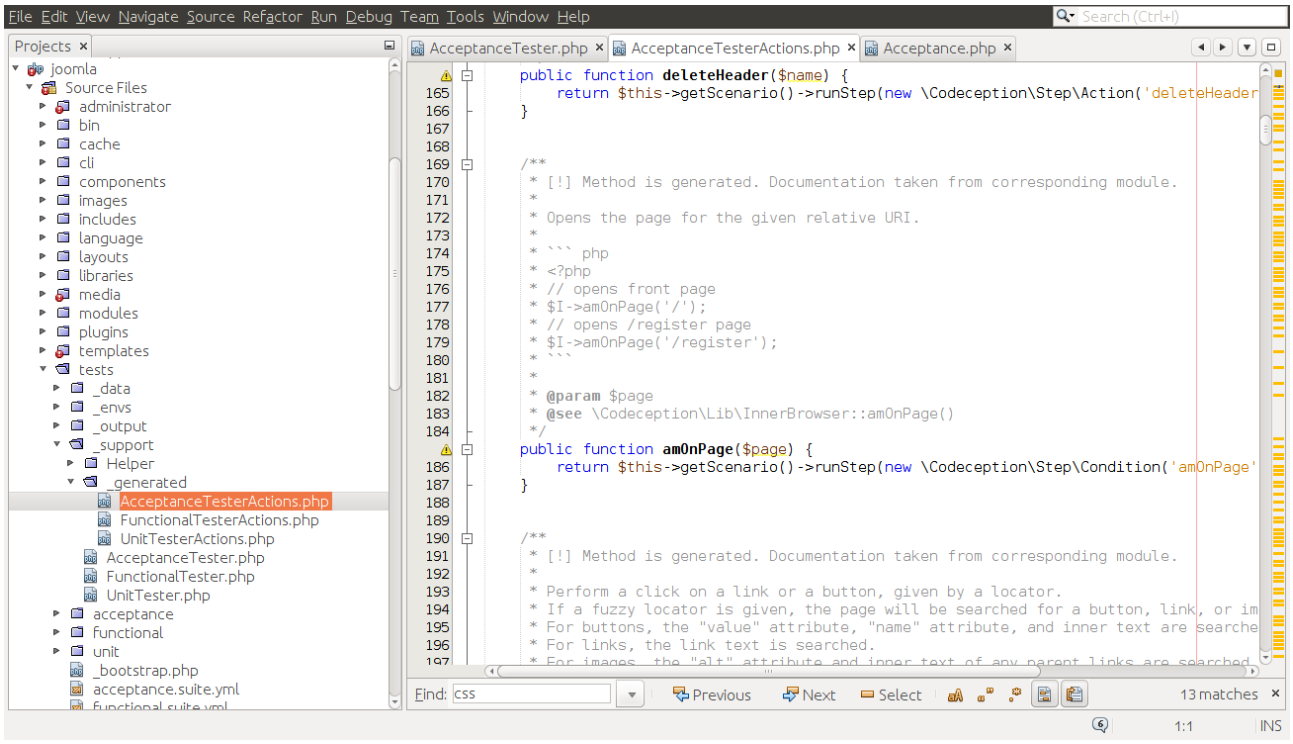


Abbildung 16: Die Klasse AcceptanceTesterAction.php unmittelbar nach der automatischen Generierung 975.png

WICHTIG: `vendor/bin/codecept build`

Das Kommando `vendor/bin/codecept build` sollten Sie nach Änderungen in der Konfiguration von Codeception ausführen. Sie können Sich merken: Immer dann wenn Sie eine Datei die mit `.suite.yml` endet und sich innerhalb des Ordners `/tests` befindet ändern, sollten Sie Codeception neu *bilden*. Sie müssen den build-Befehl auf jeden Fall dann ausführen, wenn Sie mit der Änderungen in der Konfiguration neue Module in Codeception laden und nutzen möchten.

Sind Sie beim Stöbern im Codeception Ordner auch über die Klassen UnitTester, AcceptanceTester und FunctionalTester gestopert? Falls nicht: Sie finden diese Klassen im Verzeichnis `/var/www/html/joomla/tests/_support/`. Ich musste schmunzeln, als ich diese Namen zum ersten Mal gelesen habe. Codeception schafft mit dieser Namensgebung aber ganz nebenbei, dass man beim Schreiben der Tests einen wirklichen Tester, also einen Menschen, der das Programm ausführt, vor Augen hat.

Die verhaltensgetriebene Softwareentwicklung, oder das Behavior Driven Development (BDD) wird so ideal ergänzt. BDD hält idealerweise während der Anforderungsanalyse die geforderten Funktionen der Software in einer bestimmten Textform fest. Diese Funktionen werden dabei meist in „Wenn-dann"-Sätzen beschrieben. Diese „Wenn-dann"-Sätze können nun als Tests ausgeführt werden. **Wenn** der **Tester** dies tut, **dann** sollte das passieren. So kann die Software auf ihre korrekte Implementierung getestet werden.

Getestet wird nicht nur ob **das Programm richtig funktioniert**. Der Tester stellt auch sicher, dass das **richtige Programm erstellt wurde**. Diesen Satz hatte ich an anderer Stelle schon einmal ähnlich geschrieben. Ich finde aber, dass er so wichtig ist, dass man ihn nicht oft genug wiederholen kann.

Ich stelle Ihnen nun die Tester kurz vor, bevor Sie diese in den nachfolgenden Kapiteln in der Praxis kennenlernen.

AcceptanceTester

Unter dem Akzeptanztester können Sie sich eine Person vorstellen, die technisch nicht sehr interessiert ist. Der Akzeptanztester möchte von einem System unterstützt werden. Dabei soll alles, so wie in der Spezifikation festgelegt, funktionieren.

> **RANBEMERKUNG -**
> Zum Beispiel könnte ein Akzeptanzkriterium des Content Management Systems Joomla! sinngemäß so lauten: Wenn ich die Installationsdatei auf meinen Server kopiere und die Fragen der Installationsroute richtig beantworte, dann kann ich auf den Administrationsbereich von Joomla! zugreifen und hier Einstellungen vornehmen. Sehen Sie sich das 24-sekündige Video zu diesem Szenario auf YouTube an. So bekommen Sie einen Eindruck und verstehen, warum ich an anderer Stelle schreiben, dass Akzeptanztests sogar Spaß machen können.

Beim Behavior Driven Development wurde die Spezifikation mittels Beispielen, sogenannten Szenarios, in der Entwurfsphase beschrieben. Üblicherweise wird für die Beschreibung dieser Szenarios ein bestimmtes Format verwendet. Eines dieser Formate, welches auch von Codeception unterstützt wird, ist die Beschreibungssprache Gherkin. Sie können Gherkin sowohl mit den englischen Schlüsselwörtern Given, When, Then und And oder den deutschen Schlüsselwörtern Gegeben, Wenn, Dann und Und nutzen. Ich habe mich hier im Buch für die englischen entschieden. Jedes Szenario beschreibt das Verhalten der Software in einem Teilbereich. Im Englischen werden die Szenarien auch User Stories genannt.

Beispielsweise könnte die Anforderung „Website als Administrator verwalten" mit folgenden Szenarios beschrieben werden:

```
Feature: administrator login
  In order to manage my web application
  As an administrator
  I need to have a control panel

  Scenario: Login in Administrator
    Given There is an administrator control panel
   When I Login into administrator control panel
   Then I should see the administrator control panel
```

Dieses Szenario könnte wie folgt implementiert werden.

```
$I->amOnPage(LoginPage::$url);
$I->fillField(LoginPage::$usernameField, $username);
```

```
$I->fillField(LoginPage::$passwordField, $password);
$I->click(LoginPage::$loginButton);
$I->see('Your are logged in');
```

Sie sehen, der Test, zumindest der in Gherkin geschriebene Teil, ist tatsächlich einfach zu lesen. Die Implementierung greift auf Methoden zurück, die in der Klasse AcceptanceTesterActions von Codeception generiert wurden. Diese Klasse hatten Sie sich eben bereits angesehen. Im Kapitel werden wir Akzeptanztests praktisch ausprobieren.

> RANDBEMERKUNG
> Falls Ihnen der Name AcceptanceTester nicht gefällt, könne Sie diesen in der Konfigurationsdatei acceptance.suite.yml ändern. Sie finden diese Datei im Verzeichnis /var/www/html/joomla/tests/. Wenn Sie den Tester Kunde nennen möchten, dann ändern Sie einfach die erste Zeile in der Datei acceptance.suite.yml. Ändern Sie also class_name: AcceptanceTester in class_name: Kunde. Denken Sie daran, danach mit dem Befehl vendor/bin/codecept build den neuen Namen in allen Codeception Dateien zu aktualisieren.

Akzeptanztests sind, wie im Kapitel *Akzeptanztests testen das System mit wirklichen Anwendungsfällen* schon beschrieben Tests, die das Benutzerverhalten am realistischsten reproduzieren. Idealerweise laufen Sie unter den gleichen Bedingungen wie das Produktivprogramm ab. Im Kapitel *Akzeptanztests* werde ich Ihnen zeigen, wie Sie Akzeptanztests automatisch auf einem beliebigen Browser ablaufen lassen können. Einen Vorgeschmack gibt Ihnen dieses kurze Video. Dieses Video zeigt die automatisch ablaufende Installation von Joomla!. Ganz zu Beginn eines Tests sollte ein festgelegter initialer Zustand hergestellt werden. Andernfalls ist es nicht möglich unter den gleichen Testbedingungen den Test zu reproduzieren. Das Unabhängigkeit, Einfachheit und Wiederholbarkeit für alle Testtypen gelten sollte, hatte ich im Kapitel *Was sollten Sie beim generieren von Tests beachten?* schon angesprochen.

