Code Quality Tools

Fatema Aly
FH Technikum Wien
Faculty of Computer Scince
Vienna, Austria
if20b057@technikum-wien.at

Jan Loos
FH Technikum Wien
Faculty of Computer Scince
Vienna, Austria
if20b118@technikum-wien.at

Taha Samaha
FH Technikum Wien
Faculty of Computer Scince
Vienna, Austria
if20b150@technikum-wien.at

***Abstract*— In software development area, writing a high-quality code is of immense importance, as it can save time and resources in the long run by making it easier to understand, modify, and maintain the codebase. It can also improve the overall performance and scalability of the software. Linters, code style or best practice approaches should be considered to ensure writing such a high-quality code. Furthermore, linters can be integrated into development environments and build systems, making it easy to run them on a regular basis. This can help to automate the process of checking for errors and enforcing coding conventions.**

In this paper, we will demonstrate how a CI/CD pipeline with respect to PHP using Docker is created. The main goal of this project is to automatically test the submitted files of students at the university of applied sciences for technology and digitalization (FHTW) according to predefined criteria using an isolated Docker container in the background. This process should establish a continuous improvement of code quality and ensure the maintainability of the codebase of FHTW Students.

***Keywords*— PHP, linter, syntax, errors, code, Docker**

I. Introduction

In the area of Continuous Integration or Continuous Delivery, there are many parameters for improving code quality. This can be checked with unit tests as well as code style or code patterns or best practice approaches. The goal of this project is to create a CI/CD pipeline with respect to PHP, which enables students to check the code on their own machine for safety & security, including other aspects. This process is intended to test the submitted files automatically in the background according to predefined criteria and should ultimately lead to an improved quality and maintainability of a codebase.

To achieve this goal, the generated feedback should be of high quality, clear and well-structured. Severe errors should not be lost in a flood of subsequent errors. furthermore, the generated feedback should remain objective and be based on facts and not impose subjective changes on the user.

It is worth mentioning that the software is not intended to fix errors by itself, but only to make the user aware of his own errors.

For the implementation of this tool, it is considered to integrate already existing open-source software in Moodle plugin EspressoIT. For this, the functionality of the software and of EspressoIT must be analyzed in detail and adapted if necessary.

At the end of the project, the software should be able to be started from a Docker container in the background. The software should evaluate the submitted PHP files and produce results in JSON, if possible, otherwise in either XML or HTML.

II. Search For Suitable Linters

At the beginning of the project, research was conducted to find a suitable open-source linter, that can serve the purpose of this project. During the searching phase, several PHP linters were reviewed and tested to adopt the most convenient linter for the required Moodle extension.

Besides the ability of the linter to detect errors in source-code, we also focused on the quality of the delivered messages, as they should be as understandable and obvious as possible. The format of the results had played a huge role as well in this phase, as it was strongly preferred, that the software deliver the results in JSON.

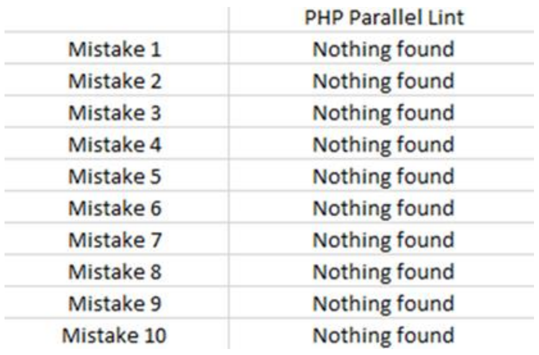
In this context, five linters were selected and tested on a PHP Code with the most 10 common Mistakes PHP developers make [1].

A. PHP Parallel Lint

PHP Parallel Lint is a linter developed specifically for PHP that uses parallel processing to check the syntax of multiple PHP files simultaneously, speeding up the linting process. It works by running multiple instances of the built-in PHP linter in parallel, each checking a different file. It can be run from the command line and integrates with various development tools and build systems. The output of PHP Parallel Lint can be in plain text, colored text, JSON, and check syntax formats [2].

Running parallel jobs in PHP is inspired by Nette framework tests. The application is officially supported for use with PHP 5.3 to 8.1 [2].

