TOOL FOR CHECKING CODE COMPATIBILITY WITH PROGRAMMING LANGUAGE RELEASES

Jehan Ryan Benjamin

179308M

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

May 2019

TOOL FOR CHECKING CODE COMPATIBILITY WITH PROGRAMMING LANGUAGE RELEASES

Jehan Ryan Benjamin

179308M

This dissertation submitted in partial fulfillment of the requirements for the Degree of MSc in Computer Science specializing in Software Architecture

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

May 2019

DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works.

....
Jehan Ryan Benjamin Date

The above candidate has carried out research for the Masters thesis under my supervision.

....
Dr. Indika Perera Date

ABSTRACT

Programming languages tend to evolve with new methodologies, user requirements, technology, security constraints etc. In order to stay relevant with changing demands, programming languages needs to keep adopting new features and enhancements. When programming languages make a new version release, it creates a necessity to upgrade the systems developed using older versions. Based on the situation of the project status, ongoing, completed or new, developers need to make the decision whether to proceed with the current version or change to new release. If the developers decides to stay with the older version of release they wouldn't get the latest features and fixes, eventually the system code would be outdated. This would lead the system to be vulnerable and prone to compatibility concerns.

The main intention of this research is to develop a simplified tool for checking code compatibility along with programming language version releases. Currently there few ways that developers check the compatibility (e.g. IDE plugins, CLI commands, compile time reports, etc.) but these are with limitations and not user friendly, aim of this research is to provide a combination of those features, in a more easy to use, optimized, customizable compatibility tool. The users will be able to run compatibility test on the desired version effortlessly, developers would be able to add their own custom rule sets to verify the project code across the selected version by using the tool. Once the tool completes the validation process, it will generate a user friendly report with the findings. The report would contain charts with error types and percentages, filenames, error line number, location, error type and possible fixes, which would be useful for developers. The developers would be able to fix the notified sections and re-run the verification process. The tool will also provide the option to get a deployable image along with updated code and version.

Keywords: programming languages, versions, code compatibility, deployable image.

ACKNOWLEDGEMENT

I like to convey my appreciation and gratitude to the project supervisor Dr. Indika Perera, for the knowledge, guidance and suggestions during this research project which was a driving force behind the completion of this work on time.

My heartfelt gratitude to my friends especially to Mr. Tiran Wijesekara, Miss. Tirsha Melani, Mr. Vilochane Vidyarathne, and all other friends which were not mentioned here whose friendship, hospitality and support in the preparation during this research.

I'd like to thank my current company Netstarter (Pvt) Ltd, especially to my project manager and colleagues for the endless support and words of encouragements throughout.

Finally, I would like to extend my deepest gratitude to my parents for their never ending support and encouragement.

TABLE OF CONTENTS

DECLARATION i

ABSTRACT ii

ACKNOWLEDGEMENT iii

LIST OF FIGURES vi

LIST OF TABLES ix

LIST OF ABBREVIATIONS x

CHAPTER 01: INTRODUCTION 1

1.1 Programming language Types 3

1.2 High-level languages 4

1.3 Programming Language Version 7

1.3.1 Benefits of version updates 8

1.3.2 Cost of updates 8

1.3.2 Dual version support 9

1.4 Backwards compatibility 9

1.5 Problem / Opportunity 11

1.6 Motivation 12

1.7 Objectives 12

CHAPTER 02: LITERATURE REVIEW 14

2.1 Programming language complications 15

2.2 Tools for sensing errors on software projects 16

2.3 PHP Code Fixer 18

2.3.1 Usage 19

2.4 Mining Software Repositories Tools 20

2.5 Deprecated methods 20

2.6 PHP built in lint command 21

CHAPTER 03: METHODOLOGY 23
3.1 System 24
3.1.1 High level architecture 25
3.1.2 Components architecture 26
3.2 Input output and processing 27
3.2.1 Custom rule sets 29
3.2.2 Type of notification outputs 30
3.2.3 Deployment process 30
4.1 Scope of implementation 32
4.2 Compatibility checker 33
4.2.1 Application implementation 33
CHAPTER 05: EVALUATION 49
5.1 Usability 50
5.1.1 Readiness check 50
5.1.2 Docker build, run process. 52
5.2 Application results evaluation 56
5.2.1 Code sniffer results 56
5.2.2 Evaluation of the test results 59
5.3 Reporting and error information verification 61
CHAPTER 06: CONCLUSION AND FUTURE WORK 63
Summary 64
6.1 Problems encountered 65
6.2 Future work 65
REFERENCES 67
APPENDIX A 70