FunctionalTester

Die Bezeichnung Funktionstests ist eine Eigenart in Codeception. Im Grunde genommen sind Funktionstests das gleiche wie Integrationstest.

Funktionstests laufen schneller als Akzeptanztests. Grund hierfür ist, dass nicht die gleichen Bedingungen wie beim Produktivsystem gefordert sind. Funktionstests für eine Webapplikation benötigen nicht zwingend einen Webserver. Diese Tests können auch mit einem Browser, der nicht über eine grafische Benutzeroberfläche verfügt und somit schneller abläuft, durchgeführt werden. Browser ohne grafische Benutzeroberfläche nennt man Headless Browser.

Funktionstests nutzen unter anderem die einheitliche Schnittstelle des Architekturstil Representational State Transfer (REST). Was können Sie sich genau unter einer REST-Schnittstelle vorstellen? Vereinfacht gesagt können Systeme über eine REST-Schnittstelle automatisch miteinander kommunizieren. Das bedeutet, dass auch Tests automatisch ablaufen können, ohne das ein Mensch zwischendurch eingreifen muss. Im Internet ist ein Großteil der für REST nötigen Infrastruktur, zum Beispiel Webserver oder HTTP-fähige Clients, vorhanden.

So ist eine Webanwendung die statische Seiteninhalte über das Hypertext-Übertragungsprotokoll HTTP anbietet REST-konform. Ein Content Management System wie Joomla! erzeugt seine Inhalte dynamisch. Die gleiche HTTP-Anfrage - also der gleiche HTTP-Request - kann, je nach Datenbankinhalt, unterschiedliche Informationen anzeigen. Joomla! erfüllt das REST Paradigma somit nicht vollständig. Zur Veranschaulichung zeige ich Ihnen aber im praktischen Teil, genau im Unterkapitel *REST-Schnittstelle* des Kapitels *Funktionstest* einen Funktionstest, der einen HTTP-Zugriff nutzt.

Wird über das Hypertext-Übertragungsprotokoll HTTP auf eine Anwendung zugegriffen, so gibt die verwendete HTTP-Methode an, welche Operation des Dienstes gewünscht ist. Die wichtigsten HTTP-Methoden sind

- GET
  Mit der GET-Methode können Sie eine Ressource, zum Beispiel eine Datei, vom Server anfordern.
- POST
  Mit der POST-Methode können Sie Daten wie Bilder oder HTML-Formular-Daten zur weiteren Verarbeitung zum Server senden.
- PUT
  Die PUT-Methode dient dazu eine Ressource, zum Beispiel eine Datei, unter Angabe der Zieladresse auf einen Webserver zu kopieren. Dabei muss die

Ressource nicht wie bei POST durch ein Skript verarbeitet werden, sondern wird an einer festgelegten Stelle platziert.

- DELETE
  Die DELETE-Methode löscht die angegebene Ressource auf dem Server.

UnitTester

Codeception nutzt das Framework PHPUnit für die Erstellung der Unittest. Hier an dieser Stelle gibt es nicht viel mehr dazu zu sagen. Wenn Sie PHPUnit schon verwenden, können Sie das nächste Kapitel sicherlich schnell bearbeiten. Andernfalls müssen Sie vielleicht das ein oder andere Mal etwas in der Dokumentation von PHPUnit nachlesen.

Mit Codeception Tests organisieren und erweitern

CEPT und CEST

Codeception selbst bietet Ihnen zwei verschiedene Formate. Das Cept-Format ist ein Szenario basiertes Format. Das Cest-Format basiert auf einer Klasse.

> RANDBEMERKUNG
> Streng genommen könnten wir die Unittests als weiteres drittes Format hinzurechnen.

Mithilfe des Befehls vendor/bin/codecept generate:**cest** acceptance TestCest

```
/var/www/html/joomla$ vendor/bin/codecept generate:cest acceptance TestCest
Test was created in /var/www/html/joomla/tests/acceptance/TestCest.php
```

erstellen Sie beispielsweise eine **Cest**-Datei für einen Akzeptanztest.

```
<?php
class TestCest{
    public function _before(AcceptanceTester $I) { }
    public function _after(AcceptanceTester $I) { }
    // tests
    public function tryToTest(AcceptanceTester $I) { }
}
```

Jede Methode in einer CEST-Datei, deren Name nicht mit einem Unterstrich beginnt, wird von Codeception als Test interpretiert. Die Methoden _before() und _after() sind spezielle Methoden. _before() wird vor jedem Test und after() wird nach jedem Test ausgeführt. Zusätzlich können Sie noch die spezielle Methode _fail() verwenden. Sie wird im Falle eines Fehlers ausgeführt und eignet sich deshalb zum Aufräumen im Fehlerfalle.

Mithilfe des Befehls vendor/bin/codecept generate:**cept** acceptance TestCest

```
/var/www/html/joomla$ vendor/bin/codecept generate:cept acceptance TestCest
Test was created in /var/www/html/joomla/tests/acceptance/TestCept.php
```

erstellen Sie eine **Cept**-Datei. Diese Datei enthält unmittelbar nach der automatischen Generierung die folgenden drei Zeilen.

```
<?php
$I = new AcceptanceTester($scenario);
$I->wantTo('perform actions and see result');
```

Nicht-Entwickler bevorzugen oft das CEPT-Format. Entwickler entscheiden sich in der Regel für das CEST-Format. Das CEST-Format unterstützt mehrere Test innerhalb einer Datei. Außerdem können Sie Programmcodes mithilfe von zusätzlichen privaten Methoden leichter wiederverwenden.

> EXKURS CEST-Format für Unittests
> Meiner Meinung nach eignet sich das Cest-Format nur für Akzeptanztests und Funktionstests. Wenn Sie Unittests schreiben verwenden Sie besser das hierfür vorgesehene Format. Das CEST-Format bietet von Haus aus keine Assertion und keine Methoden zum Erstellen von Testduplikaten.

Annotationen

Mit Annotationen können Sie bestimmte Anmerkungen in Ihrem Programmquelltexten verwenden. Mit Hilfe von Zusatzwerkzeugen lassen sich diese Anmerkungen, die im

Grunde genommen nichts anderes als Metainformationen sind, in den PHP Programmcode einbetten und auswerten.

Dies ist insbesondere dann sinnvoll, wenn PHP die Funktionen, die die Annotationen bieten, nicht unterstützt. Die Metadaten werden in Kommentaren vor dem PHP-Compiler verborgen, da dieser die Syntax nicht unterstützt. Codeception, beziehungsweise PHPUnit, bieten Zusatzwerkzeugen, die die Annotationen auswerten und richtig weiterverarbeiten können.

(Todo, die hatte ich doch schon einmal irgendwo beschrieben, oder? Nein das kommt später , es gibt auch eine für Testduplikate)

Page Objekte und Step Objekte

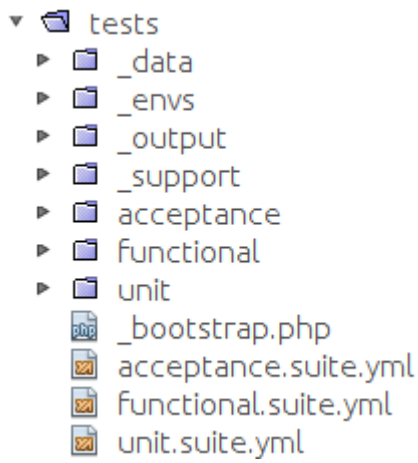
Stellen Sie sich vor, Sie haben viele unterschiedliche Tests geschrieben, in denen auf der Startseite das HTML-Element mit der ID username vorkam. Nun muss der Name dieser ID geändert werden. Ganz egal warum. Sie müssen nun viele unterschiedliche Tests korrigieren und Ihnen ist sicherlich nun schon klar, worauf ich hinaus will.

Sinnvoll ist es, Elemente des Document Objekt Modell (DOM) nur einmal hart zu codieren und dann immer wieder zu verwenden. Denn: Wenn beispielsweise der Name eines Elementes nur an einer Stelle im Programmcode steht, muss nicht die gesamte Testsuite, sondern nur eine Datei geändert werden, wenn dieser Name sich einmal ändert. Codeception hat hierfür **Page Objekte** vorgesehen. Dabei repräsentiert das Page Objekt eine Website und die DOM-Elemente sind Eigenschaften des Page Objekts. Je nach Komplexität der Website kann das Page Objekt aber auch nur einen Teil der Website, zum Beispiel eine Toolbar, darstellen. (todo praktisches beispiel verweisen. Auch für step objekt)

Step Objekte stellen eine andere Art zum Wiederwenden von Programmcode dar. Codeception hat Step Objekte dazu vorgesehen Programmcode bezüglich Aktionen, die immer wieder vorkommen, zur wiederholten Nutzung zur Verfügung zu stellen. Wenn Sie eine Website testen, die aus vielen Formularen besteht, könnten Sie beispielsweise ein Step Objekt schreiben, das die Aktionen in den Formularen simuliert. Hier könnte eine Methode mit zwei Parametern aufgenommen sein, die die Auswahl in einem Listenfeld ausführt. Die Methode könnte selectListValue(listenfeld $l, wert $w) heißen. Immer dann wenn die Auswahl in einem Listenfeld simuliert werden soll, würde ein Aufruf von selectListValue(„tiere“, „hund“) ausreichen, um die Belegung der Liste tiere mit der Option hund zu testen.

