

MARTIN DUSCHEK, 67664, 16MI1-B

EVOLUTION VON CODE BEI MAJOR-RELEASES VON
PROGRAMMIERSPRACHEN

EVOLUTION VON CODE BEI MAJOR-RELEASES
VON PROGRAMMIERSPRACHEN

MARTIN DUSCHEK, 67664, 16MI1-B

HITWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

am Beispiel von PHP

Juli 2019

Martin Duschek, 67664, 16MI1-B: *Evolution von Code bei Major-Releases von Programmiersprachen*, am Beispiel von PHP, © Juli 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

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
4	UNTERSUCHUNG GEEIGNETER MITTEL	7
4.1	Lauffähigkeit historischen Codes	7
4.1.1	Codeverwaltung	7
4.1.2	Lokale Entwicklungsumgebung	7
4.1.3	Continuous Integration mittels Containern	7
4.2	Erkennung des zu ändernden Codes	7
4.2.1	Manuelle Erkennung	7
4.2.2	Automatisierte Erkennung	7
4.3	Refactoring	7
4.3.1	Unit-Tests	7
4.3.2	Search & Replace	7
4.3.3	Wrapping	7

LISTINGS

Listing 3.1	Beispiel eines impliziten Konstruktors	5
Listing 3.2	Beispiel eines expliziten Konstruktors	6

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 (kurz für "PHP: Hypertext Preprocessor") 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.

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 Lauffähigkeit des Programms jedoch nicht beeinflusst.

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.

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

3.3 GEÄNDERTE FUNKTIONEN

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*