

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

H-TWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

November 2019

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

INHALTSVERZEICHNIS

I EINLEITUNG

1	EINLEITUNG	2
1.1	Motivation	2
1.2	Aufgabenstellung	2
1.3	Aufbau	2
2	GRUNDLAGEN	3
2.1	Softwarewartung nach ISO/IEC 14764	3
2.2	PHP	3
2.3	Versionierung	3

II PRAKTIKUMSBERICHT

3	ANFORDERUNGSANALYSE	5
3.1	Abwärtsinkompatible Änderungen	5
3.2	Veraltete Funktionen	5
3.2.1	Implizite Benennung von Konstruktoren	5
3.2.2	Statische Aufrufe nicht-statischer Funktionen	6
3.3	Geänderte Funktionen	6
3.3.1	preg_replace	7
3.3.2	setlocale	7
3.4	Neue Funktionen	7
3.4.1	Anonyme Klassen	7
3.4.2	preg_replace_callback_array()	8
3.4.3	Typdeklaration für Rückgabewerte	8
3.5	Entfernte Erweiterungen	8
3.5.1	mysql	8
3.5.2	ereg	9
4	UNTERSUCHUNG GEEIGNETER MITTEL	10
4.1	Lauffähigkeit historischen Codes	10
4.1.1	Codeverwaltung	10
4.1.2	Lokale Entwicklungsumgebung	10
4.1.3	Continuous Integration mittels Containern	10
4.2	Erkennung des zu ändernden Codes	10
4.2.1	Manuelle Erkennung	10
4.2.2	Automatisierte Erkennung	10
4.3	Refactoring	10
4.3.1	Unit-Tests	10
4.3.2	Search & Replace	10
4.3.3	Wrapping	10

LITERATUR	11
-----------	----

LISTINGS

Listing 3.1	Beispiel eines impliziten Konstruktors	5
Listing 3.2	Beispiel eines expliziten Konstruktors	6
Listing 3.3	Beispiel eines statischen Aufrufs einer nicht-statischen Funktion in PHP 7	6
Listing 3.4	Beispiel der Nutzung von preg_replace mit dem Modifikator /e	7
Listing 3.5	Beispiel der Nutzung anonymer Klassen	8

ABKÜRZUNGSVERZEICHNIS

PCRE	Perl Compatible Regular Expressions
PHP	PHP: Hypertext Preprocessor

Teil I

EINLEITUNG

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 5¹. Diese Installationen können allesamt als unsicher eingestuft werden. Seit der letzten Veröffentlichung unter Version 5 wurden vier neue Schwachstellen veröffentlicht², 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 Evaluierung 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

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 Umgebung 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

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 vorausgesetzt.

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. [Ora04]

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. [Mor14a]

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-statischen 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 Beispiel, 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, mit dem die Zeichenkette durch das Ergebnis einer Funktion ersetzt wird. Ein Beispiel ist die Umwandlung aller kleingeschriebenen Zeichen eines Strings in Großbuchstaben, dargestellt in Listing 3.4. Die Verwendung des Modifikators wird aufgrund der Maskierungsregeln für bestimmte Zeichen als sehr kompliziert beschrieben. Gleichzeitig stellt die einfache Art der Evaluierung des Ergebnisses keine Schutzmechanismen zur Verfügung, wodurch Sicherheitslücken entstehen können, sobald es einem Angreifer gelingt, ausführbaren Code in diese Funktion einzuschleusen.

Listing 3.4: Beispiel der Nutzung von *preg_replace* mit dem Modifikator */e*

```
<?php
$uppercase = preg_replace(
    "/([a-z]*)/e",
    "strtoupper($1)",
    $mixedCase
);
?>
```

3.3.2 *setlocale*

Die Funktion *setlocale()* dient dazu, regionale Eigenheiten abzubilden. Dazu gehören zum Beispiel unterschiedliche Datumsformate oder die Formatierung von Zahlen (bspw. Trennzeichen für Dezimalzahlen). Für die Einstellung einer Region können Kategorien angegeben werden, auf die sich die Änderung auswirken soll. Ab Version 7 ist es nicht mehr möglich, die Kategorie als Zeichenkette anzugeben. Für diese Änderung ist kein Grund angegeben, allerdings liegt die Vermutung nahe, dass sich dadurch die Prüfung der Kategorie innerhalb der Funktion vereinfachen lässt, da *PHP* verschiedene benannte Konstanten zur Anwendung zur Verfügung stellt. Dies lässt sich auch durch die Historie der betreffenden Funktion im Quellcode belegen, durch die ersichtlich wird, dass ein großer Teil der Überprüfung der Funktionsparameter entfernt wurde. [nik14]