Codeception unter die Lupe genommen

Sehen wir die einzelnen Verzeichnisse von Codeception einmal kurz an. Nach der Installation von Codeception zu Beginn dieses Kapitels haben Sie den Ordner tests mit folgendem Inhalt vorgefunden. (todo vendor abhänigkeiten)



Ordnerstruktur

_data

Im Verzeichnis _data können Sie einen Export aus einer Datenbank ablegen. Wenn Sie möchten, können Sie Codeception über die Konfiguration so einstellen, dass vor dem Ausführung eines Tests eine Datenbank aus diesem Export erstellt wird und die Tests so in einem vordefinierten Zustand ausgeführt werden können.

> RANDBEMERKUNG
> Um einen Datenbankexport automatisch vor jedem Testdurchgang in Ihre Datenbank einlesen zu lassen müssen Sie das Db Modul aktivieren. Wie Sie das tun können steht unter der Überschrift *Db* Sie in der Dokumentation von Codeception. Wollen Sie verstehen wie dieses Modul genau arbeitet? Dann sehen Sie sich die Datei /var/www/html/joomla/vendor/codeception/codeception/src/Codeception/Module/Db.php an. Sie können das Modul auch an Ihre Umgebung anpassen, falls es bei Ihnen Besonderheiten gibt. Im Joomla! Projekt gibt es zum Beispiel einen Präfix für jede Datenbanktabelle. Deshalb wurde in der Joomla! Testumgebung das Modul JoomlaDb kreiert, dass das Codeception Modul Db um diese Besonderheit erweitert.

_env

Möchten Sie Tests in unterschiedlichen Installationsumgebungen, die verschiedene Konfigurationen voraussetzen, ausführen? Ein typischer Anwendungsfälle sind Akzeptanztests in verschiedenen Browsern. Es kann aber auch sein, dass Ihre Webanwendung mehrere Datenbanken unterstützen soll. Wenn Sie also mit verschiedenen Konfigurationen gleichzeitig arbeiten möchten können Sie Codeception dafür konfiguriere und dieses Verzeichnis ist für die verschieden Konfigurationsdateien vorgehen. Sehen Sie sich aber vorher auch einmal Docker an. Mit Docker können Sie unterschiedliche Testumgebungen sehr gut isoliert voneinander und sogar parallel ausführen. In der Codeception Dokumentation können Sie mehr zum offiziellen Codeception auf Docker Hub lesen.

_output

Diesen Ordner werde Sie einmal lieben lernen. Wenn Sie später Tests ablaufen lassen und einer dieser Tests fehlschlägt, finden Sie Informationen zu Fehlergründen in diesem Verzeichnis.

_support

In diesem Ordner finden Sie selbst erstellte oder automatisch generierte Support Dateien. Die meisten dieser Datei wurden erstellt, als wir im Kapitel *Testtypen in Codeception* den `build`-Befehl ausgeführt haben.

acceptance

Das Verzeichnis `acceptance` einhält die in Gherkin geschrieben Feature Dateien. Also die Dateien, die für die Ausführung der Akzeptanztests in Codeception zuständig sind. In diesem Verzeichnis ist zusätzlich eine Datei die `_bootstrap.php` heißt und in der Sie PHP Programmcode hinterlegen können, der vor jedem Akzeptanztest automatisch ausgeführt wird.

functional

Das Verzeichnis `functional` einhält alle Testdateien die Funktionstests enthalten. Diese werden in Unterverzeichnissen zusammengefasst. Ein Unterverzeichnis wird auch Testsuite genannt. Im Verzeichnis `functional` ist zusätzlich eine Datei die `_bootstrap.php` heißt und in der Sie PHP Programmcode hinterlegen können, der vor jedem Funktionstest automatisch ausgeführt wird.

unit

Das Verzeichnis unit einhält die Dateien zu Ihren Unittests. Diese werden in Unterverzeichnisse zusammengefasst. Ein Unterverzeichnis wird Testsuite genannt. Im Verzeichnis unit ist zusätzlich eine Datei die _bootstrap.php heißt und in der Sie PHP Programmcode hinterlegen können, der vor jedem Unittest automatisch ausgeführt wird.

Konfiguration

_boostrap.php

In Codeception gibt es mehrere Dateien, die _boostrap.php heißen. Die drei _boostrap.php Dateien in den Unterverzeichnissen zu den drei Testtypen hatte ich Ihnen gerade eben im vorhergehenden Kapitel schon beschrieben. Diese drei Dateien führen Programmcode vor jedem Test – speziell für den einen Testtyp in deren Verzeichnis sie sich befinden - aus. Sicherlich gibt es aber bestimmte Initialisierungen, die für alle Ihre Tests gleichermaßen gelten. Alles, was für jeden Testtyp gilt, können Sie der Einfachheit halber in die Datei _boostrap.php, die direkt im Verzeichnis tests liegt, einfügen. Kommen wir noch einmal auf unser Beispiel Plugin AgPaypal zurück. Für diese Joomla! Erweiterung gibt es ein paar Variablenbelegungen, auf die wir später in allen Tests zugreifen möchten. Bitte fügen Sie doch den nachfolgenden Programmcode in die Datei /var/www/html/joomla/tests/_bootstrap.php ein und speichern die Datei dann.

```
<?php
define("_JEXEC", 'true');
error_reporting(E_ALL);
ini_set('display_errors', 1);
define('JINSTALL_PATH', '/var/www/html/joomla');
define('JPATH_LIBRARIES', JINSTALL_PATH . '/libraries');
require_once JPATH_LIBRARIES . '/import.legacy.php';
require_once JPATH_LIBRARIES . '/cms.php';
```

Was bewirken die Einträge genau? Die Zeile define("_JEXEC", 'true'); setzt die Konstante JEXEC. Nur wenn diese Konstante definiert ist, kann Joomla! ausgeführt und somit auch getestet werden.

> RANDBEMERKUNG
> Alle Skript innerhalb von Joomla! beginnen mit der Zeile defined('_JEXEC') or die;.

> Auch in unser Plugin Agpaypal haben wir diese Zeile integriert. Der Joomla! Prozess stirbt, wenn die Konstante nicht gesetzt ist. Dies ist eine Sicherheitsfunktion innerhalb von Jooma!. Kein Skript innerhalb von Joomla! kann somit von außen, also vom Browser irgeneines Internetnutzers, so manipuliert werden, dass es etwas anderes macht als die Joomla! Entwickler mit ihm beabsichtigten. Die Konstante kann nur innerhalb der Anwendung gesetzt werde, also wenn ein Skript innerhalb von Joomla direkt inkludiert, dass heißt per require oder include in PHP eingebunden wird. Das setzten der Konstante muss also innerhalb des selben Prozesses erfolgen.

Mit error_reporting(E_ALL); und ini_set('display_errors', 1); bewirken wir, dass Fehler nicht unterdrückt sondern angezeigt werden.

Und last but not least bewirkt das Einbinden der Dateien import.legacy.php und cms.php, dass Sie in den Tests auf alle Joomla! Bibliotheken zugreifen können.

/codeception.yml

Die Datei joomla//codeception.yml haben Sie mit dem boostrap-Befehl erstellt und sie enthält die Standardwerte für die notwendigen Konfigurationsangaben für alle drei Testtypen. Den boostrap-Befehl hatten wir zu Beginn des Kapitels im Abschnitt *Codeception | Installation* ausgeführt. Ich drucke Ihnen hier die automatisch erstellte Version dieser Konfigurationsdatei ab. Wir werden diese Datei im späteren Verlauf erweitern. Im Kapitel Analyse werden wir hier die Parameter für die Messung der Testcodeabdeckung, die für alle Tests gleichzeitig gelten sollen, eintragen.

Weitere Informationen zu den einzelnen Konfigurationsparametern finden Sie auf der Website von Codeception in der Dokumentation.

```
actor: Tester
paths:
    tests: tests
    log: tests/_output
    data: tests/_data
    support: tests/_support
    envs: tests/_envs
settings:
    bootstrap: _bootstrap.php
```

```
    colors: true
    memory_limit: 1024M
extensions:
    enabled:
        - Codeception\Extension\RunFailed
modules:
    config:
        Db:
            dsn: ''
            user: ''
            password: ''
            dump: tests/_data/dump.sql
```

/tests/acceptance.suite.yml

Die Datei joomla/tests/acceptance.suite.yml haben Sie mit dem boostrap-Befehl erstellt und sie enthält die Standardwerte für die notwendigen Konfigurationsangaben für Akzeptanztests. Den boostrap-Befehl hatten wir zu Beginn des Kapitels im Abschnitt *Codeception | Installation* ausgeführt. Ich drucke Ihnen hier die automatisch erstellte Version dieser Konfigurationsdatei ab. Wir werden diese Datei im späteren Verlauf erweitern.

```
class_name: AcceptanceTester
modules:
    enabled:
        - PhpBrowser:
            url: http://localhost/myapp
        - \Helper\Acceptance
```

functional.suite.yml

Die Datei /var/www/html/joomla/tests/functional.suite.yml haben Sie mit dem boostrap-Befehl erstellt und sie enthält die Standardwerte für die notwendigen Konfigurationsangaben für Funktionstests. Den boostrap-Befehl hatten wir zu Beginn des Kapitels im Abschnitt *Codeception | Installation* ausgeführt. Ich drucke Ihnen hier die automatisch erstellte Version dieser Konfigurationsdatei ab. Wir werden diese Datei im späteren Verlauf erweitern.

```
class_name: FunctionalTester
modules:
    enabled:
        # add framework module here
        - \Helper\Functional
```

unit.suit.yml

Die Datei /var/www/html/joomla/tests/unit.suite.yml haben Sie mit dem bootstrap-Befehl erstellt und sie enthält die Standardwerte für die notwendigen Konfigurationsangaben für Unittests. Den bootstrap-Befehl hatten wir zu Beginn des Kapitels im Abschnitt *Codeception | Installation* ausgeführt. Ich drucke Ihnen hier die automatisch erstellte Version dieser Konfigurationsdatei ab. Wir werden diese Datei im späteren Verlauf erweitern.

```
class_name: UnitTester
modules:
    enabled:
        - Asserts
        - \Helper\Unit
```

Arbeiten mit Codeception

Das Sie Codeception über den Befehl codecept ausführen, haben Sie schon mit Beispielen kennengelernt. Sie haben auch schon Steuerbefehle kennen gelernt, nämlich build, generate und run.