During the test, it was clear that this tool is only capable of detecting severe syntax errors. This is only useful for detecting files that are not executable. Therefore, this linter would not serve the purpose of the project, as it is expected from students to have already run their code at least once before submitting it for grading.



	PHP Parallel Lint
Mistake 1	Nothing found
Mistake 2	Nothing found
Mistake 3	Nothing found
Mistake 4	Nothing found
Mistake 5	Nothing found
Mistake 6	Nothing found
Mistake 7	Nothing found
Mistake 8	Nothing found
Mistake 9	Nothing found
Mistake 10	Nothing found

Fig. 1. Results of PHP Parallel Lint

B. *PHP Lint*

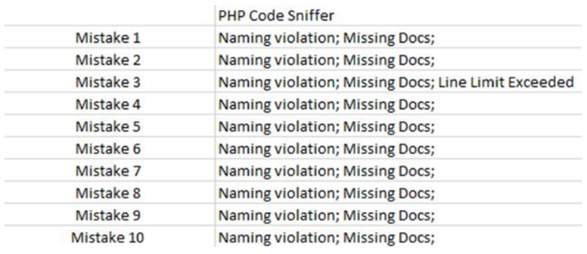
PHP Lint is the built-in syntax checker for PHP, which is included with the PHP interpreter. It is used to check the syntax of PHP code, looking for any errors or inconsistencies that may prevent the code from running correctly. It can be run from the command line by invoking the PHP interpreter and passing the file to be checked as an argument. PHP Lint will check the syntax of the code and will return a message indicating whether the code is error-free or if there are any syntax errors. This tool checks only for syntax errors and doesn't check for any other type of coding issues. The installation of PHP Lint requires PHP 8.0 or higher and Composer 2.0 or higher. This linter can format the results into a JSON or JUnit XML file and can also display warnings in the output [3].

The conducted review showed that PHP Lint is also only able to indicate severe syntax errors, as PHP Parallel Lint, which would not be of a great advantage for this project.

C. *PHP_CodeSniffer*

PHP_CodeSniffer is a tool that can be used to check PHP code for adherence to a specific coding standard. It can be used to detect violations of a predefined set of coding rules, such as naming conventions, indentation, and commenting practices. PHP Code Sniffer works by analyzing the token stream of a PHP script, rather than simply checking the syntax. This allows it to detect more subtle issues and enforce a wide range of coding standards. It can be integrated into various development environments, such as IDEs and text editors, and can be used as part of a continuous integration pipeline to automatically check code for compliance before it is committed. PHP_CodeSniffer requires PHP version 5.4.0 or greater, although individual sniffs may have additional requirements such as external applications and scripts [4].

The testing phase indicated that naming conventions should be followed to write more consistent code and to be able to recognize the properties of functions and variables directly from their names. Furthermore, documentation should be written for each function and class, to ease work with the code in the future. Code Sniffer also identifies long lines, which makes the code harder to read and incomprehensible. It is also capable of detecting and fixing incorrect line breaks, so that the script can also run on Unix systems without problems.



	PHP Code Sniffer
Mistake 1	Naming violation; Missing Docs;
Mistake 2	Naming violation; Missing Docs;
Mistake 3	Naming violation; Missing Docs; Line Limit Exceeded
Mistake 4	Naming violation; Missing Docs;
Mistake 5	Naming violation; Missing Docs;
Mistake 6	Naming violation; Missing Docs;
Mistake 7	Naming violation; Missing Docs;
Mistake 8	Naming violation; Missing Docs;
Mistake 9	Naming violation; Missing Docs;
Mistake 10	Naming violation; Missing Docs;

