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

[15. Oktober 2019 at 8:08 – classicthesis v4.6]

INHALTSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

ABKÜRZUNGSVERZEICHNIS

PCRE	Perl Compatible Regular Expressions
API	Application Programming Interface
PHP	PHP: Hypertext Preprocessor
ISO	International Organization for Standardization („Internationale Organisation für Normung“)
IEC	International Electrotechnical Commission („Internationale Elektrotechnische Kommission“)
php7mar	PHP 7 Migration Assistant Report

1 EINLEITUNG

Am 03. Dezember 2015 erschien mit PHP 7.0.0 das erste Major-Release der Programmiersprache 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

Migration des Onlineshops der Firma *Tickets75*...

1.2 AUFGABENSTELLUNG

Ziel dieser Arbeit ist die Evaluation 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

Die International Organization for Standardization („Internationale Organisation für Normung“) (ISO) ist ein im Jahr 1947 gegründeter Zusammenschluss internationaler Normungskommissionen, mit dem Ziel internationale Standards zu entwickeln und zu etablieren.[Intb] Die Entwicklung von Standards wird von der 1906 gegründeten Schwesterorganisation International Electrotechnical Commission („Internationale Elektrotechnische Kommission“) (IEC) übernommen, oftmals in Zusammenarbeit mit der ISO.[Inta] Aus dieser Zusammenarbeit entstandene Standards tragen die Kürzel beider Organisationen im Namen. Ein solcher Standard ist **ISO/IEC 14764** mit dem Titel **Software Engineering — Software Life Cycle Processes — Maintenance**, der erstmals im Jahr 1999 veröffentlicht wurde. **ISO/IEC 14764** normiert 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

Was ist ein Major-Release, welche Versionierungsarten sind möglich, welche wird von PHP eingesetzt?

3 ÄNDERUNGEN DER PHP-API

Dieser Abschnitt stellt eine Auswahl der Bedingungen vor, welche die PHP-API in Version 7 gegenüber Version 5 an lauffähige Software stellt und welche neuen Mittel Entwicklern zur Verfügung gestellt werden. Die Änderungen werden in den Kontext der Weiterentwicklung der Programmiersprache gestellt, um Aussagen über die Gründe dieser zu treffen und - **ISO/IEC 14764** entsprechend - die Anforderungen an die Migration festzustellen.

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 Ausführungsumgebung zwingend vorrausgesetzt.

3.1.1 *Interpretation indirekter Variablenzugriffe*

PHP bietet die Möglichkeit des indirekten Zugriffs auf Variablen. Das bedeutet, dass der Wert einer Variablen den Namen einer weiteren Variablen darstellt. Bisher war die Syntax durch mehrere Sonderfälle geregelt. Mit PHP 7 wird eine strikte Evaluierung eines solchen Audrucks von links nach rechts eingeführt, um die Nutzung dieser zu vereinheitlichen. Wie sich die einzelnen Fälle Unterscheiden ist in Tabelle 3.1 aufgeführt.

3.1.2 *Switch-Anweisungen mit mehreren default-Blöcken*

Switch-Anweisungen, welche mehrere default-Blöcke enthalten werden ab sofort als fehlerhafte Syntax erkannt und werfen einen Fehler.

Tabelle 3.1: Vergleich der Evaluation indirekter Variablen zwischen PHP 5 und PHP 7

Ausdruck	**PHP 5**	**PHP 7**
$$foo['bar']['baz']	${$foo['bar']['baz']}	($$foo)['bar']['baz']
$foo->$bar['baz']	$foo->{$bar['baz']}	($foo->$bar)['baz']
$foo->$bar['baz']()	$foo->{$bar['baz']}()	($foo->$bar)['baz']()
Foo::$bar['baz']()	Foo::{$bar['baz']}()	(Foo::$bar)['baz']()

Dies war bisher nicht der Fall, allerdings wurde bei einer solchen Anweisung nur der letzte default-Block ausgewertet. Dieses Verhalten zeigt sich in Listing 3.1. Der entsprechende Codeausschnitt gibt unter PHP 5 immer Ëvaluatedäus, bei dem Versuch der Aufführung unter PHP 7 wird ein Fehler geworfen. Damit wird ein Bruch der PHP-Spezifikation [PHP] behoben.