- Mit dem Befehl **build** veranlassen Sie Codeception dazu, automatisch Programmcode anhand der Eintragungen in den Konfigurationsdateien zu generieren.
- Mit dem Befehl **run** führt Codeception Tests aus. Wenn Sie codecept run starten, können Sie weitere Argumente mitgeben:
  /var/www/html/joomla$ vendor/bin/codecept run **[optionen] [suite] [test]**.
  Der Befehl /var/www/html/joomla$ vendor/bin/codecept run **unit tests/unit/suites/plugins/content/agpaypal/agpaypalTest** führt beispielsweise genau den einen Unittest **agpaypalTest** aus.
  Und die Eingabe /var/www/html/joomla$ vendor/bin/codecept run **-q** unit

tests/unit/suites/plugins/content/agpaypal/agpaypalTest unterdrück die Ausgabe in der Konsole. Die Option q steht für des englische Wort quite.

> **RANDBEMERKUNG**
> Sie können alle Unittests in einer Suite gleichzeitig ausführen. Verwenden Sie dazu einfach den Befehl
> tests/codeception/vendor/bin/codecept run unit
> ohne dabei eine spezielle Datei als Parameter mitzugeben. (todo tests/codeception/vendor/bin/codecept run)

- Mit dem Befehl **generate** erstellen Sie die Tests. Beim Befehl generate müssen Sie den Namen einer Suite und einen Namen für den Test, der erstellt werden soll, mitgeben.
  - generate:cept erstellt einen Test im CEPT-Format
  - generate:cest erstellt einen Test im CEST-Format
  - generate:phpunit erstellt einen PHPUnittest ohne die Erweiterungen die Codeception bietet. Der Test erbt von der Klasse PHPUnit_Framework_TestCase und enthält die Methoden _setUp() und tearDown() automatisch enthalten.
  - generate:test erstellt einen PHPUnittest der von der Klasse Codeception\Test\Unit erbt und zusätzlich zu den Mehtoden _setUp() und tearDown() die Methoden _before() und _after() enthält.

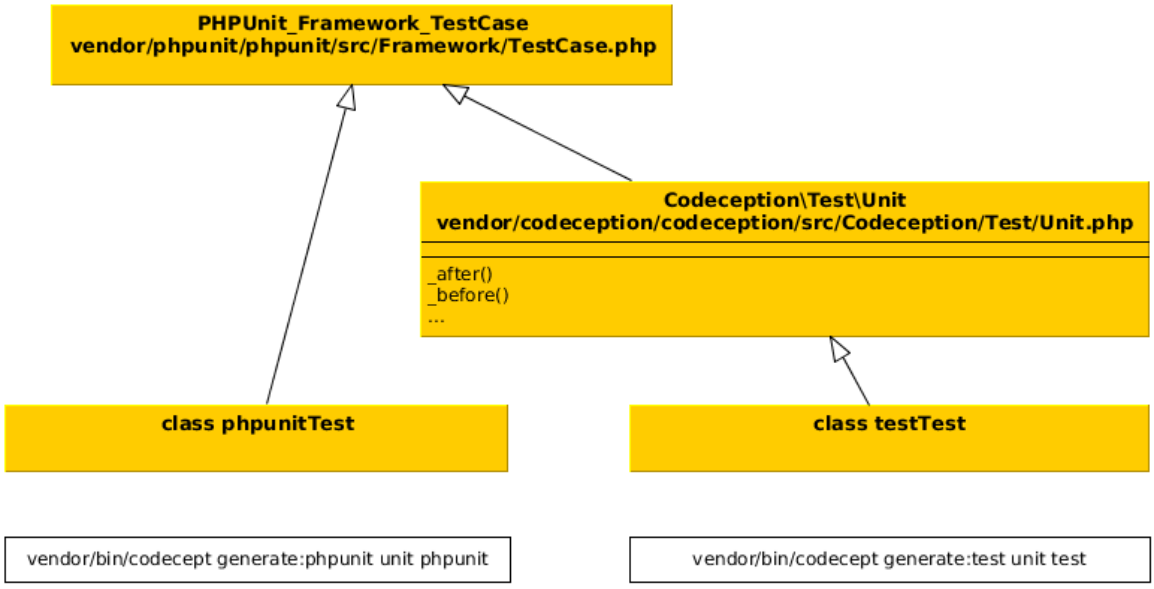


Abbildung 17: Die Klasse Unit erweitert das PHPUnit Framework 959.png

Dies war nun sehr viel Theorie. Wir werden alle diese Befehle im weiteren Verlauf dieses Buches auch praktisch kennen lernen.

Die Datei codecept.php finden Sie im Verzeichnis vendor/bin/. Führen Sie die Datei doch einfach einmal mit der Option -v aus. Dann werden Ihnen alle möglichen Optionen und Argumente angezeigt.

Kurzgefasst

Im 3. Kapitel soll es um das Framework Codeception (https://github.com/Codeception/Codeception) gehen. Welche Konzepte beinhaltet es? Wie erstellt man Tests? Welche Konfigurationsmöglichkeiten gibt es.

Unittests

> The fewer tests you write,
> the less productive you are
> and the less stable your code becomes.
> [Erich Gamma]

(todo Einleitung)

Unittests sind Tests, die Sie gleichzeitig mit der Programmierung des eigentlichen Programms erstellen können. Wenn Sie testgetrieben entwickeln sollten Sie die Tests vor dem eigentlichen Programmcode erstellen. Jeder Test erfolgt, wie der Name schon vermuten lässt, isoliert von anderen Programmteilen. Unittests testen kleinste Einheiten. Im Deutschen nennt man diese Tests auch Modultests oder Komponententests. Bei der testgetriebenen Entwicklung erstellen Sie die Unittests vor jeder kleinen Programmänderung. Beim Erstellen eines Unittests kennen Sie die Implementierung der zu prüfenden Unit oder haben zumindest eine Vorstellung davon, wie Sie die Unit programmieren möchten. Unittests zählen aber trotzdem zu den spezifikationsorientierten Tests, also den Black Box Tests. Denn, das Ziel von Unittests ist es nicht, den Programmcode systematisch zu überprüfen und möglichst alle Programmcodeabschnitte zu testen. Dies tun die implementationsorientierten White Box Tests. Ziel von Unittests ist es, alle Testfälle abzudecken, egal ob dabei alle Programmcodeabschnitte durchlaufen werden oder nicht.

> **RANDBEMERKUNGEN**
> Für unser Plugin Agpaypal hatten wir im Kapitel Tests planen grob folgende drei Testfälle festgehalten: Bei der Prüfung eines Textes auf ein bestimmtes Muster das in einen anderen Text umgewandelt werden soll, kann dieses Muster genau einmal, keinmal oder mehrmals im Text vorkommen.
>
> Am Ende dieses Buches werden wir uns im Kapitel Analyse ein Werkzeug ansehen, dass die Testabdeckung berechnet. Hier wird der Programmcode systematisch durchsucht. Es handelt sich aber nicht um einen White Box Test im eigentlichen Sinne. Ziel der Tools zur Messung der Codeabdeckung ist es, sicherzustellen, dass kein wichtiger Testfall vergessen wurde. Da eine hohe Codeabdeckung auch mit Tests erreicht werden kann, die nichts überprüfen, hat die Codeabdeckung für die Qualität der Tests eine nur eingeschränkte Aussagekraft.
> Black Box Tests und White Box Tests hatte ich in einer Randbemerkung im Kapitel *Softwaretests - eine Einstellungssache?* | *Kontinuierliches Testes* | *Behavior-Driven-Development (BDD)* ausführlicher beschrieben.

Einen ersten Überblick verschaffen

Ich zeige Ihnen hier am Beispiel einer Erweiterung für Joomla! wie Sie Unittest erstellen können. Sie lesen dieses Buch sicherlich, weil Sie sich überlegen, Test in ihre eigene Software einzubauen. Falls Sie noch nicht testgetrieben entwickeln, kommt Ihnen diese Methode zunächst sicherlich sehr umständlich vor. Mir ging es anfangs zumindest so. Eine neue Funktionalität einfach einbauen erscheint viel unkomplizierter. Der Mehraufwand für den Test kommt einem unnötig vor. Außerdem kann man diesen Test ja, wenn nötig, nachträglich noch hinzufügen. Ich empfehle Ihnen die testgetriebene Vorgehensweise einmal auszuprobieren. Vielleicht geht es Ihnen ja so wie mir und Sie möchten diese Erfahrung nicht mehr missen.