Fig. 2. Results of PHP_CodeSniffer

D. *PHP Mess Detector*

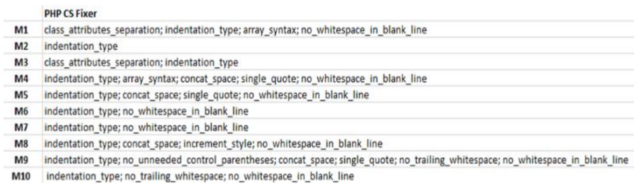
PHP Mess Detector (PHPMD) is a tool that can be used to analyze PHP code and detect potential problems such as messy, duplicated, or unused code. It can help to identify complex, hard-to-maintain, or error-prone areas of the codebase, and improve the overall quality and maintainability of the code. PMD uses a set of predefined rules to check for issues i.e., naming conventions, code complexity, unused variables, and code duplication. PHPMD can be run from the command line, integrated into various development environments, and used as part of a continuous integration pipeline to automatically check code for compliance before it is committed. PHPMD is additionally able to format the results into JSON reports, XML reports, single HTML file with possible problems, plain text and into the Static Analysis Results Interchange Format (sarif) [5].

The tests showed that PHPMD is also able to detect violations of naming conventions. In addition, PHPMD warned during the test that the class contains too many public functions and that the functions should be preferably divided into more classes, to deliver a better structured code. The linter also points out unused variables and the missing class imports. It also indicates errors that are relevant in a production environment, such as the use of undefined variables or the use of debug functions.

E. *PHP Coding Standard Fixer*

PHP Coding Standard Fixer (PHP CS Fixer) is a tool that can be used to automatically fix coding standard violations in PHP code. It can be used to enforce a specific coding standard, such as PSR-1, PSR-2, or the Symfony Coding Standard, and can automatically fix issues such as indentation, naming conventions, and whitespace. PHP CS Fixer can be run from the command line and can also be integrated into various development environments, such as IDEs and text editors. It supports a wide range of options and configurations, making it highly customizable and adaptable to different projects and teams [6].

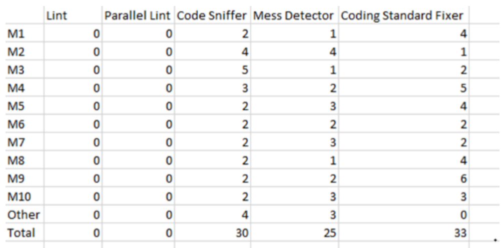
The testing phase indicated that PHP Coding Standard Fixer finds mainly formatting errors, i.e., wrong number of line breaks, wrong number of spaces, inconsistent initialization of arrays, inconsistent use of quotes or too many round brackets. Fixing such errors significantly contributes to enhance the readability of the code.



	PHP CS Fixer
M1	class_attributes_separation; indentation_type; array_syntax; no_whitespace_in_blank_line
M2	indentation_type
M3	class_attributes_separation; indentation_type
M4	indentation_type; array_syntax; concat_space; single_quote; no_whitespace_in_blank_line
M5	indentation_type; concat_space; single_quote; no_whitespace_in_blank_line
M6	indentation_type; no_whitespace_in_blank_line
M7	indentation_type; no_whitespace_in_blank_line
M8	indentation_type; concat_space; increment_style; no_whitespace_in_blank_line
M9	indentation_type; no_unneeded_control_parentheses; concat_space; single_quote; no_trailing_whitespace; no_whitespace_in_blank_line
M10	indentation_type; no_trailing_whitespace; no_whitespace_in_blank_line

Fig. 3. Results of PHP CS Fixer

As a next step, the number of the detected errors were summarized and analyzed. The following table and graph show that some linters can significantly find more errors in some functions than others. Overall, the number of errors found by PHP_CodeSniffer, Mess Detector and Coding Standard Fixer are very similar.



	Lint	Parallel Lint	Code Sniffer	Mess Detector	Coding Standard Fixer
M1	0	0	2	1	4
M2	0	0	4	4	1
M3	0	0	5	1	2
M4	0	0	3	2	5
M5	0	0	2	3	4
M6	0	0	2	2	2
M7	0	0	2	3	2
M8	0	0	2	1	4
M9	0	0	2	2	6
M10	0	0	2	3	3
Other	0	0	4	3	0
Total	0	0	30	25	33

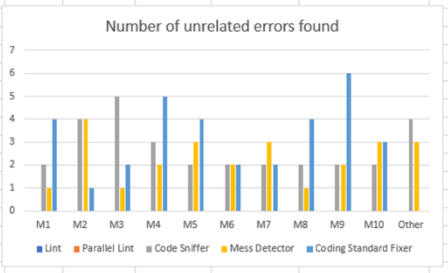


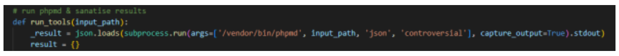
Fig. 4. Comparison of results of found linters.