LIST OF FIGURES

Figure 1: Computer language and its types [26] 3

Figure 2: Transitions of a High-level Language Program [30] 5

Figure 3: The fifteen most popular languages on GitHub [27] 6

Figure 4: Top 10 Programming Languages [3] 6

Figure 5: PHP error reporting options 10

Figure 6: PHP supported versions [9] 13

Figure 7: C++ Semantic error sample code 15

Figure 8: PHP Related Semantic Error Code Sample 16

Figure 9: Zend Studio Semantic Analysis [28] 16

Figure 10: PHP CodeSniffer findings against PEAR coding standards [11] 17

Figure 11: LGTM alert samples [12] 18

Figure 12: PHP Code Fixer terminal command 19

Figure 13: phpcf sample results [13] 19

Figure 14: Candoia work flow [15] 20

Figure 15: PHP Deprecated Methods Finding Script 21

Figure 16: PHP Lint script 22

Figure 17: Proposed work flow 24

Figure 18: Proposed System Architecture 25

Figure 19: Components architecture 26

Figure 20: system input output 27

Figure 21: Activity flow 29

Figure 22: Custom rules 30

Figure 23: package.json 34

Figure 24: Electron init code 35

Figure 25: Readiness check 35

Figure 26: Docker checker 36

Figure 27: Docker Install option 36

Figure 28: Docker information 36

Figure 29: Docker project info 37

Figure 30: Docker image create code sample 38

Figure 31: Container terminal output 39

Figure 32: Docker starting view 40

Figure 33: Docker and Project information 41

Figure 34: Docker image build 42

Figure 35: Dockerfile contents 42

Figure 36: supervisord.conf file 43

Figure 37: Composer package.json 44

Figure 38: Custom rule set 44

Figure 39: Pie chart 45

Figure 40: Error notification 46

Figure 41: Warning notification 46

Figure 42: Container diff 47

Figure 43: Docker deployment process [21] 48

Figure 44: log files 50

Figure 45: Application landing page 51

Figure 46: Docker information 51

Figure 47: Project info form 52

Figure 48: Docker image run and container start 53

Figure 49: container and project info 53

Figure 50: Docker executes and results 54

Figure 51: Phpstorm code sniffer setup 54

Figure 52: Docker build commands 55

Figure 54: webpack build scripts 55

Figure 53: platform related builds 55

Figure 55: Terminal summary output 57

Figure 56: report summary on application 57

Figure 57: misleading information 62

Figure 58: Updated pie chart view 62

LIST OF TABLES

Table 1: Types of High Level Language 4

Table 2: Compatibility checker Input and result output 27

Table 3: Sample projects information 56

Table 4: Compatibility results summary 58

Table 5: Compatibility results summary for 5.6 59

Table 6: Compatibility results summary for 7.0 60

Table 7: Results summary for 7.1 61

LIST OF ABBREVIATIONS

Abbreviation	Description
PEAR	PHP Extension Add-On Repository
MSR	Mining Software Repositories
PHP	Recursive acronym for PHP: Hypertext Preprocessor
CPU	Central Processing Unit
PM	Project Manager
BA	Business Analysis
IT	Information Technology
HTML	HyperText Markup Language
OOP	Object Oriented Programming
IDE	Integrated Development Environment
LGTM	Looks Good To Me
CLI	Command Language Interpreter
CSV	Comma Separated Values
XML	Extensible Markup Language
JS	JavaScript
CSS	Cascading Style Sheets
JSON	JavaScript Object Notation
Amazon ECS	Amazon Elastic Container Service

CHAPTER 01: INTRODUCTION

Programming languages tend to develop gradually over time in response to user domain requirements, hardware advances, and improvements done from research developments. A programming language expansion artifact may include new compilers and interpreters or new language standards. It is a challenging task for developers when programming languages keep evolving at various levels. Firstly, the impact on developers can be negative. For example, if two language versions are incompatible (e.g., Python 2 and 3) developers must choose to either co-evolve with codebase (which may be costly) or reject the new language version (which may have support implications). Secondly, evaluating a proposed language change is difficult; language designers often lack the infrastructure to assess the change. This may lead to older features remaining in future language versions to maintain backward compatibility, increasing the language's complexity (e.g., FORTRAN 77 to Fortran 90). Thirdly, new language features may interact badly with existing features, leading to unforeseen bugs and ambiguities (e.g., the addition of Java generics). [1]

Mature programming languages such as C, Java, Python change less frequently, with a comparison to other recently introduced programming languages. Commonly used, popular programming languages have a clear evolution, marked by its version: for Java 5, 6, 7 etc., PHP 5.5, 5.6, 7.0 etc. Along with major releases comes new APIs, fixes bugs, adds new features, and new frameworks.

A common issue that developers face is choosing a version of a programming language to use for a particular software application project. Some developers may choose the latest version of a language as the best solution, but newer version would not always be the better option. Selecting a wrong version of a language would lead the downfall of the project to almost certain failure, or at least make the success of a project much more difficult. To avoid the risks of choosing a wrong version, it's beneficial to avoid dangerous assumptions, understand the benefits and costs of language updates, examine your own personal preferences, and evaluate the findings.

1.1 Programming language Types

There are variety of programming languages types available for developers, each language having its own type of methodology and implementation. Different types of languages serve different purposes, some types are:

- **Machine languages**, which are interpreted directly in hardware to be executed by a central processing unit (CPU).
- **Assembly languages**, a low-level programming language for microprocessors and other programmable devices, that works as a thin wrapper over a corresponding machine language.
- **High-level languages**, which are anything machine-independent (C, FORTRAN, or Pascal).
- **System languages**, which are designed for writing low-level tasks, like memory and process management.
- **Scripting languages**, these programs are written for a special runtime environment that automates the execution of tasks that could alternatively be executed one-by-one by a human operator that is generally extremely high-level and powerful.
- **Domain-specific languages**, only used for highly special-purpose domains.
- **Visual languages**, these are non-text based language, that lets users create programs by manipulating program elements graphically.