Endlich: Ein erstes Testbeispiel

Codeception führt Unittests mit PHPUnit aus. Arbeiten Sie schon mit PHPUnit? Dann müssen Sie nicht umlernen. Sie können jeden schon fertig geschriebenen PHPUnit-Test in die Codeception Testsuite integrieren und hier ausführen. Ich empfehle Ihnen aber einmal über den PHPUnit Tellerrand auf Codeception zu blicken, denn, Codeception bietet Ihnen eine Menge nützlicher Zusatzfunktionen.

Die Vorlage für den ersten Test

Eine dieser Zusatzfunktion von Codeception haben Sie schon benutzen, als Sie den ersten Test während der Installation von Codeception erstellten – nämlich den Testcode-Generator. Mit dem Befehl vendor/bin/codecept generate:test unit

```
/var/www/html/joomla$ vendor/bin/codecept generate:test unit
/suites/plugins/content/agpaypal/agpaypal
```

haben Sie den ersten Test erstellt. Sie finden diesen Test in der Datei agpaypalTest.php im Verzeichnis /joomla/tests/unit/suites/plugins/content/agpaypal/

Ausführen können Sie den Test in der Datei agpaypalTest.php über den Befehl codecept run unit tests/unit/suites/plugins/content/agpaypal/agpaypalTest.

```
/var/www/html/joomla$ vendor/bin/codecept run unit
tests/unit/suites/plugins/content/agpaypal/agpaypalTest
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
Unit Tests (1) ----------
```

```
✔ agpaypalTest: Me (0.00s)
---------------------------------------------------------------------------------
Time: 149 ms, Memory: 8.00MB
OK (1 test, 0 assertions)
```

Der gerade neu generierte Test verläuft erfolgreich. Alles andere wäre auch verwunderlich. Bisher gibt es ja noch keinen wirklichen Testcode. Die automatisch erstellte Testdatei enthält ausschließlich leere Methoden.

```
<?php
namespace suites\plugins\content\agpaypal;
class agpaypalTest extends \Codeception\Test\Unit {
protected $tester;
 protected function _before() {}
 protected function _after(){}
 public function testMe() {}
}
```

Damit die Testklasse selbst Funktionen von Codeception nutzen kann, lassen wir sie von der Framework-Klasse `\Codeception\Test\Unit` erben. Im nächsten Abschnitt werden wir den ersten Test programmieren. Wichtig ist, dass dessen Signatur, analog des automatisch erstellten Beispieltests `testMe()`, der Konvention für PHPUnit Testfallmethoden folgt. Hiernach muss der Methodenname mit dem Wort **test** beginnen.

> RANDBEMERKUNG
> Falls Sie aus irgendeinem Grund den Namen einer Testmethode nicht mit dem Präfix `test` verstehen möchten, können Sie auch die Annotation `@test` verwenden.

Der erste selbst programmierte Test

Was wollen wir testen?

Für unser Plugin Agpaypal hatten wir im Kapitel Tests planen grob folgende Testfälle festgehalten: Bei der Prüfung eines Textes auf ein bestimmtes Muster das in einen anderen Text umgewandelt werden soll, kann dieses Muster

- genau einmal,
- keinmal oder
- mehrmals im Text

vorkommen.

Wie integrieren wir am besten Tests, die diese Testfälle erfüllen? Um diese Frage zu beantworten sehen wir uns den Programmcode unseres Plugins noch einmal an.

```
/var/www/html/joomla/plugins/content/agpaypal/agpaypal.php
<?php
defined('_JEXEC') or die;
class plgContentAgpaypal extends JPlugin{
 public function onContentPrepare($context, &$row, $params, $page = 0)          {
 $search = "@paypalpaypal@";
 $replace = '<form name="_xclick" action="https://www.paypal.com/de/cgi- bin/webscr"
method="post">
 <input type="hidden" name="cmd" value="_xclick">
 <input type="hidden" name="business" value="me@mybusiness.com">
 <input type="hidden" name="currency_code" value="EUR">
 <input type="hidden" name="item_name" value="Teddybär">
 <input type="hidden" name="amount" value="12.99">
 <input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click- but01.gif" border="0"
name="submit" alt="Zahlen Sie mit PayPal – schnell,
 kostenlos und sicher!">
 </form>';
 if (is_object($row)){
   $row->text = str_replace($search, $replace, $row->text);
 } else{
   $row = str_replace($search, $replace, $row);
 }
 return true;
 }
}
```

Fangen wir mit dem Testfall an, bei dem der Text @paypalpaypal@ in der Variablen $row, und somit im Text des anzuzeigenden Beitrags, genau einmal vorkommt. Bei einem fehlerfreien Programmablauf sollte folgendes passieren: Die Variable $row enthält nach Durchlaufen der Methode onContentPrepare() anstelle des Textes

@paypalpaypal@

den Text

```
<form name="_xclick" action="https://www.paypal.com/de/cgi-bin/webscr" method="post">
<input type="hidden" name="cmd" value="_xclick"><input type="hidden" name="business"
value="me@mybusiness.com">
<input type="hidden" name="currency_code" value="EUR">
<input type="hidden" name="item_name" value="Teddybär">
<input type="hidden" name="amount" value="12.99">
<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-but01.gif" border="0"
name="submit" alt="Zahlen Sie mit PayPal – schnell, kostenlos und sicher!">
</form>
```

Ausschließlich diesen Punkt müssen wir testen, wenn wir die korrekte Bearbeitung bei einmaligem Vorkommen des Textes @paypalpaypal@ sicherstellen möchten. Ob andere Dinge korrekt gehandhabt werden, wird in den dafür zuständigen Klassen geprüft. Im Moment kümmern wir uns also noch nicht darum, ob zum Beispiel die Anzeige der Schaltfläche im Beitrag auf der Website korrekt ist. Wir Testen hier nur diese Einheit, also die Methode onContentPrepare() unabhängig vom ganzen Programm!

Beim testgetriebenen Erstellen von Unittests haben sich in der Praxis die folgenden Regeln bewährt, die ich Ihnen hiermit ans Herz lege:

1. Prüfen Sie vor dem Hinzufügen einer neuen Funktion immer erst, ob alle schon vorhandenen Tests fehlerfrei durchlaufen werden und korrigieren Sie etwaig Fehler.
2. Implementieren Sie Tests, die die kleinste mögliche Einheit im Programmcode testen.
3. Implementieren Sie die Tests unabhängig von einander.
4. Geben Sie den Tests sprechende Namen damit diese gut wartbar und aussagekräftig sind und bleiben.

Die Implementierung des ersten Tests

Die Methoden in der generierten Testdatei sind noch leer. Dies ändern wir nun! Sie finden die Testdatei agpaypalTest.php, also die Datei mit den leeren Methoden, im Verzeichnis /joomla/tests/unit/suites/plugins/content/agpaypal/. Zumindest dann wenn Sie diese vorher während der Installation von Codeception im Kapitel *Codeception – ein Überblick* | *Codeception* | *Installation* mit mir erstellt haben. Öffnen Sie diese Datei bitte nun in Ihrer Entwicklungsumgebung zur Bearbeitung.

Für den ersten Test erstellen wir die Methode testOnContentPrepareIfSearchstringIsInContentOneTime(). Jetzt wird es endlich konkret! Sehen Sie sich zunächst einmal die fertige Methode an:

```
<?php
namespace suites\plugins\content\agpaypal;
class agpaypalTest extends \Codeception\Test\Unit {
   protected $tester;
   protected function _before() { }
   protected function _after() { }
   public function testOnContentPrepareIfSearchstringIsInContentOneTime()
   {
         require_once JINSTALL_PATH . '/libraries/joomla/event/dispatcher.php';
         $subject = \JEventDispatcher::getInstance();
         $config = array(
               'name' => 'agpaypal'",
               'type' => 'content',
               'params' => new \JRegistry
         );
         $params = new \JRegistry;
         require_once JINSTALL_PATH . '/plugins/content/agpaypal/agpaypal.php';
         $class = new \PlgContentAgpaypal($subject, $config, $params);
         $contenttextbefore = '<p>Texte vor der Schaltfläche.</p>'
         .'<p>@paypalpaypal@</p>'
         .'<p>Text hinter der Schaltfläche.</p>';
         $contenttextafter = '<p>Texte vor der Schaltfläche.</p>'
         .'<p><form name="_xclick" action="https://www.paypal.com/de/cgi-bin/webscr"
method="post">'
         .'<input type="hidden" name="amount" value="12.99">'
         .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-but01.gif"
```

```
border="0" name="submit">'
            .'</form></p>'
            .'<p>Text hinter der Schaltfläche.</p>';
            $class->onContentPrepare('', $contenttextbefore, $params);
            $this->assertEquals($contenttextafter, $contenttextbefore);
    }
}
```

Sehen wir uns den Programmcode der Testmethode genauer an: Ich haben für die Bedingung, dass der Text „@paypalpaypal@" zur Anzeige im Frontend in den Text der einen Jetzt-kaufen-Button anzeigt umgewandelt wird, eine Behauptung aufgestellt.

> **RANDBEMERKUNG**
> Zur Besseren Übersicht habe ich den auszutauschenden Test ein wenig gekürzt. Todo was habe ich genau gekürzt und prüfen ob das überall passt.

Diese Behauptung habe ich mithilfe der assertEquals() Methode formuliert. In dieser Methode musste ich die Parameter so setzten, dass meine Behauptungen erfüllt sind.

Die Variable $contenttextafter enthält den Text den ich erwarte und $contenttextbefore enthält den Eingabetext, also den Text, der zu Beginn den Text „@paypalpaypal@" enthält. Nach dem Durchlaufen der Methode onContentPrepare() aber gleich $contenttextafter sein sollte. Ich behaupte also assertEquals($contenttextafter, $contenttextbefore)**.** Den Rest übernimmt das PHPUnit-Framework.