Listing 3.1: Beispiel meherer default-Blöcke in Switch-Anweisungen

```
<?php
switch (1) {
    default:
        echo("Never evaluated");
        break;
    default:
        echo("Evaluated")
        break;
}
?>
```

3.1.3 *Verkehrte Reihenfolge der Variablenzuweisung mit list*

Die Funktion *list()* ermöglicht die Zuweisung von Variablen als wären diese ein Array. Quellcode der sich auf die bisherige Praxis verlässt, dass *list()* den letzten angegebenen Wert zuerst zuweist, kann nun nicht mehr eingesetzt werden, da die Reihenfolge der Zuweisung umgekehrt wurde. Obwohl keine klaren Gründe auszumachen sind, liegt die Vermutung nahe, dass die Änderung Verwirrungen über das Verhalten der Funktion vermindern soll. Listing 3.2 würde bei der Ausführung unter PHP 5 beispielsweise „3“ als Ergebnis ausgeben. Dies entspricht nicht der erwartbaren Reihenfolge.

Listing 3.2: Beispiel der Verwendung von list()

```
<?php
list($first, $second, $third) = [1,2,3];

echo($first);
?>
```

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 3.3 dargestellt. PHP 7 unterstützt diese Notation zwar noch, allerdings wird die, in PHP 5 eingeführte, explizite Benennung mit dem Schlüsselwort *__construct* (siehe Listing 3.4) bevorzugt. Hierdurch soll die Verwirrung darum, wann eine Funktion einen Konstruktor darstellt aufgehoben werden. [Mor14a]

Listing 3.3: Beispiel eines impliziten Konstruktors

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

Listing 3.4: 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 3.5 wird

eine Warnung ausgeben, dass eine nicht-statische Methode statisch aufgerufen wird.

Listing 3.5: 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, mit dem die Zeichenkette durch das Ergebnis einer Funktion ersetzt wird. Ein Beipiel ist die Umwandlung aller kleingeschriebenen Zeichen eines Strings in Großbuchstaben, dargestellt in Listing 3.6. 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, ausfühbaren Code in diese Funktion einzuschleusen.

Listing 3.6: 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, beispielsweise Java [Oraa], bekannt ist. Diese können benutzt werden, um gleichzeitig mit der Definition eine einmalig genutzte Klasse zu instanziieren, ohne eigens dafür eine neue lokale Klasse erstellen zu müssen., wie in Listing 3.7 dargestellt wird.

Listing 3.7: 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 Ersetzungsfuntion. Im eingeführten *preg_replace_callback_array()* kann nun ein assiozatives 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 Ersetzungsfuntion 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, wie in Listing 3.8. Dadurch sollen unter anderem ungewollte Rückgabewerte verhindert werden, als auch die automatisierte Dokumentation von Funktionen vereinfacht werden. [Mor14b]

Listing 3.8: Typdeklaration für Rückgabewerte

```
<?php
public function foo(): int {
    return 42;
}
?>
```

3.5 ENTFERNTE ERWEITERUNGEN

Einige Funktionalitäten von PHP sind nicht in die Sprache selbst eingebaut, sondern werden durch externe Erweiterungen eingebunden, die jedoch standardmäßig mit PHP ausgeliefert werden. Diese stehen somit nicht unter der Verwaltung der PHP-Entwickler und werden unabhängig weiterentwickelt.