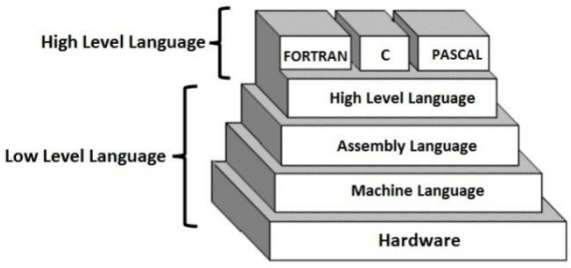

Figure 1: Computer language and its types [26]

1.2 High-level languages

Very early in the development of computers attempts were made to make programming easier by reducing the amount of knowledge of the internal workings of the computer that was needed to write programs. If programs could be presented in a language that was more familiar to the person solving the problem, then fewer mistakes would be made. High-level programming languages allow the specification of a problem solution in terms closer to those used by human beings. These languages were designed to make programming far easier, less error-prone and to remove the programmer from having to know the details of the internal structure of a particular computer. These high-level languages were much closer to human language. [2]

Table 1: Types of High Level Language

Type	Description	Examples
Procedural Languages	Series of well-formed instructions are run to compose a program. There is a starting point and a logical order to the instructions to be executed, until the endpoint is reached. It uses program control constructs (If, Then, Loops, Subroutines, and Functions).	PASCAL, BASIC, FORTRAN, COBOL
Event-driven languages	A program that listens to user events such as the clicking of the mouse or the press of a key on a keyboard. When an event is triggered, it is processed using a defined sequence of instructions called an event handler. Useful for control programs where events such as readings from sensors are used to control devices.	Visual Basic, C++, Javascript
Object-Oriented Language	Defined objects have properties and methods. These properties can be set initially or changed at run-time. The things that an object can do are specified in methods.	Visual Basic, C#, JAVA, PYTHON
Markup Languages	These are designed for the processing, definition and presentation of text. It specifies code for formatting, both the layout and style, within a text file. The code used to specify the formatting are known as tags. HTML is a widely known and used markup language.	HTML. XML, XHTML, ASP.

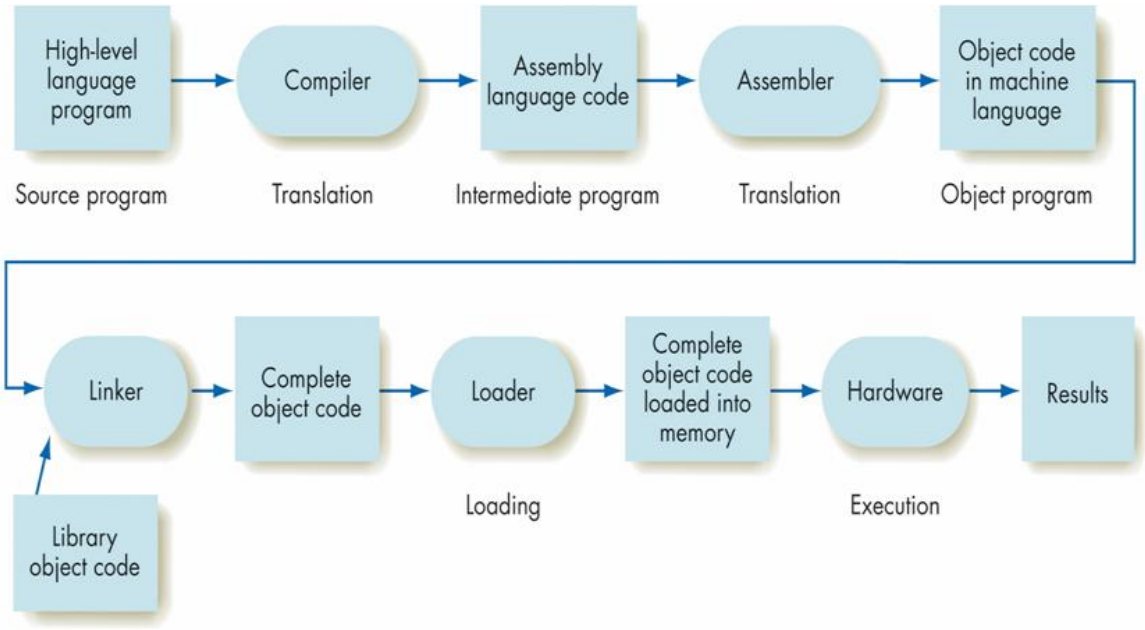

Figure 2: Transitions of a High-level Language Program [30]

Each language has its own characteristics, advantages, and disadvantages. It's critical to choose the right language for the project depending on mid-term, long-term goals and overall expectations. When selecting a programming language, it is good to be in trend, but need to make sure all project stakeholders (e.g., product owner(s), PMs, BAs, IT Manager) agree regarding use of certain languages and technologies for frontend, backend and data warehousing. Conduct internal and external audits to make sure the chosen languages don't have conflicts with current corporate systems, operating systems, applications, etc. Using interpretive, dynamic, open source languages developers can achieve speedy development and more cost-effective solutions. Technologies with high-security standards and abilities to integrate with legacy systems/environments chose for enterprise application or large scale projects. Following figure was taken from Github (a popular repository), shows that Javascript is the most trending programming language and also Python language has passed Java to become the second most popular language. Typescript has also increased in reputation. Rust, Google Go, Kotlin are some of the upcoming programming languages which are not listed.