3.4 NEUE FUNKTIONEN

3.4.1 *Anonyme Klassen*

Mit dem Hinzufügen von anonymen Klassen implementiert *PHP* ein Konzept, das bereits aus anderen Objektorientierten Sprachen, bei-

spielsweise Java [Oraa], bekannt ist. Diese können benutzt werden, um gleichzeitig mit der Definition eine einmalig genutzte Klasse zu instanziiieren, ohne eigens dafür eine neue lokale Klasse erstellen zu müssen., wie in Listing 3.5 dargestellt wird.

Listing 3.5: Beispiel der Nutzung anonymer Klassen

```
<?php
$foo = new class {
    public function bar() {
        echo "Hello World";
    }
};

$foo->bar();
?>
```

3.4.2 *preg_replace_callback_array()*

Ähnlich wie die im Abschnitt 3.3.1 beschriebene Funktion *preg_replace()* mit dem Modifikator */e*, ersetzt *preg_replace_callback_array()* Zeichenketten anhand eines Musters und einer Ersetzungsfunktion. Im eingeführten *preg_replace_callback_array()* kann nun ein assoziatives Array angegeben werden, das mehrere Muster und ihre entsprechenden Callback-Funktionen enthält. Durch die Nutzung verschiedener Ersetzungsfunktionen kann auf die Nutzung einer einzelnen, stark verzweigten Ersetzungsfunktion verzichtet werden. Dadurch wird entsprechender Quellcode lesbarer und besser wartbar (vgl. [Mar12, S. 34f]).

3.4.3 *Typdeklaration für Rückgabewerte*

Als schwach typisierte Sprache bot PHP bisher keine Möglichkeit der Deklaration von Typen für Rückgabewerte von Funktionen. Dies kann nun durch Angabe des Typs zwischen Funktionsdeklaration und dem Code der Funktion geschehen. Dadurch sollen unter anderem ungewollte Rückgabewerte verhindert werden, als auch die automatisierte Dokumentation von Funktionen vereinfacht werden. [Mor14b]

3.5 ENTFERNT ERWEITERUNGEN

3.5.1 *mysql*

Die seit PHP 5 als veraltet erklärte Erweiterung *mysql* wird nicht mehr unterstützt. Dies wird mit Sicherheitsrisiken begründet. So unterstützt *mysql* beispielsweise keine **Prepared Statements**, welche einen wirk-

samen Schutz gegen **SQL Injections** bieten. [Orab] Zudem stehen mit *mysqli* und *PDO* aktuellere Erweiterungen zur Verfügung.

3.5.2 *ereg*

UNTERSUCHUNG GEEIGNETER MITTEL

4.1 LAUFFÄHIGKEIT HISTORISCHEN CODES

4.1.1 *Codeverwaltung*

4.1.2 *Lokale Entwicklungsumgebung*

4.1.3 *Continuous 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

- [Mar12] Robert C. Martin. *Clean code: a handbook of agile software craftsmanship* /. [Repr.] Robert C. Martin series. Upper Saddle River, NJ: : Prentice Hall, 2012. xxix+431. ISBN: 978-0-13-235088-4.
- [Mor14a] Levi Morrison. *PHP: rfc:remove_php4_constructors*. 17. Nov. 2014. URL: https://wiki.php.net/rfc/remove_php4_constructors (besucht am 30.09.2019).
- [Mor14b] Levi Morrison. *PHP: rfc:return_types*. 20. März 2014. URL: https://wiki.php.net/rfc/return_types (besucht am 02.10.2019).
- [Oraa] Oracle. *Anonymous Classes (The Java™ Tutorials > Learning the Java Language > Classes and Objects)*. Anonymous Classes (The Java™ Tutorials). URL: <https://docs.oracle.com/javase/tutorial/java/java00/anonymousclasses.html> (besucht am 02.10.2019).
- [Orab] Oracle. *MySQL :: MySQL 8.0 Reference Manual :: 13.5 Prepared SQL Statement Syntax*. URL: <https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-prepared-statements.html> (besucht am 02.10.2019).
- [Orao4] 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).
- [nik14] nikic. *Remove string category support in setlocale()*. GitHub. 10. Sep. 2014. URL: <https://github.com/php/php-src/commit/4c115b6b71e31a289d84f72f8664943497b9ee31#diff-b31234a9f5a03a328b60d0453988140f> (besucht am 01.10.2019).