**PHP Parallel Lint**

Der Linter kann, ohne vorher eingerichtet zu werden, als phar-Datei ausgeführt werden. Der Versuch wurde mit „php parallel-lint.phar TenSamples.php“ gestartet und konnte keines der zehn Fehler finden. Das Tool kommt ohne weitere Konfigurationsmöglichkeiten, weshalb davon auszugehen ist, dass es nicht möglich ist diese Art von Fehlern mit diesem Linter identifizieren zu können. Weitere Tests haben ergeben, dass der Linter nur auf sehr grobe Fehler anschlägt, wie das Weglassen des Variable Namens nach einem Dollarzeichen oder das Vergessen eines Semikolons. Zwar schneidet der PHP Parallel Lint in Geschwindigkeit mit am besten ab, so sind die Fehler, die es erkennen kann, doch zu unzureichend und in der Praxis selten zu gebrauchen, da die meisten, wenn nicht sogar alle, dieser Fehler durch den PHP-Interpreten bereits automatisch entdeckt werden und das Ausführen der PHP Skripte verhindert. Der Linter wäre nur dann zu gebrauchen, wenn man ohne das Skript auszuführen überprüfen möchte, ob das Skript in der Theorie ausführbar wäre. Besonders praktisch wäre dies, wenn im Projekt Skripte vorhanden sind, die nur bei Bedarf geladen werden oder nur in einem bestimmten Kontext funktionieren, da diese meistens nicht automatisch überprüft werden und grundsätzlich auch schwer zu testen sind, so könnte man einen Studenten bei der Abgabe warnen, dass manche Dateien wahrscheinlich nicht ausführbar wären. Über die Nützlichkeit einer solchen Warnung lässt sich diskutieren, da man schon davon ausgehen kann, dass ein Student bei der Abgabe funktionierenden Code abgibt.

**PHP Code Sniffer**

Der Linter kann, ohne vorher eingerichtet zu werden, als phar-Datei ausgeführt werden. Der Versuch wurde mit „php phpcs.phar TenSamples.php“ gestartet und konnte keines der zehn Fehler finden, dafür jedoch 30 Verstoße gegen etablierte Coding Standards. Diese Coding Standards tragen leider nicht zur Sicherheit des Codes bei, sorgen jedoch für einheitlichen Code, bessere Instandhaltungsfähigkeiten und bessere Kompatibilität. So verlangt das Skript Dokumentationen über jede definierte Funktion und Klasse, um eine nachhaltige Entwicklung zu fordern. Es konnte auch fehlerhafte Zeilenumbruchzeichen ausfindig machen und könnte diese bei Bedarf auch von selbst beheben, so sollte das Skript dann auch problemlos auf unix-Systemen laufen können. Außerdem kann es feststellen, ob man sich an die Groß-/ Kleinschreibung der Namen eingehalten hat, was dafür sorgt, dass man Eigenschaften einer Variablen oder Funktion bereits an ihrem Namen feststellen kann, zum Beispiel das private Variablen mit einem Unterstrich anfangen sollen oder dass die Wörter eines Funktionsnamens immer groß und zusammengeschrieben wird. Der Linter warnt auch, wenn eine Zeile zu viele Zeichen beinhaltet, um die Lesbarkeit des Codes zu schützen. Insgesamt ist der Code Sniffer kein schlechter Linter, denn dieser schafft es die Qualität des Codes mit einfachen Ratschlägen zu verbessern. Für die allgemeine Nutzung auf einer Vielzahl von unterschiedlichen Studenten und deren Code ist der Linter jedoch nicht so gut geeignet. Diesen Linter sollte man, wenn überhaupt, während der aktiven Entwicklung an einem Projekt nutzen, damit man kontinuierlich den Code verbessern und nachhaltig entwickeln kann. Am Ende eines Projekts, während der Abgabe, sind die von dem Linter gefundene Probleme meist nicht mehr zu lösen, da die bloße Menge an angesammelten Problemen, die Zahlen im vierstelligen Bereich erreichen kann - zu groß wäre, als dass sie ein Student dann noch beheben könnte.