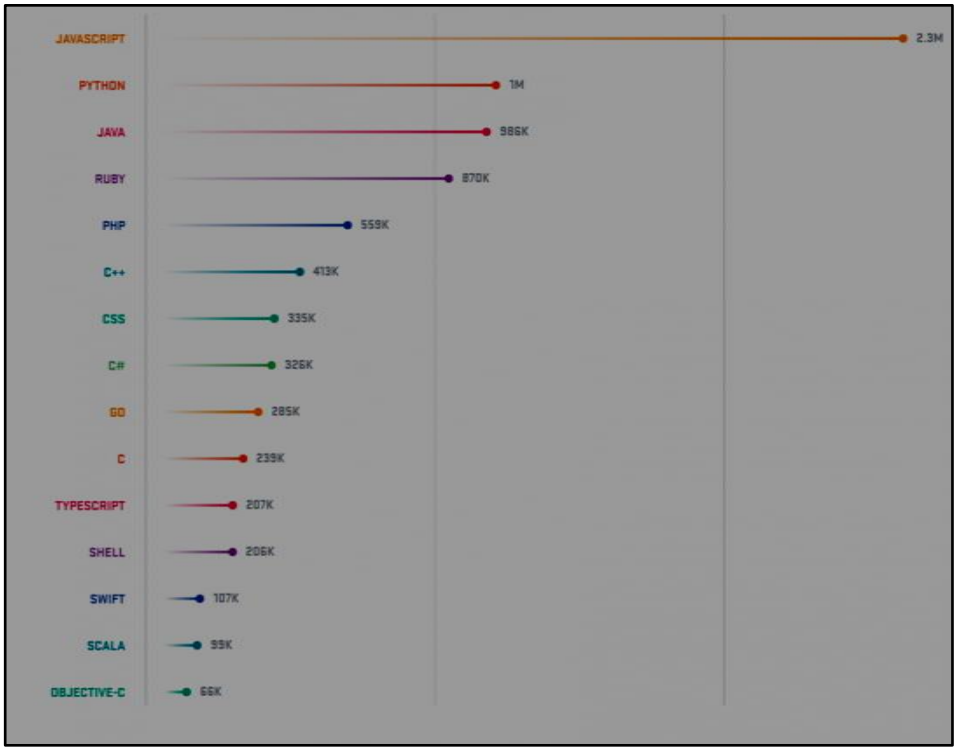

Figure 3: The fifteen most popular languages on GitHub [27]

Following figure show top 10 programming languages and their information on implementation, history, file extension etc. [3]

Figure 4: Top 10 Programming Languages [3]

1.3 Programming Language Version

Developers need to select a suitable version of the programming language in order to develop the proposed solution. When selecting a version, developers often make the mistake of choosing the latest assuming that it's the best choice, in some cases, it could lead to project failure causing time and money. As a developer rather than making assumptions, understanding the benefits costs associated with updates, reliability, security, ease of use, etc. would be advantageous.

There are many cases that when developers have to make decisions, when the programming language makes a release, whether to apply the changes existing application. The developers are given the choice of keeping the same version of the language used in the original or to update. Older versions of a language may not include the latest security features or may lack other features, requiring developers to invest more time in implementing further solutions. There could be patches providing latest features to be used in older versions but eventually, that version going to be outdated.

Another case is when the developers get to create a new application, a commonly preferred choice would be to rely on the latest release of the language. However, new versions of a language may not have widespread library support or fully tested functionality. [4] Selecting a more stable release would be the smarter decision.

Integrating application environments, organizations often force developers to use their favored language version regardless of whether it's the best choice for the particular application at hand. Standardizing on a particular language version across the organization makes it easier to reuse tooling and libraries, and can help developers avoid being tripped up by version differences when they move from one project to another. However, it can also result in lost flexibility, an inability to use existing code, and wasted developer time to rewrite code in the preferred language version. [4]

1.3.1 Benefits of version updates

Programming language update releases are intended to increase the current functionality in some form, example Microsoft Entity Framework. Each new version of Visual Studio comes with an updated version of the Entity Framework, every update delivering major improvements in functionality. C++ is another example offering substantial new functionality.

Updates could also make the language more consistent and reliable removing all the confusing codes. For example: Python 2.7 supports print as a command as well as a function, leading to confusing code. Python 3.4 consistently views print() as a function.

HTML5 and JavaScript have become so popular due to increase of reliability and fastness, the combination of those two is becoming the preferred language for mobile applications. Continuous updates have made it possible to create an application that works on most of the devices both quickly and reliably.