III. Running Linters Simultaneously In Python

In the next step, two linters should be adopted from the already five selected linters and should be executed simultaneously. For this purpose, and based on the findings of the testing phase, PHP MD and PHP-CS Fixer were chosen. At this point, it was found, however, that PHP-CS Fixer is not able to generate an appropriate summary, so it had to be replaced with an alternative linter.

During the search for a new linter, PHP Static Analysis Tool (PHPStan) arose. PHPStan is a static analysis tool for PHP that can detect a wide range of errors and bugs in your codebase. It's a set of rules to check source code for various issues i.e., Undefined variables, type errors, unused variables and accessing array elements with incorrect keys. It works by analyzing the code without running it, so it can catch errors even before the code is executed. PHPStan can also suggest improvements to the code, such as adding missing type hints. And certainly, it can generate a convenient result's summary. Therefore, it was adopted as a replacement for PHP-CS-Fixer [7].

The two linters have been built into a Python script, which can run the two linters simultaneously using *subprocess* module, that allows spawning new processes, connect to their input/output/error pipes, and obtain their return codes. For this process, the function *subprocess.run()* is used to run a command in a sub-process and wait for it to complete. This function runs the command and returns a CompletedProcess instance, which contains the return code and output of the command [8].



```
# run phpmd & sanatise results
def run_tools(input_path):
    _result = json.loads(subprocess.run(args=['/vendor/bin/phpmd', input_path, 'json', 'controversial'], capture_output=True).stdout)
    result = {}
```

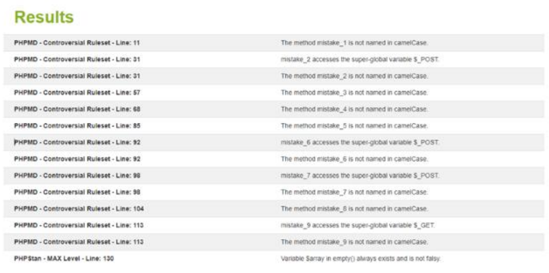
Fig. 5. Subprocess.run() function from python script

The *subprocess.run()* function in this case contains the following parameters:

- Args: the arguments used to launch the process. This may be a list or a string [8].
- Capture_output: this parameter allows capturing the standard output and standard error streams of the subprocess module into a variable. When set to True, the standard output and standard error streams will be captured into the *stdout* and *stderr* attributes of the CompletedProcess instance returned by the *subprocess.run()* function [8].
- Stdout: Captured stdout from the child process. A bytes sequence, or a string if *run()* was called with an encoding, errors, or text=True. None if stdout was not captured [8].

The python script removes any error duplication as well and combines the output into one HTML file. To generate the HTML file, the python library *dominate* was installed and imported. This library allows generating and structuring HTML content easily in Python using an elegant DOM API without having to manually write HTML code [9].

Next, the results of the unified linter were adjusted to the Moodle design and color scheme using CSS as the next table demonstrates. The left column shows the broken rule and the line number, and the right column shows the error message itself.



Results

PHPMD - Controversial Ruleset - Line: 11	The method mistake_1 is not named in camelCase.
PHPMD - Controversial Ruleset - Line: 31	mistake_2 accesses the super-global variable $_POST.
PHPMD - Controversial Ruleset - Line: 31	The method mistake_2 is not named in camelCase.
PHPMD - Controversial Ruleset - Line: 57	The method mistake_3 is not named in camelCase.
PHPMD - Controversial Ruleset - Line: 68	The method mistake_4 is not named in camelCase.
PHPMD - Controversial Ruleset - Line: 85	The method mistake_5 is not named in camelCase.
PHPMD - Controversial Ruleset - Line: 92	mistake_6 accesses the super-global variable $_POST.
PHPMD - Controversial Ruleset - Line: 92	The method mistake_6 is not named in camelCase.
PHPMD - Controversial Ruleset - Line: 98	mistake_7 accesses the super-global variable $_POST.
PHPMD - Controversial Ruleset - Line: 98	The method mistake_7 is not named in camelCase.
PHPMD - Controversial Ruleset - Line: 104	The method mistake_8 is not named in camelCase.
PHPMD - Controversial Ruleset - Line: 113	mistake_9 accesses the super-global variable $_GET.
PHPMD - Controversial Ruleset - Line: 113	The method mistake_9 is not named in camelCase.
PHPStan - MAX Level - Line: 130	Variable $array in empty() always exists and is not falsy.