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

3.5.2 *ereg*

Die Erweiterung *ereg* bietet verschiedene Funktionen für die Nutzung von **POSIX**-kompatiblen **Regulären Ausdrücken**. Die Erweiterung wurde zugungsten von *PCRE* entfernt, da diese unter anderem bessere Unterstützung von Unicode-Zeichen bietet und aktiv weiterentwickelt wird. [Pop14]

3.6 FAZIT

Die Ziele der Weiterentwicklung von PHP lassen sich folgendermaßen zusammenfassen:

- Erhöhung der Sicherheit
- Bessere Verständlichkeit des geschriebenen Quellcodes

Die Erhöhung der Sicherheit wird hauptsächlich durch die Entfernung von Erweiterungen erreicht, die nicht weiterentwickelt werden. Entwickler werden dadurch gezwungen, diese mit Mitteln zu ersetzen, welche zum Einen auch zukünftig mit Sicherheitsupdates versorgt werden und zum Anderen Features bieten um den Quellcode zusätzlich zu härten. Auch die Erweiterung von Konzepten der Typsicherheit und die Ersetzung von als Sicherheitsrisiko geltenden Funktionen dient diesem Zweck.
Dem Ziel der besseren Verständlichkeit von Quellcode dienen beispielsweise die Vereinheitlichung von Variableninterpretation, die Abschaffung der doppelten Konzepte der Deklaration von Konstruktoren und die Anpassung der *list()*-Funktion an gewohnte Denkweisen.

4 UNTERSUCHUNG GEEIGNETER MITTEL

Wie in Kapitel 3 gezeigt, sind die Veränderungen zwischen PHP 5 und PHP 7 nicht nur sehr umfangreich, sondern erfordern auch große Eingriffe in den betroffenen Quellcode. **ISO/IEC 14764** sieht vor, die Migration zu definieren

4.1 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 sind. Dafür relevante Beispiele sind in Kapitel 3 gelistet, die gesamte Liste kann der Dokumentation entnommen werden. Die Erkennung kann je nach Umfang des Quellcodes und der verwendeten Funktionen entweder manuell oder automatisiert durchgeführt werden. Für beide Arten werden im folgenden Beispiele genannt und die jeweiligen Vor- und Nachteile diskutiert.

4.1.1 *Manuelle Erkennung*

Eine manuelle Erkennung des Codes bietet sich vor allem bei kleinen Softwareprojekten an, bei denen ein vollumfänglicher Überblick über den eingesetzten Code besteht. Hier kann durch die in typischen Editoren und Entwicklungsumgebungen integrierte Suche genutzt werden um alle Vorkommen von nicht lauffähigen Funktionen zu finden und diese anschließend einem Refactoring zu unterziehen. Besonders einfach gestaltet sich diese Methode bei entfernten Funktionen, beispielsweise die der Erweiterung *ereg*3.5.2. Diese kann der Entwickler in der Dokumentation nachschlagen und den Code auf etwaige Vorkommen prüfen. Schwierig wird die manuelle Erkennung bei Änderungen wie der Einhaltung des Standards in Switch-Anweisungen3.1.2. Hier ist eine Suche nur über umfangreiche Suchmuster (Reguläre Ausdrücke) möglich, die meist nicht trivial zu erstellen sind und viele Einzelfälle (z.B. verschachtelte Switch-Anweisungen) abdecken müssen. In diesen Fällen ist durch die manuelle Suche höchstens eine Eingrenzung des Problems möglich.

4.1.2 *Automatisierte Erkennung*

Da die zuvor besprochene manuelle Erkennung betroffenen Codes nur für einzelne Fälle oder kleine Projekte in Frage kommt, bietet sich als alternative die automatiche Erkennung an, mit dem Ziel, dem Entwick-