Security is a major concern for programming languages and almost all the commonly used languages release security patches whenever there is vulnerability. Java, Python, PHP are some of the examples.

1.3.2 Cost of updates

Version updates clearly have its advantages and disadvantages, most consequential cost of an update is backward compatibility. There can be cases where features provided by the previous version are no longer available and removed without any replacements. In worse case may end up with an unusable project. There could be hidden costs in terms of support and language offers. For example: Python 2.7 provides library support that 3.4 version hasn’t adopted. Also in Python from 2.7 to 3.4 print() considered as a function and all occurrences in the project needs to be fixed.

Deprecated methods are another cost since developers have to pay particular attention since there could be less helpful error messages. For example PHP version 5.4 to 5.5, there were some significant amount of changes which faced the deprecated function issue.

1.3.2 Dual version support

There are some languages that officially provide support for multiple versions parallelly. For example: Python provides support for 2.7 and 3.4 and PHP provide support for version 5.6 and 7.x at the same time. These supports on older releases will eventually stop and there wouldn't be any option other than updating to the latest.

1.4 Backwards compatibility

Mature languages change less frequently. For mainstream languages, such changes should be planned in a very careful way to avoid the invalidation of the existing large code base. Therefore, the modifications are typically backward compatible, i.e. the earlier written code should compile with the new version of the compiler and the meaning of the old code should not change. This is a major requirement as it is unrealistic that the maintainers of a large code base execute a full code review on the source to detect whether certain code parts are affected by the changes. [5]

When the evolution of the language requires some of the features in the previous code not to be used in the latest version, then it releases backward incompatible changes. The selected function will be *deprecated,* which means those are available for use in the current version but developers are advised avoid the use of those elements. In order to easily find out deprecated language elements, compilers emit the diagnostic messages over them, for web based programming languages the notification will be visible on the browser this depends on the error settings applied by the developer. For example in PHP the developers have the option to enable or disable the errors visible on the browser.

```
error_reporting = E_ALL & ~E_DEPRECATED & ~E_NOTICE
```

Figure 5: PHP error reporting options

Project maintainers should need to make sure the changes are done and no deprecated elements are in use, since the deprecated elements could be officially removed from language and remaining code referring would throw syntax errors.

The most serious problems related to language evolution come from situations where the syntax of the old language version remains valid in the new version but the semantics i.e. the meaning of the source code changes. In such cases no compiler diagnostics are produced, nothing warns the maintainers for the dangerous difference. The source code is re-compiled with the new compiler version supporting the new language version without error messages or even warnings. When the program will be executed, however, one can observe completely different results. To catch such situations is very difficult even when full regression test coverage exists. Naturally, mainstream languages try to avoid such backward incompatible semantic changes. However, even with the best intentions, such situations happen regularly. [5]

Programming languages handle compatibility in various ways. While some languages strictly release incompatible versions, others languages release fully compatible versions, also there are other languages that use third party software libraries/tools to make the code compatible.

Java is a strongly compatible programming language with forward backward compatibility. Source compatibility concerns translating Java source code into class files including whether or not code still compiles at all. Binary compatibility is defined in The Java Language Specification as: 'A change to a type is binary compatible with (equivalently, does not break binary compatibility with) pre-existing binaries if pre-existing binaries that previously linked without error will continue to

link without error.' Behavioral compatibility includes the semantics of the code that is executed at runtime. [6]

Scala uses a multi-paradigm design, which uses general programming concepts and patterns in an optimized manner, if a developer built a program using Scala version 1.2, it won't be compatible with version 1.3. If a library is compiled using 1.2 and tries to run it on an application with a version 1.3, it will simply not work and the developers have to wait until the next release comes. Scala have improved their way of releases by using a schema EPIC.MAJOR.MINOR and guarantees compatibility between minor releases.

Unlike other programming languages, JavaScript depends on the browser compatibility. If a programmer writes a code in ECMAScript 2015 (ES6), older browsers would not be able to understand the ES6 and generate issues. In order to resolve this browser compatibility issue, initially developers used ES6 to ES5 using *Babel*, but this was not the most efficient solution since babel only transforms syntax (e.g. arrow functions). There is a polyfill solution provided by Polyfill.io, [7] which can polyfill ES6 code in the client side browsers, even though this is a solution for an older browser, browsers that supports ES6 will be still sending for server requests. Dynamic-polyfill checks whether the required features are natively supported before making any server side requests. [8]

1.5 Problem / Opportunity

"Don't fix it if it isn't broken". Most of the system owners and developers have this mind set, but when it comes to evolving requirements and technologies the systems / software should not be stagnated. This research study is focused on compatibility between version releases for programming languages. Not all the software systems are built and maintained up to date. System owners tend to notice the system vulnerabilities and failures when there are complaints from the users, this could be costly, affecting the business and trust. Clients look to update the system to latest version only when all the other approaches fail. The main problem would be that the

system could be so outdated a simple version upgrade would not be possible and building the system from scratch would be time consuming and costly.