Der Satz „Hier musste ich nur die Parameter so setzten, dass die Bedingungen erfüllt sind." hört sich einfacher als er ist.

- Als erste habe ich dafür ein Objekt des Typs PlgContentAgpaypal instanziiert und in der Variablen $class gespeichert. Dazu musste ich die Objekte $subject und $config erzeugen. Diese Objekte benötigen die Klasse PlgContentAgpaypal bei der Instantiierung.
- Die Objekte $params und $contenttextbefore habe ich danach erstellt, weil diese als Parameter in der Methode onContentPrepare() benötigt werden. $contenttextafter enthält den Text, in den $contenttextbefore umgewandelt werden soll.

- Nun habe die die zu testende Methode aufgerufen: $class->onContentPrepare('', $contenttextbefore, $params);
- Zum Schluss habe ich dann die im vorhergehenden Kapitel *Was wollen wir testen?* beschriebenen Bedingungen als Parameter in die Methode assertEquals() geschrieben. Diese Methode verifiziert die Gleichheit zweier Objekte.

So, nachdem nun klar ist was der Testprogrammcode genau macht führen wir diesen mithilfe des Befehls codecept run unit aus.

> **RANDBEMERKUNG**
> Da wir im Moment nur einen Test in unsere Suite haben, können wir problemlos die ganze Suite auf einen Schlag ausführen. Mit dem Befehl codecept run unit tests/unit/suites/plugins/content/agpaypal/agpaypalTest können Sie den Test auf die Testdatei agpaypalTest.php beschränken.

```
/var/www/html/joomla$ vendor/bin/codecept run unit
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
Unit Tests (1) ------------------------------------------------------------
✔ agpaypalTest: on content prepare if searchstring is in content one time (0.01s)
---------------------------------------------------------------------------
Time: 158 ms, Memory: 10.00MB
OK (1 test, 1 assertion)
```

Der Test war erfolgreich. Wurde er bei Ihnen auch als erfolgreich markiert? Dann probieren Sie doch einfach einmal was passiert, wenn Sie einen Buchstabendreher in die Variablen $contenttextbefore oder die Variable $contenttextafter einbauen. Dann müsste Ihnen nämlich ein Fehler angezeigt werden! Wenn der Test bei Ihnen nicht erfolgreich war, haben Sie wohl den Buchstabendreher schon irgendwo im Programmcode und sollten diesen nun suchen und korrigieren.

(todo wenn platz hier fehler abdrucken)

> Standardmäßig sagt Ihnen PHPUnit genau was fehlgeschlagen ist. Oft ist aber nicht direkt klar, warum dieser Fehler genau aufgetreten ist. Insbesondere dann, wenn Sie sehr viele aktive Tests haben, dieser eine fehlgeschlagene Test ganz

zu Beginn erstellt wurde und Sie sich nicht mehr recht erinnern.
Sie können sich das Leben leichter machen, wenn Sie Erläuterung in den Testanweisungen als Parameter mitgeben. Zum Beispiel könnten Sie folgende Anweisung verwenden:

```
$this->assertEquals($contenttextafter, $contenttextbefore, "@paypalpaypal@ wurde nicht richtig umgewandelt als er genau einmal im Text vorkam.");
```

Der Text "@paypalpaypal@ wurde nicht richtig umgewandelt als er genau einmal im Text vorkam." wird im zweiten Fall bei einem fehlgeschlagenen Test neben der Nachricht von PHPUnit mit ausgegeben. Dieser kleine Mehraufwand kann Ihnen auf lange Sicht bei der Testanalyse viel Zeit sparen.
In diesem Buch hier nutze ich diesen Parameter nicht. Die Codebeispiele sind ohne diesen Parameter kürzer und kompakter. (todo Beispiel für Ausgabe der Anmerkung)

Testgetrieben entwickeln

Wie sieht die testgetriebene Entwicklung nun in der Praxis aus? Ich möchte Sie dazu einladen mir einmal über die Schulter zu schauen. So können Sie einen einen ersten Eindruck und somit ein Gefühl für diese Methode bekommen. In der testgetriebenen Entwicklung schreiben Sie den Test, bevor Sie den eigentlichen Programmcode schreiben. Der Test ist also schon da, wenn Sie den Programmcode, der diesen Test erfüllt, erstellen. Sie starten mit einem ganz einfachen Design. Dieses Design verbessern Sie dann fortwährend. So werden komplizierte Designelemente, die nicht unbedingt notwendig sind, vermieden. Sie wählen gezwungenermaßen immer den einfachsten Weg!

Ein testgetriebenes Programmierbeispiel

Kommen wir auf unser Beispiel zurück. Bisher wird der Jetzt-kaufen-Button mit fixen Werten eingesetzt. Ihre nächste Aufgabe ist es nun, eine Möglichkeit zu schaffen, dieser Schaltfläche unterschiedliche Preise zuzuordnen. Okay, wie machen Sie das? Ich schlage vor, Sie geben dem auszutauschenden Text einen Parameter. Warum machen Sie nicht gleich alle Optionen des Buttons flexibel? Vermutlich wird diese Aufgabe später auf Sie zukommen. Weil wir in kleinen Schritten vorgehen. Im Moment geht es nur darum diese eine Option flexibel zu machen. Mehr ist momentan noch nicht gefordert. Beginnen wir mit der Implementation des Tests.

Erst der Test ...

Indem wir zuerst den Test für eine Funktion erstellen, können wir nach jeder Änderung am Code sicherstellen, dass alle geforderten Funktionen erfüllt sind. Was ist nun genau der nächste Testfall? Wir haben uns schon dazu entschieden dem Suchtest einen Parameter mitzugeben. Eine mögliche Umsetzung wäre, den Betrag in die Mitte des Suchtextes einzufügen. Zum Beispiel so: @paypal amount=15.00 paypal@. Im nachfolgenden Codebeispiel finden Sie die dazugehörige Testmethode. Die Variable $contenttextbefore enthält den Parameter im Suchtext und in der Variablen $contenttextafter wurde der Betrag ebenfalls angepasst.

```
<?php
namespace suites\plugins\content\agpaypal;
class agpaypalTest extends \Codeception\Test\Unit {
   protected $tester;
   protected function _before() {}
   protected function _after() {}
   public function testOnContentPrepareIfSearchstringIsInContentOneTimeAndContainsNoValue()
{…}
public function
testOnContentPrepareIfSearchstringIsInContentOneTimeAndContainsValueForAmount() {
        require_once JINSTALL_PATH . '/libraries/joomla/event/dispatcher.php';
        $subject = \JEventDispatcher::getInstance();
        $config = array(
                'name' => 'agpaypal"',
                'type' => 'content',
                'params' => new \Jregistry
        );
        $params = new \JRegistry;
        require_once JINSTALL_PATH . '/plugins/content/agpaypal/agpaypal.php';
        $class = new \PlgContentAgpaypal($subject, $config, $params);
        $contenttextbefore = '<p>Texte vor der Schaltfläche.</p>'
                .'<p>@paypal amount=15.00 paypal@</p>'
                .'<p>Text hinter der Schaltfläche.</p>';
        $contenttextafter = '<p>Texte vor der Schaltfläche.</p>'
                .'<p><form name="_xclick" action="https://www.paypal.com/de/cgi-bin/webscr"
method="post">'
                .'<input type="hidden" name="amount" value="15.00">'
```

```
                .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-but01.gif"
border="0" name="submit">'
                .'</form></p>'
                .'<p>Text hinter der Schaltfläche.</p>';
        $class->onContentPrepare('', $contenttextbefore, $params);
        $this->assertEquals($contenttextafter, $contenttextbefore);
    }
}
```

Dieser Test ist im Code der Produktivversion noch nicht umgesetzt und wird deshalb fehlschlagen. Überzeugen Sie selbst. Ich habe die Ausgabe der Kommandozeile im nächsten Kasten abgedruckt.

```
/var/www/html/joomla$ vendor/bin/codecept run unit
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
Unit Tests (2) ------------------------------------------------------
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains no value (0.00s)
✖ agpaypalTest: On content prepare if searchstring is in content one time and contains value for amount (0.00s)
------------------------------------------------------
Time: 144 ms, Memory: 10.00MB
There was 1 failure:
…
FAILURES!
Tests: 2, Assertions: 2, Failures: 1.
```

… dann die Funktion

Als nächstes geht es nun darum, den Test zu erfüllen – und zwar auf möglichst einfache Art und Weise. Sicherlich denken Sie schon weiter. Sie vermuten wahrscheinlich, dass alle Werte die mit dem Jetzt-kaufen-Button mitgegeben werden, manipulierbar sein sollen. Wahrscheinlich haben Sie auch hierfür schon eine Lösung im Kopf. Komplizierte Designelemente, die nicht unbedingt notwendig sind, sollen Sie aber aufschieben. Sie programmieren nur das, was Sie tatsächlich jetzt gerade benötigen. Nicht das, was später vielleicht notwendig ist.

Dazu suchen wir die Texte @paypal und paypal@ mithilfe der Methode preg_match_all(). preg_match_all() speichert alle Funde in der Variablen $matches[0][0] und der Suchtext (.*?) extrahieren den Text zwischen @paypal und paypal@ in der Variablen $matches[1][0]. Nun prüfende wir, ob $matches[1][0] den Text amount= enthält. Falls ja, belegen wir den auszutauschenden Suchtext mit $matches[0][0] und passen den Wert für den Preis an. Falls nicht lassen wir alles beim Alten. Dies hat zudem keine negativen Auswirkungen auf den schon bestehenden Testfall. Dieser bleibt weiterhin erfüllt.