ler einen vollumfänglichen Überblick der zu überarbeitenden Stellen im Code zu liefern. Im vorliegenden Fall wurde das Tool *php7mar*[1] des Entwicklers *Alexia* genutzt. *php7mar* erkennt mithilfe von **Regulären Audrücken**, **String-Matching** und **Lexikalischer Analyse** kritischen Code in Projekten und generiert daraus einen Bericht, bestehend aus Zeilenangaben, gefundenen Problemen und Lösungshinweisen. Ein Beispiel eines solchen Berichts findet sich in Listing4.1. In der Datei *GMCSS.php* werden drei Fehlerklassen gefunden: Erstens mehrere Fälle der Nutzung der veralteten Definition von Konstruktoren, zweitens einige Vorkommen der Entfernten Erweiterung *mysql*, sowie drittens ein indirekter Variablenzugriff, dessen Aussage unter PHP 7 möglicherweise eine andere ist (vgl. Kapitel 3.1.1). Insbesondere der erste Fall zeigt die Überlegenheit eines Analysetools, da solche Fehler nur schwer mit einer trivialen Suche zu finden sind.

Listing 4.1: Beispiel eines generierten Berichts mit *php7mar*

```
#### C:\Users\Nutzer\Documents\GitHub\
    gambio_tickets75\StyleEdit\classes\GMCSS.php
* oldClassConstructors
* Line 55:  'function GMCSS($p_css_file, $p_type='
    archive')'
* Line 384: 'function GMCSSImport($p_css_file = false
    , $p_import_mode = '')'
* Line 791: 'function GMCSSExport($p_css_file)'
* Line 912: 'function GMCSSUpload($p_files, $p_type)'
* Line 982: 'function GMCSSArchive()'
* deprecatedFunctions
* Line 302: '$t_css_query = mysql_query("'
* Line 311: 'if((int)mysql_num_rows($t_css_query) >
    0)'
* Line 313: '$t_row_styles = mysql_fetch_array(
    $t_css_query, MYSQL_ASSOC);'
* Line 316: '$t_css_query = mysql_query("'
* Line 325: 'if((int)mysql_num_rows($t_css_query) >
    0)'
* variableInterpolation
* Line 359: 'global $$shippingModule; //notice $$'
```

4.2 REFACTORING

Refactoring beschreibt die Technik, bestehenden Code in seiner Struktur so zu verändern, dass dieser änderbar bleibt und wichtige Bestandteile leicht indentifizierbar bleiben, ohne jedoch den eigentlichen Sinn des Programms zu verändern. [Fow99] Bei der Migration von Software muss dem Refactoring ein hoher Stellenwert zugemessen werden, da große Bestandteile des Codes ausgetauscht und verändert werden, was nach Lehman [Leh80, S. 1060-1076] zur Degeneration

1 Alexia. php7mar. URL: https://github.com/Alexia/php7mar

von Software führt. Die Schwierigkeit des Refactorings liegt häufig im zeitaufwändigen Verstehen des vorliegenden Codes. Diese Zeit muss jedoch für die Migration der Software ohnehin aufgewendet werden, womit sich ein Refactoring hier durchaus anbietet. Die Ziele des Refatorings können durch verschiedenste Techniken erreicht werden, beispielsweise die Zerlegung von Funktionen in kleinere, bessere verständliche Funktionen oder die Auslagerung von immer wieder genutztem Code in eine eigene Funktion.

4.2.1 *Extrahieren*

Bei der Extraktion von Funktionen sollen häufig genutzte Programmteile in eine eigene, wiederverwendbare Funktion ausgelagert werden. Dazu wird die neue Funktion angelegt,

4.2.2 *Zerlegung von Funktionen*

4.2.3 *Unit-Tests*

4.3 LAUFFÄHIGKEIT HISTORISCHEN CODES

Der Standard ISO/IEC 14764 sieht auch nach der erfolgten Migration eine Unterstützung der alten Umgebung vor. Dies ist natürlich vor allem bei Produkten sinnvoll, die von Dritten eingesetzt werden, die dadurch vor nicht erwartetem Verhalten der Software geschützt werden. Allerdings ist eine Unterstützung der alten Umgebung auch bei unternehmenseigener Software sinnvoll. Beispielsweise lassen sich Fehler im Programm zurückverfolgen, Änderungen nachvollziehen und sichergestellt werden, dass im Falle eines Fehlers bei der Migration eine lauffähige Version zur Verfügung steht. Daraus ergeben sich jedoch einige Probleme. So muss nicht nur der Quellcode von alten Versionen eines Programms zur Verfügung stehen, sondern auch die verschiedenen Umgebungen um die Software ausführen zu können.