1.6 Motivation

Programming languages tend to evolve; there will always be a cost of update when changing from one version to another. Code compatibility across different programming language versions are handled by developers. The developers tend to use compile time logs, code sniffers, tools such as selenium or execution time errors to find code compatibility issues. This process is time consuming and convincing the client for an upgrade will be difficult. The outcome of this research is to provide a tool that could easily search through the project code to identify compatibility issues and provide a full report to the user, making the transition from one version to another easy and with less update cost. The compatibility tool is intended to be user friendly allowing any person to run the program and generates a report. It would also provide advance features for developer to set their own rule sets to verify and create a deployable image along with updated codebase and updated programming language version.

1.7 Objectives

The main objective of this research is to provide a tool to check code compatibility across programming language version releases. The proposed tool is expected to cover all the areas related to code validation. Code standards, syntax errors, deprecated methods, user specific validations. Along with the main objectives following areas would be covered from the compatibility tool.

- Listing of available versions that are stable and supported from the vendors.
- Simple user friendly tool which could be used without any prior experience.
- Multiplatform support, the tool can be used across windows, apple or any linux OS.
- Provide user friendly graphs and tables in the report.
- Users should be able to update the files and re-run the checker.

- The users should be able to get a deployable image which could be deployed to server.
- Another objective of this tool is to read the project code from the repositories, since majority of developers use repositories for their project, it is essential have the option to read from the repositories as well as from local drive.

Since this is a 6 months research, due to time constraints the scope of the research has been narrowed down to one programming language and its releases. The research will be based on PHP language and its compatibility with the releases. The reason for selecting PHP was it is one of the leading software language used for developing web based solutions and has completed a major version change from 5.6 to 7.0. Currently PHP supports 5.6.x, 7.0.x, 7.1.x, 7.2.x and 7.3 Also 7.4 will be release on December 2019.

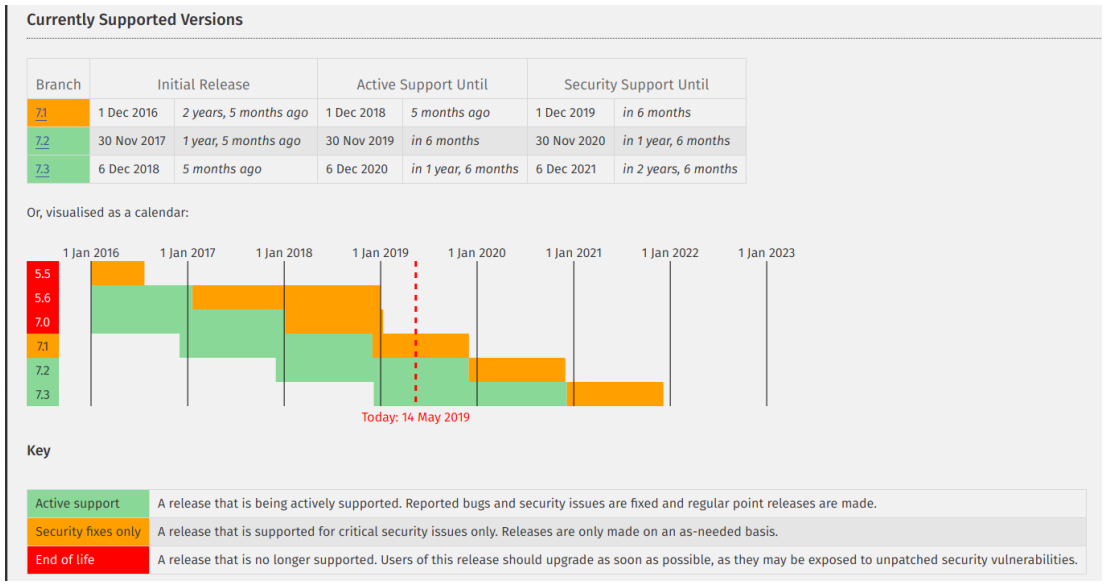

Figure 6: PHP supported versions [9]

CHAPTER 02: LITERATURE REVIEW

Many programming languages have their own way of handling language release. There are research done based on specific languages and platforms. This section is to identify the importance of those researches based on the compatibility tool requirements. There are few research work done on checking for compatibility and language evolution related issues. They were single language specific but were helpful for the cause of this research.

2.1 Programming language complications

The most consequential problem in relation to language evolution is different semantics related between versions. This occurred in C++11 and the previous version for C++ language. One can recognize a completely different result to catch such situations is very dubious even when an almost full regression test coverage exists. [5] Following code sample below has non-identical semantics in C++11 and pre C++11 versions. The output of the previous version would be 12345 while other version prints 11111.

```
struct S {
    S() : i(++counter) {}
    static int counter;
    int i;
};
int S::counter = 0;
int main() {
    std::vector<S> v(5);
    for (std::size_t x=0; x<v.size(); ++x)
        std::cout << v[x].i;
}
```

Figure 7: C++ Semantic error sample code

PHP 3 had a clear semantic that assignment, argument passing, and returns are all by value, creating a logical copy of the data in question. The programmer can opt into reference semantics with a "&" annotation. [2] There were conflicts with the introduction of object-oriented programming in PHP 4 and 5 against these types of references, even though PHP object-oriented notations were taken from Java programming language. Java uses semantic objects are treated by references, while primitive types are treated by value. [10]

```
$a = goldbachs_conjecture() ? 3.14159 : "a string";
```

Figure 8: PHP Related Semantic Error Code Sample

There are IDEs that support semantic analysis, Zend studio supports semantic analysis for PHP projects. The semantic analysis feature enables warning and error messages to be displayed on the project code. Also, it allows the developers to see compile errors and potential programmer errors.