**PHPMD**

Das Tool lässt sich ganz einfach via composer oder phar installieren. Es stehen sechs verschieden Regelsets zur Verfügung. Clean Code Rules, Code Size Rules, Controversial Rules, Design Rules, Naming Rules, Unused Code Rules. Jedes Regelset beinhaltet mehrere untergeordnete Regeln, die man in eine xml Datei als Anforderung stellen kann. Es ist nicht zwingend eine Regeldatei zu erstellen, den Namen der einzelnen Regel im Konsolenbefehl zu listen ist auch ausreichend. Lässt man die Regeln komplett aus verwendet PHPMD ein Standard Regelset.  
So kann ein Konsolenbefehl aufgebaut sein:

|  |
| --- |
| phpmd src html ruleset.xml --reportfile result/PHPMD\_Result.php |

Eine Quelldatei oder Quellordner, Ein Ausgabeformat, Eine Regel Datei und die Datei wo die Ausgabe gespeichert wird. In diesem Fall geben wir das Format html in einer php Datei aus und das ist ein großer Vorteil von PHPMD, das benutzerfreundliche Frontend. Im Browser sieht es dann sehr übersichtlich aus und die Seite ist auch interaktiv. Es werden Statistiken, Zählungen und die einzelnen verstoße aufgezählt. In diesem Versuch wurden 32 Verstosse vom Linter festgestellt.

|  |  |  |
| --- | --- | --- |
| Count | % | Rule set |
| 19 | 59.4 % | Controversial Rules |
| 5 | 15.6 % | Clean Code Rules |
| 4 | 12.5 % | Design Rules |
| 2 | 6.2 % | Unused Code Rules |
| 1 | 3.1 % | Code Size Rules |
| 1 | 3.1 % | Naming Rules |

**PHPLINT**

Ebenso hier kann man diesen Linter mit composer schnell eingerichtet bekommen und mit dem Linten loslegen. Auch mit Docker ist er kompatibel. Bei diesem Tool hat es sich herausgestellt, dass es eher die Standardaufforderung einer richtigen PHP Syntax von mehreren Ordnern mit PHP Dateien oder mehrere einzelne PHP Dateien gleichzeitig überprüft. So kann ein Konsolenbefehl aufgebaut sein.

|  |
| --- |
| phplint src --extensions=php --xml=result/PHPLint\_Result.xml --warning --verbose |

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungHier werden mit der extensions Option nur PHP Dateien überprüft, mit warning werden zusätzlich PHP Compiler Warnungen ausgegben. --verbose oder -v|vv|vvv, erhöht die Ausführlichkeit von Meldungen: 1 für normale Ausgabe, 2 für ausführlichere Ausgabe und 3 für Debug. Das Lint Ergebnis wird als xml Format in der vorgegebenen Datei gespeichert. Bei diesem Versuch gab es 0 Fehler da die Syntax gestimmt hat. Baut man ein Fehler ein wie z.b ein Semikolon zu vergessen. Sieht das Ergebnis wie folgt aus:

**PHP Coding Standards Fixer**

Der Versuch wurde mit „php composer.phar phpcs“ gestartet und konnte bei jedem Abschnitt zumindest einen Fehler finden. Das Tool verfügt über zahlreiche Konfigurationsmöglichkeiten, daher wurde eine zusätzliche Konfigurationsdatei benötigt, um die gewünschten Rules zu bestimmen. Für diesen Versuch wurde hauptsächlich Symfony als Rules verwendet. Außerdem wurden die non-Linux-Whitespaces in der Konfigurationsdatei konfiguriert.

Die ausgeführten Tests haben ergeben, dass der Linter auf mehrere Arten von Fehlern stoßt, wie z.B. das increment style, überflüssige Kontrollklammern, no-trailing-whitespace und concat-space.

Was meiner Meinung nach als großer Nachteil angesehen werden könnte, ist, dass der Linter im Falle eines Syntaxfehlers, wie z.B. das Vergessen eines Semikolons, den Fehler nicht korrigiert und nur ausgibt, dass ein Syntaxfehler beim Linting gefunden wurde, ohne nähere Angaben zu machen, was dieser Fehler eigentlich ist, was in einem großen Projekt problematisch sein könnte.

Das Besondere an PHP-CS-Fixer ist aber, dass der Linter die Fehler nicht nur erkennt, sondern sie auch für den Benutzer behebt. Dies ist besonders praktisch für große Projekte, bei denen es mühsam ist, eine große Anzahl von Fehlern von Hand zu beheben.