4.3.1 *Versionsverwaltung*

Grundlage für die Unterstützung alter Versionen einer Software ist, dass diese Versionen in ihrem ausgelieferten Zustand vorhanden sind und Änderungen, die seitdem vorgenommen wurden protokolliert und nachvollziehbar sind. Ein einfacher Ansatz dazu wäre beispielsweise die Ablage des Programmcodes in einem eindeutig benannten Ordner, nachdem eine Änderung durchgeführt wurde, sowie der Einsatz eines Programms wie beispielsweise **Diff** [2] zur Kenntlichmachung von Änderungen zwischen zwei Dateien. Diese Vorgehensweise kann

2 GNU Diffutils, URL: http://www.gnu.org/software/diffutils/

zwar für kleinere Projekte genügen, ist jedoch für große Projekte mit mehreren hundert einzelnen Dateien und beliebig vielen bearbeitenden Personen nicht geeignet. Dieses Problem kann durch den Einsatz einer dedizierten Software zur Versionsverwaltung gelöst werden. Diese Software beinhaltet den gesamten Quellcode des Projekts in einem **Repository**, stellt diesen den Nutzern bereit und protokolliert jede Änderung, sogenannte **Commits**. Durch diese Protokollierung kann der Quellcode in jeden beliebigen früheren Zustand zurückversetzt werden, und somit auch weiter gewartet werden. Bekannte Programme zur Versionsverwaltung mit den genannten Funktionen sind zum Beispiel *Git*[3] oder *Mercurial*[4]

4.3.2 *Ausführungsumgebung*

Zu jeder historischen Version der Software muss die passende Ausführungsumgebung zur Verfügung stehen. Dies betrifft nicht nur PHP, sondern beispielsweise auch Python mit der immer noch andauernden flächendeckenden Umstellung von Version 2 auf 3.

4.3.2.1 *Lokale Ausführungsumgebung*

4.3.2.2 *Continuous Integration mittels Containern*

3 Git, https://git-scm.com/
4 Mercurial, URL: https://www.mercurial-scm.org/

5 MIGRATION DES TICKETS75 ONLINE-SHOPS

Ablauf der praktischen Migration nach ISO/IEC 14764

6 SCHLUSSBETRACHTUNGEN

Fazit/Ausblick

[Fow99] Martin Fowler. „Refactoring - Improving the Design of Existing Code“. In: (1999). Unter Mitarb. von Kent Beck und John Brant, S. 337.

[Inta] International Electrotechnical Commission. *IEC - About the IEC*. About the IEC. URL: https://www.iec.ch/about/?ref=menu (besucht am 05. 10. 2019).

[Intb] International Organization for Standardization: *About ISO*. About ISO. URL: http://www.iso.org/cms/render/live/en/sites/isoorg/home/about-us.html (besucht am 05. 10. 2019).

[Leh80] Meir M. Lehman. *Programs, Life Cycles, and Laws of Software Evolution*. 1980.

[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/javaOO/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).

[Ora04] 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).

[PHP] PHP Group. *PHP Language Specification*. GitHub. Unter Mitarb. von nikic, smalyshev, zhujinxuan und mousetraps. URL: https://github.com/php/php-langspec/blob/master/spec/11-statements.md#the-switch-statement (besucht am 04. 10. 2019).

[Pop14] Nikita Popov. *PHP: rfc:remove_deprecated_functionality_in_php7*. 11. Sep. 2014. URL: https://wiki.php.net/rfc/remove_deprecated_functionality_in_php7 (besucht am 03. 10. 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).