MARTIN DUSCHEK, 67664, 16MI1-B

EVOLUTION VON CODE BEI MAJOR-RELEASES VON PROGRAMMIERSPRACHEN

EVOLUTION VON CODE BEI MAJOR-RELEASES VON PROGRAMMIERSPRACHEN

am Beispiel der Migration zu PHP7

MARTIN DUSCHEK, 67664, 16MI1-B

I-ITWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

November 2019

[30. September 2019 at 12:08 – classicthesis v4.6]

Martin Duschek, 67664, 16MI1-B: *Evolution von Code bei Major-Releases von Programmiersprachen*, am Beispiel der Migration zu PHP7, © November 2019

INHALTSVERZEICHNIS

LISTINGS

ABKÜRZUNGSVERZEICHNIS

PCRE	Perl Compatible Regular Expressions
PHP	PHP: Hypertext Preprocessor

Teil I

EINLEITUNG

1 EINLEITUNG

Am 03. Dezember 2015 erschien mit PHP 7.0.0 das erste Major-Release seit elf Jahren. Damit einhergehend wurde die Einstellung der Weiterentwicklung der vorhergehenden Version 5 für den 10. Januar 2019 angekündigt. Der Entwicklungsstopp führt dazu, dass Sicherheitslücken in der Implementation der alten Version nicht mehr geschlossen werden, was wiederum dazu führt, dass bereits ausgelieferte Software angreifbar wird sobald neue Lücken gefunden werden.
Derzeit setzen 79,1% der 10 Millionen meistgenutzten Webseiten PHP als serverseitige Programmiersprache ein, davon 61,5% PHP in der veralteten Version 5[1]. Diese Installationen können allesamt als unsicher eingestuft werden. Seit der letzten Veröffentlichung unter Version 5 wurden vier neue Schwachstellen veröffentlicht[2], die in unterstützten Versionen bereits geschlossen wurden.

1.1 MOTIVATION

Bei der Migration des Onlineshops der Firma *Tickets75*

1.2 AUFGABENSTELLUNG

Ziel dieser Arbeit ist die Evaluiation verschiedener Techniken und Technologien, die ein Upgrade der Programmiersprache in Softwareprojekten einfacher und nachhaltig gestalten oder erst in effizienter Weise ermöglichen. Dabei wird die Migration eines Onlineshops von PHP 5.6 zu PHP 7.0 als praktisches Beispiel herangezogen und die verschiedenen Ansätze geprüft. Als Leitfaden dient der Internationale Standard **ISO/IEC 14764**.

1.3 AUFBAU

1 W3Techs, „Usage statistics of PHP for websites“, https://w3techs.com/technologies/details/pl-php/all/all

2 CVE details, „PHP 5.6.40 Security Vulnerabilities“, https://www.cvedetails.com/vulnerability-list/vendor_id-74/product_id-128/version_id-298516/PHP-PHP-5.6.40.html

2 GRUNDLAGEN

2.1 SOFTWAREWARTUNG NACH ISO/IEC 14764

Der Standard **ISO/IEC 14764** beschreibt den Prozess der Wartung von Software bis zu deren Einstellung. Darin wird unter Anderem beschrieben, welche Schritte bei der Migration von Software zu befolgen sind, sobald diese an eine neue Umgebung angepasst werden muss. Folgende Aktionen sind durch den Ausführenden nach **ISO/IEC 14764** umzusetzen:

- Analyse der Anforderungen und Definition der Migration
- Entwicklung von Werkzeugen zur Migration
- Entwicklung der an die neue Ugebung angepassten Software
- Durchführung der Migration
- Verifikation der Migration
- Support der alten Umgebung

2.2 PHP

PHP: Hypertext Preprocessor (PHP) ist eine Skriptsprache, welche seit 1994 entwickelt wird und seit 1995 Open-Source bereitgestellt wird. Obwohl **PHP** viele Einsatzzwecke abdeckt, wird es hauptsächlich dazu genutzt, dynamische Websites zu programmieren.

2.3 VERSIONIERUNG

Teil II

PRAKTIKUMSBERICHT

3 ANFORDERUNGSANALYSE

Dieser Abschnitt soll beleuchten, welche Bedingungen PHP in Version 7 gegenüber Version 5 an lauffähige Software stellt. Zudem werden die Änderungen in den Kontext der zeitlichen Entwicklung gestellt, um Aussagen über Gründe dieser zu treffen.

3.1 ABWÄRTSINKOMPATIBLE ÄNDERUNGEN

Änderungen in dieser Kategorie führen in älteren Versionen zu Fehlern oder unerwartetem Verhalten und sind in dieser Umgebung somit nicht lauffähig. Durch diese wird ein Wechsel der Umgebung zwingend vorrausgesetzt.