(todo preg_match_all ausführlicher?)

```
<?php
defined('_JEXEC') or die;
class PlgContentAgpaypal extends JPlugin {
 public function onContentPrepare($context, &$row, $params, $page = 0) {
        if (is_object($row)) {
                preg_match_all('/@paypal(.*?)paypal@/i', $row->text, $matches,
PREG_PATTERN_ORDER);
        }
        else {
                preg_match_all('/@paypal(.*?)paypal@/i', $row, $matches,
PREG_PATTERN_ORDER);
        }
        if (count($matches[0] > 0) && (strpos($matches[1][0], 'amount=') !== false) > 0) {
                $search = $matches[0][0];
                $amount = trim(str_replace('amount=', '', $matches[1][0]));
        }
        else {
                $search = "@paypalpaypal@";
                $amount ='12.99';
        }
        $replace = '<form name="_xclick" action="https://www.paypal.com/de/cgi-bin/webscr"
method="post">'
                .'<input type="hidden" name="amount" value="' . $amount . '">'
                .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-but01.gif"
border="0" name="submit">'
                .'</form>';
        if (is_object($row)) {
                $row->text = str_replace($search, $replace, $row->text);
```

```
            }
            else {
                    $row = str_replace($search, $replace, $row);
            }
            return true;
    }
}
```

Probieren Sie es aus! Was passiert wenn Sie nun den Befehl vendor/bin/codecept run unit in die Kommandozeile eingeben? Die beiden Testfälle sind erfüllt.

```
/var/www/html/joomla$ vendor/bin/codecept run unit
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
Unit Tests (2) ------------------------------------------------------------
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains no value (0.01s)
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains value for amount (0.00s)
------------------------------------------------------------------------
Time: 164 ms, Memory: 10.00MB
OK (2 tests, 2 assertions)
```

Das ist Ihnen zu einfach? Dann treten Sie den Beweis an. Schreiben Sie einen weiteren Test, der dies belegt. Zum Beispiel einen Test zum Suchtext @paypal **aomunt**=15.00 paypal@. Wie wollen Sie damit umgehen, wenn derjenige, der den Jetzt-kaufen-Button einfügen will, sich vertippt und anstelle von amount das Wort aomunt eingibt? Eine Möglichkeit sehen wir uns im nächsten Kapitel an.

Und weiter – erst testen, dann programmieren

Im Zusammenspiel von Testen und Implementieren wird das Programm immer weiter entwickelt. Die Software wird somit in ganz kleinen Schritten geschrieben. Der vorhergehende Testfall wurde erfüllt. Nun gibt es eine neue Aufgabe.

Die Aufgabe ergibt sich meist aus der Spezifikation, die noch nicht ganz erfüllt ist - das Programm also noch nicht ganz fertig ist. Es kann aber auch sein, dass Sie mit dem bisherigen Stand noch nicht froh sind – Ihnen die bisherige Lösung nicht gut

genug ist. In unserm Beispiel werden zwar die Testfälle erfüllt. Es gibt aber weitere Testfälle die ein unvorhergesehenes Programmverhalten zeigen. Ein Beispiel ist die Eingabe eines Suchtextes, bei dem ein Tippfehler vorliegt. In diesem Falle würde Ihr Plugin nicht greife. Das Problem ist, dass der auszutauschende Text nur mit einem Parameter amount richtig gesetzt wird. Falls amount nicht vorkommt, würde nur ein Text in der Form @paypalpaypal@ ersetzt. Ich beweise Ihnen dieses mit dem folgenden Test.

```
   …
public function
testOnContentPrepareIfSearchstringIsInContentOneTimeAndContainsIncorrectValue() {
        require_once JINSTALL_PATH . '/libraries/joomla/event/dispatcher.php';
        $subject = \JeventDispatcher::getInstance();
        $config = array(
                'name' => 'agpaypal'",
                'type' => 'content',
                'params' => new \Jregistry
        );
        $params = new \Jregistry;
        require_once JINSTALL_PATH . '/plugins/content/agpaypal/agpaypal.php';
        $class = new \PlgContentAgpaypal($subject, $config, $params);
        $contenttextbefore = '<p>Texte vor der Schaltfläche.</p>'
                .'<p>@paypal aomunt=15.00 paypal@</p>'
                .'<p>Text hinter der Schaltfläche.</p>';
        $contenttextafter = '<p>Texte vor der Schaltfläche.</p>'
                .'<p><form name="_xclick" action="https://www.paypal.com/de/cgi-bin/webscr"
method="post">'
                .'<input type="hidden" name="amount" value="15.00">'
                .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-but01.gif"
border="0" name="submit">'
                .'</form></p>'
                .'<p>Text hinter der Schaltfläche.</p>';
        $class->onContentPrepare('', $contenttextbefore, $params);
        $this->assertEquals($contenttextafter, $contenttextbefore);
  }
...
```

Wenn zwischen @paypal und paypal@ Text steht, das Wort amount= aber nicht in diesem Text enthalten ist, wird kein Jetzt-kaufen-Button eingefügt. Der Text schlägt fehl:

```
/var/www/html/joomla$ vendor/bin/codecept run unit
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
Unit Tests (3) ------------------------------------------------------------
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains no value (0.01s)
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains value for amount (0.00s)
✖ agpaypalTest: On content prepare if searchstring is in content one time and contains incorrect value (0.00s)
--------------------------------------------------------------------------------
Time: 168 ms, Memory: 10.00MB
There was 1 failure:
….
FAILURES!
Tests: 3, Assertions: 3, Failures: 1.
```

Eine kleine Änderung bringt hier die Lösung. Falls Text zwischen @papyal und paypal@ enthalten ist, setzten wir den auszutauschenden Text in jedem Fall gleich $matches[0][0]. Den Preis belassen wir auf den Standardwert. Es gibt keine Regel aus der wir herleiten können, welche Tippfehler genau den Parameter amount meinen. Idealerweise erweitern wir das Plugin später ja auch so, dass gar keine Tippfehler mehr vorkommen können ….

```
<?php
defined('_JEXEC') or die;
class PlgContentAgpaypal extends JPlugin {
 public function onContentPrepare($context, &$row, $params, $page = 0) {
        if (is_object($row)) {
                preg_match_all('/@paypal(.*?)paypal@/i', $row->text, $matches,
PREG_PATTERN_ORDER);
        }
        else {
```

```
                preg_match_all('/@paypal(.*?)paypal@/i', $row, $matches,
PREG_PATTERN_ORDER);
        }
        if (count($matches[0] > 0)) {
                $search = $matches[0][0];
                if ((strpos($matches[1][0], 'amount=') !== false) > 0) {
                        $amount = trim(str_replace('amount=', '', $matches[1][0]));
                }
                else {
                        $amount ='12.99';
                }
        }
        else {
                $search = "@paypalpaypal@";
                $amount ='12.99';
        }
                $replace = '<form name="_xclick" action="https://www.paypal.com/de/cgi-
bin/webscr" method="post">'
                        .'<input type="hidden" name="amount" value="' . $amount . '">'
                        .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-
but01.gif" border="0" name="submit">'
                        .'</form>';
        if (is_object($row)) {
                $row->text = str_replace($search, $replace, $row->text);
        }
        else {
                $row = str_replace($search, $replace, $row);
        }
        return true;
 }
}
```

Ein erneuter Testdurchlauf zeigt, dass nun alle drei bisherigen Tests erfolgreich sind.

```
/var/www/html/joomla$ vendor/bin/codecept run unit
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
```

```
Unit Tests (3) ----------------------------------------------------------------
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains no value (0.00s)
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains value for amount (0.00s)
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains incorrect value (0.00s)
---------------------------------------------------------------------------
Time: 116 ms, Memory: 10.00MB
OK (3 tests, 3 assertions)
```

Schön. Das beruhigt. Auch wenn es zugegebenermaßen noch viele Baustellen in dieser Joomla! Erweiterung gibt.

Weiter geht es Schritt für Schritt

Machen wir mit der nächsten Baustelle weiter. Laut Spezifikation kann es vorkommen, das in einem Text mehrere Jetzt-kaufen-Button vorkommen. Was sagt ein Test dazu?

```
    …
public function testOnContentPrepareIfSearchstringIsInContentManyTimes() {
        require_once JINSTALL_PATH . '/libraries/joomla/event/dispatcher.php';
        $subject = \JEventDispatcher::getInstance();
        $config = array(
                'name' => 'agpaypal"',
                'type' => 'content',
                'params' => new \Jregistry
        );
        $params = new \Jregistry;
        require_once JINSTALL_PATH . '/plugins/content/agpaypal/agpaypal.php';
        $class = new \PlgContentAgpaypal($subject, $config, $params);
        $contenttextbefore = '<p>Texte vor der Schaltfläche.</p>'
                .'<p>@paypal amount=16.00 paypal@</p>'
                .'<p>Text hinter der Schaltfläche.</p>'
                .'<p>Texte vor der Schaltfläche.</p>'
                .'<p>@paypal amount=15.00 paypal@</p>'
                .'<p>Text hinter der Schaltfläche.</p>';
        $contenttextafter = '<p>Texte vor der Schaltfläche.</p>'
```

```
                .'<p><form name="_xclick" action="https://www.paypal.com/de/cgi-
bin/webscr" method="post">'
                .'<input type="hidden" name="amount" value="16.00">'
                .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-but01.gif"
border="0" name="submit">'
                .'</form></p>'
                .'<p>Text hinter der Schaltfläche.</p>'
                .'<p>Texte vor der Schaltfläche.</p>'
                .'<p><form name="_xclick" action="https://www.paypal.com/de/cgi-
bin/webscr" method="post">'
                .'<input type="hidden" name="amount" value="15.00">'
                .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-but01.gif"
border="0" name="submit">'
                .'</form></p>'
                .'<p>Text hinter der Schaltfläche.</p>';
        $class->onContentPrepare('', $contenttextbefore, $params);
        $this->assertEquals($contenttextafter, $contenttextbefore);
    }
```

Mehrere Jetzt-kaufen-Button unterstützt das Plugin in der aktuellen Version leider noch nicht. Im nächsten Kasten sehen Sie, dass der Test fehlschlägt!

```
/var/www/html/joomla$ vendor/bin/codecept run unit
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
Unit Tests (4) ------------------------------------------------------------
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains no value
(0.00s)
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains value for
amount (0.00s)
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains incorrect value
(0.00s)
✖ agpaypalTest: On content prepare if searchstring is in content many times (0.00s)
…
FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

Was müssen wir tun, um diese Forderung zu erfüllen? Die Funktion preg_match_all() liefert uns alle Textstellen, die mit @paypal beginnen und mit paypal@ enden. Diese können wir in einer Schleife durchlaufen. Dabei können wir alles das, was wir bisher an Programmcode geschrieben haben, verwenden.