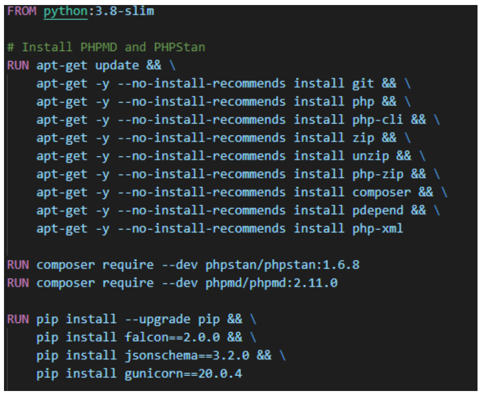
Fig. 6. Table of results of PHP MD and PHP-CS-Fixer

IV. Integration With EspressoIT

In this phase, the already completed Python script should be integrated into EspressoIT. Accordingly, the tool should run in its own Docker container, isolated from the underlying operating system, just like other modules of EspressoIT.

To assemble a Docker image and finally to allow the PHPML tool to run in a remote Docker container, a Dockerfile was implemented.

A Dockerfile is a script which contains all the essential commands, libraries, working directories and the necessary environment variables, that the user needs to call on the command line to automate the process of creating a docker image. Whereby the Docker image is the package itself, which is lightweight, executable and allows the user to run the software inside the docker container by using the command "docker run [OPTIONS] IMAGE [COMMAND] [ARG...]" [10].



```
FROM python:3.8-slim

# Install PHPMD and PHPStan
RUN apt-get update && \
    apt-get -y --no-install-recommends install git && \
    apt-get -y --no-install-recommends install php && \
    apt-get -y --no-install-recommends install php-cli && \
    apt-get -y --no-install-recommends install zip && \
    apt-get -y --no-install-recommends install unzip && \
    apt-get -y --no-install-recommends install php-zip && \
    apt-get -y --no-install-recommends install composer && \
    apt-get -y --no-install-recommends install pdepend && \
    apt-get -y --no-install-recommends install php-xml

RUN composer require --dev phpstan/phpstan:1.6.8
RUN composer require --dev phpmd/phpmd:2.11.0

RUN pip install --upgrade pip && \
    pip install falcon==2.0.0 && \
    pip install jsonschema==3.2.0 && \
    pip install gunicorn==20.0.4
```

Fig. 7. Screenshot from PHPML Dockerfile

The tool PHP Multi Linter (PHPML) runs in a Docker container and communicates with the business layer via the REST interface. The business layer periodically sends submitted source code to the tool and contains the results, which can be rendered consistently via EspressoIT.

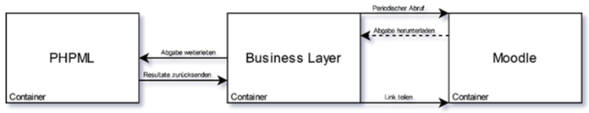


Fig. 8. Simple sequence diagram

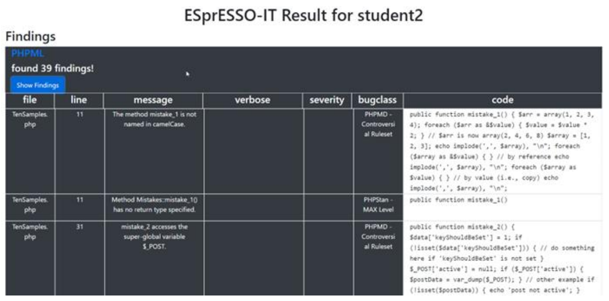
With the Docker container, there is now a single operating system and resources are shared between containers. Therefore, it is small and loads in a few seconds.