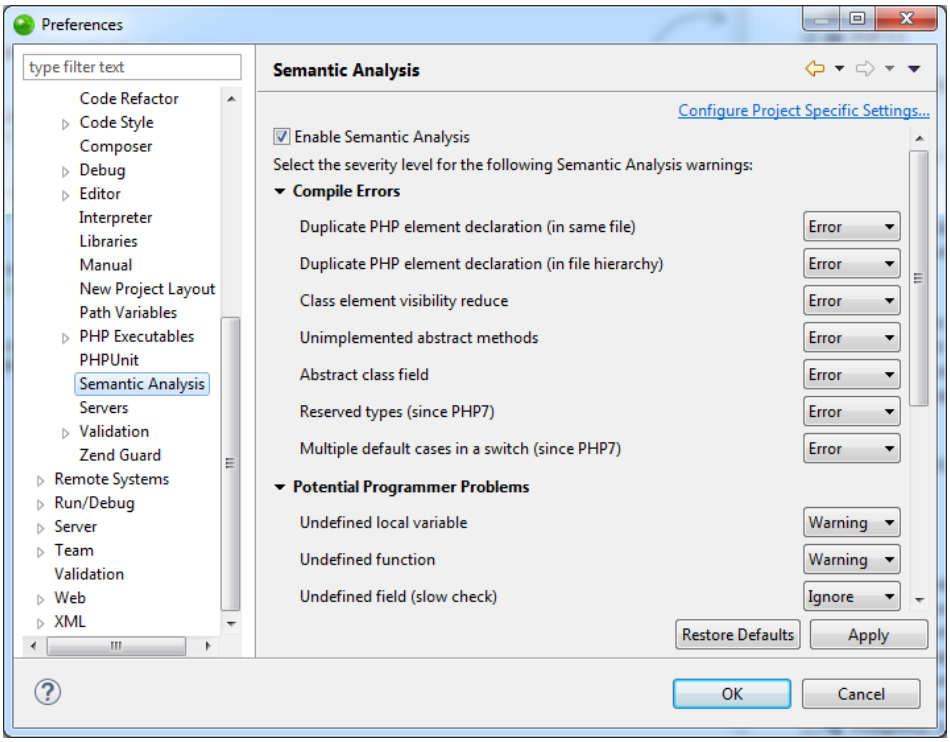

Figure 9: Zend Studio Semantic Analysis [28]

2.2 Tools for sensing errors on software projects

Code sniffers are a commonly used plug-in for many IDE's. "*PHP_CodeSniffer*" is a set of two PHP scripts; the main phpcs script that tokenizes PHP, JavaScript and CSS files to detect violations of a defined coding standard, and a second phpcbf script to automatically correct coding standard violations. PHP_CodeSniffer is an essential development tool that ensures your code remains clean and consistent. [11]

A coding standard in PHP_CodeSniffer is a collection of sniff files. Each sniff file checks one part of the coding standard only. Multiple coding standards can be used within PHP_CodeSniffer so that the one installation can be used across multiple projects. The default coding standard used by PHP_CodeSniffer is the PEAR coding standard.

```
$ phpcs /path/to/code/myfile.php

FILE: /path/to/code/myfile.php
----------------------------------------------------------------------
FOUND 5 ERROR(S) AFFECTING 2 LINE(S)
----------------------------------------------------------------------
  2 | ERROR | Missing file doc comment
 20 | ERROR | PHP keywords must be lowercase; expected "false" but found "FALSE"
 47 | ERROR | Line not indented correctly; expected 4 spaces but found 1
 51 | ERROR | Missing function doc comment
 88 | ERROR | Line not indented correctly; expected 9 spaces but found 6
----------------------------------------------------------------------
```

Figure 10: PHP CodeSniffer findings against PEAR coding standards [11]

"*Looks Good To Me*" (LGTM), this tool could be used as the last step for verifying the code base before moving to live. It checks for potential known bugs and evaluates the design. At the moment LGMT analyzes code bases written in Java, JavaScript, and Python. This tool uses the code in the repository with the along with its revisions to analyze the changes that have been done. [12]