```
<?php
defined('_JEXEC') or die;
class PlgContentAgpaypal extends JPlugin {
 public function onContentPrepare($context, &$row, $params, $page = 0) {
        if (is_object($row)) {
                preg_match_all('/@paypal(.*?)paypal@/i', $row->text, $matches,
PREG_PATTERN_ORDER);
        }
        else {
                preg_match_all('/@paypal(.*?)paypal@/i', $row, $matches,
PREG_PATTERN_ORDER);
        }
        if (count($matches[0] > 0)) {
                for ($i = 0; $i < count($matches [0]); $i++){
                        $search = $matches[0][$i];
                        if ((strpos($matches[1][$i], 'amount=') !== false) > 0){
                                $amount = trim(str_replace('amount=', '', $matches[1][$i]));
                        }
                        else{
                                $amount ='12.99';
                        }
                        $replace = '<form name="_xclick" action="https://www.paypal.com/de/cgi-
bin/webscr" method="post">'
                                .'<input type="hidden" name="amount" value="' . $amount . '">'
                                .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-
click-but01.gif" border="0" name="submit">'
                                .'</form>';
                        if (is_object($row)) {
                                $row->text = str_replace($search, $replace, $row->text);
                        }
                        else {
                                $row = str_replace($search, $replace, $row);
                        }
```

```
                }
            }
            return true;
    }
}
```

Und das war es auch schon. Alle Tests sind nun erfolgreich.

```
/var/www/html/joomla$ vendor/bin/codecept run unit
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
Unit Tests (4) ------------------------------------------------------------
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains no value (0.00s)
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains value for amount (0.00s)
✔ agpaypalTest: On content prepare if searchstring is in content one time and contains incorrect value (0.00s)
✔ agpaypalTest: On content prepare if searchstring is in content many times (0.00s)
---------------------------------------------------------------------------------
Time: 152 ms, Memory: 10.00MB
OK (4 tests, 4 assertions)
```

Geht das noch einfacher? Gibt es Redundanzen?

Wir haben bei allen Schritten bisher versucht die einfachste Lösung umzusetzen. Manchmal ist der erste Versuch aber nicht der optimale. Wir wollen aber eine gute Lösung umsetzten. Das bedeutet, dass wir zusätzlich fortlaufend auch immer mal wieder einen Blick auf das Design werfen sollten. Auch dieses sollten wir beständig fortentwickeln und verbessern. Der Code sollte beständig so einfach und verständlich wie möglich bleiben. Redundanzen sollten im Keim erstickt werden. Prüfen Sie Ihren Programmcode nach jedem Schritt und bringen ihn wenn möglich in eine einfachere Form. Kommt Ihnen dies zu umständlich vor? Vertrauen Sie darauf, dass gut strukturierter Programmcode während der ganzen Entwicklungsphase auch gut erweiterbar ist. (Todo Was habe ich gemacht)

```
<?php
defined('_JEXEC') or die;
class PlgContentAgpaypal extends JPlugin {
 public function onContentPrepare($context, &$row, $params, $page = 0)
 {
        if (is_object($row)) {
                $this->startCreateButtons($row->text);
        }
        else {
                $this->startCreateButtons($row);
        }
        return true;
}
 private function startCreateButtons(&$text) {
        preg_match_all('/@paypal(.*?)paypal@/i', $text, $matches, PREG_PATTERN_ORDER);
        if (count($matches[0] > 0)) {
                $this->createButtons($matches, $text);
        }
        return true;
}
 private function createButtons($matches, &$text) {
        for ($i = 0; $i < count($matches[0]); $i++) {
                $search = $matches[0][$i];
                if ((strpos($matches[1][$i], 'amount=') !== false) > 0) {
                        $amount = trim(str_replace('amount=', '', $matches[1][$i]));
                }
                else {
                        $amount ='12.99';
                }
                $replace = '<form name="_xclick" action="https://www.paypal.com/de/cgi-
bin/webscr" method="post">'
                        .'<input type="hidden" name="amount" value="' . $amount . '">'
                        .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-
but01.gif" border="0" name="submit">'
                        .'</form>';
                $text = str_replace($search, $replace, $text);
        }
        return true;
```

```
    }
}
```

Fällt Ihnen etwas auf? Wir haben eine neue Methode erstellt. Diese Änderung hat auch Auswirkungen auf unseren Test. Wir möchten ja immer die kleinste Einheit testen. Unser aktueller Testaufbau testet aber nun das Zusammenspiel mehrerer Methoden der Klasse AgpaypalTest. Wir sollten den Test so abändern, das jede Methode für sich getestet wird. Fangen wir mit der Methode testStartCreateButtonsIfSearchstringIsInContentOneTimeAndContainsNoValue() an. (todo private Methode) Die Methode contentPrepare() heben wir uns für später auf. An ihr sehen wir uns später Testduplikate genauer an.

```
<?php
namespace suites\plugins\content\agpaypal;
class agpaypalTest extends \Codeception\Test\Unit {
   protected $tester;
   protected function _before() { }
   protected function _after() { }
   public function testStartCreateButtonsIfSearchstringIsInContentOneTimeAndContainsNoValue() {
        require_once JINSTALL_PATH . '/libraries/joomla/event/dispatcher.php';
        $subject = \JeventDispatcher::getInstance();
        $config = array(
                'name' => 'agpaypal'",
                'type' => 'content',
                'params' => new \JRegistry
        );
        $params = new \JRegistry;
        require_once JINSTALL_PATH . '/plugins/content/agpaypal/agpaypal.php';
        $class = new \PlgContentAgpaypal($subject, $config, $params);
        $contenttextbefore = '<p>Texte vor der Schaltfläche.</p>'
                .'<p>@paypalpaypal@</p>'
                .'<p>Text hinter der Schaltfläche.</p>';
        $contenttextafter = '<p>Texte vor der Schaltfläche.</p>'
                .'<p><form name="_xclick" action="https://www.paypal.com/de/cgi-bin/webscr"
method="post">'
                .'<input type="hidden" name="amount" value="12.99">'
                .'<input type="image" src="http://www.paypal.com/de_DE/i/btn/x-click-but01.gif"
```

```
border="0" name="submit">'
                    .'</form></p>'
                    .'<p>Text hinter der Schaltfläche.</p>';
            $class->startCreateButtons($contenttextbefore);
            $this->assertEquals($contenttextafter, $contenttextbefore);
    }
….
```

Nach dem Umstellen sollten die Tests weiterhin alle erfüllt sein. Im nächsten Kasten sehen Sie, dass dies auch so ist.

```
/var/www/html/joomla$ vendor/bin/codecept run unit
Codeception PHP Testing Framework v2.2.9
Powered by PHPUnit 5.7.13 by Sebastian Bergmann and contributors.
Unit Tests (5) ------------------------------------------------------------------
- agpaypalTest: Start create buttons if searchstring is in content one time and co✔ agpaypalTest: Start
create buttons if searchstring is in content one time and contains no value (0.00s)
- agpaypalTest: Start create buttons if searchstring is in content one time and co✔ agpaypalTest: Start
create buttons if searchstring is in content one time and contains valuefor amount (0.00s)
- agpaypalTest: Start create buttons if searchstring is in content one time and co✔ agpaypalTest: Start
create buttons if searchstring is in content one time and contains incorrect value (0.00s)
✔ agpaypalTest: Start create buttons if searchstring is in content many times (0.00s)
✔ agpaypalTest: Start create buttons if searchstring is in content no time (0.00s)
------------------------------------------------------------------------------
Time: 149 ms, Memory: 10.00MB
OK (5 tests, 5 assertions)
```

Das Plugin ist nun wieder übersichtlich und den Testcode haben wir auch angepasst. Es gibt noch viel zu tun. Welche Funktion wollen Sie als nächstes in Angriff nehmen Beginnen Sie wieder mit dem Test ….

Hier im Buch packen wir die Arbeit nicht an. Wir implementieren keine neuen Funktionen in das Plugin. Hier geht es ja nicht in erster Linie um die Programmierung von Joomla! Erweiterungen – obwohl es dazu auch viel zu schreiben gäbe. Vielmehr legen wir den Schwerpunkt auf die Verbesserung der Tests. Wie testen wir besser und welche Möglichkeiten stehen uns zur Verfügung?