V. Results

With the help of the implemented PHPML tool, lecturers have the possibility to activate the tool for a submission, like the other tools, which are already implemented for other programming languages. Students will also be able to perceive suggestions for improvement in a timely manner after submitting their source code.

The output is described as detailed as possible, so that students can quickly identify the places, that should be improved in the source code.

The results of the PHPML tool are displayed in a table. The first column shows the file that contains the error, whereas the second one shows the line number in the source code. Next to the line column, the error message itself is displayed. The table shows also which Linter detected the error and which function contains the error. The table also shows which Linter detected the error and which function contains the error. The table also shows which Linter detects the error and which function contains it.



ESprESSO-IT Result for student2

Findings

PHPML

found 39 findings!

Show Findings

file	line	message	verbose	severity	bugclass	code
TenSamples.php	11	The method mistake_1 is not named in camelCase.			PHPMD - Controversial Ruleset	public function mistake_1() { $arr = array(1, 2, 3, 4); foreach ($arr as &$value) { $value = $value * 2; } // $arr is now array(2, 4, 6, 8) $array = [1, 2, 3]; echo implode(',', $array), "\n"; foreach ($array as &$value) { } // by reference echo implode(',', $array), "\n"; foreach ($array as $value) { } // by value (i.e., copy) echo implode(',', $array), "\n";
TenSamples.php	11	Method Mistakes::mistake_1() has no return type specified.			PHPStan - MAX Level	public function mistake_1()
TenSamples.php	31	mistake_2 accesses the super-global variable $_POST.			PHPMD - Controversial Ruleset	public function mistake_2() { $data['keyShouldBeSet'] = 1; if (!isset($data['keyShouldBeSet'])) { // do something here if 'keyShouldBeSet' is not set } $_POST['active'] = null; if ($_POST['active']) { $postData = var_dump($_POST); } // other example if (!isset($postData)) { echo 'post not active'; }

Fig. 9. Screenshot from delivered findings of PHPML.

It is worth mentioning, that the PHPML tool is also extensible, which means that additional linters can be included to generate more accurate results or add certain errors to the output. For this, only the respective code block must be duplicated and adapted for the output of the respective linter. Currently, the tool is not yet parallelized, but it should not be a major challenge to achieve this.

References

[1] "Buggy PHP Code: The 10 Most Common Mistakes PHP Developers Make | Toptal®," Toptal Engineering Blog. https://www.toptal.com/php/10-most-common-mistakes-php-programmers-make (accessed Jan. 26, 2023).

[2] "PHP Parallel Lint," GitHub, Jan. 22, 2023. https://github.com/php-parallel-lint/PHP-Parallel-Lint (accessed Jan. 27, 2023).

[3] 安 正 超 , " PHPLint, " GitHub, Jan. 20, 2023. https://github.com/overtrue/phplint (accessed Jan. 27, 2023).

[4] "squizlabs/PHP_CodeSniffer," GitHub, Jan. 28, 2023. https://github.com/squizlabs/PHP_CodeSniffer (accessed Jan. 27, 2023).

[5] "PHPMD," GitHub, Jan. 26, 2023. https://github.com/phpmd/phpmd (accessed Jan. 27, 2023).

[6] "PHP Coding Standards Fixer," GitHub, Jan. 28, 2023. https://github.com/FriendsOfPHP/PHP-CS-Fixer (accessed Jan. 29, 2023).

[7] "PHPStan - PHP Static Analysis Tool," GitHub, Jan. 28, 2023. https://github.com/phpstan/phpstan (accessed Jan. 28, 2023).

[8] "subprocess — Subprocess management — Python 3.8.5 documentation," docs.python.org. https://docs.python.org/3/library/subprocess.html

[9] T. F. and J. Wharton, "dominate: Dominate is a Python library for creating and manipulating HTML documents using an elegant DOM API.," PyPI. https://pypi.org/project/dominate/ (accessed Jan. 29, 2023).

[10] "docker run," Docker Documentation, Nov. 15, 2019. https://docs.docker.com/engine/reference/commandline/run/