3.2 VERALTETE FUNKTIONEN

Als veraltet markierte Funktionen sind in der neuen Umgebung zwar noch unterstützt, sollten aber nach Möglichkeit nicht mehr eingesetzt und schnellstmöglich durch geeignete Funktionen ersetzt werden, da sie möglicherweise in zukünftigen Versionen entfernt oder verändert werden. Werden diese Funktionen trotzdem eingesetzt, wird eine Warnung ausgegeben, die Programmierer darauf hinweisen soll, dass die Verwendung der Funktion möglicherweise gefährlich sein kann. Die Lauffähigkeit des Programms wird bis zur abschließenden Entfernung der Funktion jedoch nicht beeinflusst. [2]

3.2.1 *Implizite Benennung von Konstruktoren*

Mit der Einführung der objektorientierten Programmierung in PHP 4 wurde festgelegt, dass Funktionen mit dem selben Namen wie die umschließende Klasse implizit als Konstruktor der Klasse erkannt werden. Ein Beispiel zur Implementierung eines Konstruktors nach diesem Prinzip ist in Listing 3.1 dargestellt. PHP 7 unterstützt diese Notation zwar noch, allerdings wird die, in PHP 5 eingeführte, explizite Benennung mit dem Schlüsselwort *__construct* (siehe Listing 3.2) bevorzugt. Hierdurch soll die Verwirrung darum, wann eine Funktion einen Konstruktor darstellt aufgehoben werden. [1]

Listing 3.1: Beispiel eines impliziten Konstruktors

```
<?php
class foo {
    function foo($a) {
```

```
        echo("Created instance of class 'foo'");
    }
}
?>
```

Listing 3.2: Beispiel eines expliziten Konstruktors

```
<?php
class foo {
    function __construct($a) {
        echo("Created instance of class 'foo'");
    }
}
?>
```

3.2.2 *Statische Aufrufe nicht-statischer Funktionen*

Mit dem Schlüsselwort *static* versehene Funktionen einer Klasse erlauben das Benutzen der Funktion, ohne die Instantiierung der Klasse selber. Damit steht die entsprechende Funktion nicht im Kontext eines Objekts, sondern im Kontext der entsprechenden Klasse. Im Gegensatz zu anderen objektorientierten Programmiersprachen (bspw. Java) war es in PHP bisher möglich, auch nicht-statische Methoden ohne eine Instantiierung zu verwenden. Diese Möglichkeit wurde mit PHP 7 für veraltet erklärt und sollte nicht mehr genutzt werden. Dadurch werden Programmierfehler verhindert, da der Kontext, in dem eine Funktion ausgeführt wird nun Eindeutig ist. Das Beispiel 3.3 wird eine Warnung ausgeben, dass eine nicht-statische Methode statisch aufgerufen wird.

Listing 3.3: Beispiel eines statischen Aufrufs einer nicht-satischen Funktion in PHP 7

```
<?php
class foo {
    function bar() {
        echo("'bar' is not a static function");
    }
}

foo::bar();
?>
```

3.3 GEÄNDERTE FUNKTIONEN

In diese Gruppe fallen Funktionen, deren Benutzung und/oder Verhalten geändert wurden, allerdings nicht vollständig veraltet sind. Dies bedeutet zum Beipiel, dass einzelne Funktionsparameter entfernt wurden oder andere Datentypen zurückgegeben werden.

3.3.1 *preg_replace*

Die Funktion *preg_replace()* ersetzt Teile einer Zeichenkette nach einem, als regulärem Ausdruck angegebenen, Muster. Mit *PCRE-Modifikatoren* kann die Verhaltensweise des regulären Ausdrucks gesteuert werden. In PHP 7 wurde der Modifikator
e entfernt

3.4 NEUE FUNKTIONEN

4 UNTERSUCHUNG GEEIGNETER MITTEL

4.1 LAUFFÄHIGKEIT HISTORISCHEN CODES

4.1.1 *Codeverwaltung*

4.1.2 *Lokale Entwicklungsumgebung*

4.1.3 *Continous Integration mittels Containern*

4.2 ERKENNUNG DES ZU ÄNDERNDEN CODES

Um alten Code migrieren zu können, müssen alle Stellen gefunden werden, die in ihrer ursprünglichen Form in der neuen Umgebung nicht lauffähig wären.

4.2.1 *Manuelle Erkennung*

4.2.2 *Automatisierte Erkennung*

4.3 REFACTORING

4.3.1 *Unit-Tests*

4.3.2 *Search & Replace*

4.3.3 *Wrapping*

LITERATUR

[1] Morrison Levi. *PHP: rfc:remove_php4_constructors*. 17. Nov. 2014. URL: https://wiki.php.net/rfc/remove_php4_constructors (besucht am 30. 09. 2019).

[2] Oracle. *How and When to Deprecate APIs*. 2004. URL: https://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/deprecation/deprecation.html (besucht am 30. 09. 2019).