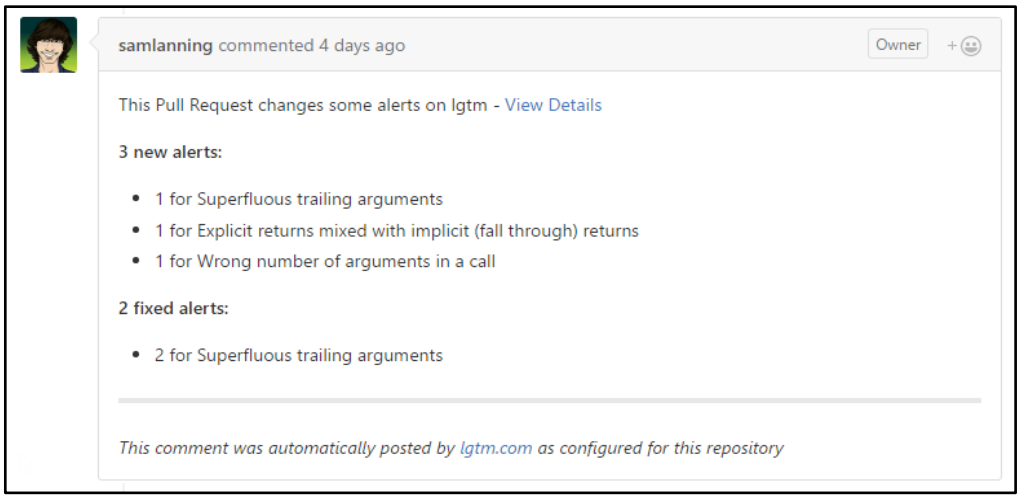

Figure 11: LGTM alert samples [12]

2.3 PHP Code Fixer

As the PHP evolve over time the vendors deprecates functions and remove them in later versions. These functions and variables were used in projects and needs to be removed when upgrading to newer version. Code fixer parse PHP code to find issues in functions, variables. It checks the php.ini configuration directives that are deprecated. It can also suggest replacements for the code that uses deprecated features. [13]

PHP code fixer is a terminal checker that scans for compatibility of the project code across the latest supported versions. This tools helps developer finds the following usage:

- Identifies the use deprecated features in PHP projects (ini-directives / constants / variables / functions).
- Identifies changed behavior compared to later versions.

This tool provides suggestions to the users on the areas that need to be updated. Using the composer or Phar file users can install and start using the PHP Code Fixer tool.

2.3.1 Usage

After successful installation users can run the "**phpcf**" command in the console and pass file or directory names. This phpcf allows users with following options:

- Set the PHP interpreter version; exclude file or directory names for scanning.
- Skip large files by setting maximum file size, to ignore larger files that would increase the execution time.
- Set specific file extensions to be scanned. [default: "php, php5, phtml"] [13]
- Files can be skipped based on the given values.
- Get the results as a JSON output file.

```
$ phpcf      [-t|--target [TARGET]]
       [-e|--exclude [EXCLUDE]]
       [-s|--max-size [MAX-SIZE]]
       [--file-extensions [FILE-EXTENSIONS]]
       [--skip-checks [SKIP-CHECKS]] [--output-json [OUTPUT-JSON]]
```

Figure 12: PHP Code Fixer terminal command

Following is an output of the test carried out from "phpcf".

```
- PHP 7.2 (7) - your version is greater or equal
+------------+---------------+-------------------------------------------------------------------------+
| File:Line  | Type          | Issue                                                                   |
+------------+---------------+-------------------------------------------------------------------------+
| /7.2.php:2 | function      | Function "create_function()" is deprecated.                             |
| /7.2.php:7 | function      | Function "read_exif_data()" is deprecated.                              |
| /7.2.php:14| function      | Function "each()" is deprecated.                                        |
| /7.2.php:9 | constant      | Constant "INTL_IDNA_VARIANT_2003" is deprecated.                        |
| /7.2.php:3 | ini           | Ini "mbstring.func_overload" is deprecated.                             |
| /7.2.php:5 | function_usage| Function usage "assert() (@assert_on_string)" is deprecated.            |
| /7.2.php:12| function_usage| Function usage "parse_str() (@parse_str_without_argument)" is deprecated.|
+------------+---------------+-------------------------------------------------------------------------+

- PHP 7.3 (2) - your version is greater or equal
+-----------+---------------+-----------------------------------------------------------------------+
| File:Line | Type          | Issue                                                                 |
+-----------+---------------+-----------------------------------------------------------------------+
| /7.3.php:3| constant      | Constant "FILTER_FLAG_SCHEME_REQUIRED" is deprecated.                 |
| /7.3.php:2| function_usage| Function usage "define() (@define_case_insensitive)" is deprecated.   |
+-----------+---------------+-----------------------------------------------------------------------+

Total problems: 28

Replace Suggestions:
1. Don't use function mcrypt_generic_end() => Consider replace to mcrypt_generic_deinit().
2. Don't use function read_exif_data() => Consider replace to exif_read_data().
3. Don't use function each() => Consider replace to foreach().
4. Don't use ini mbstring.http_output => Consider replace to default_charset.
5. Don't use variable $HTTP_RAW_POST_DATA => Consider replace to php://input.
6. Don't use constant INTL_IDNA_VARIANT_2003 => Consider replace to INTL_IDNA_VARIANT_UTS46.

Notes:
1. Usage piet() (@call_with_passing_by_reference): Call with passing by reference is deprecated. Problem is "&$hoho"
2. Usage preg_replace() (@preg_replace_e_modifier): Usage of "e" modifier in preg_replace is deprecated: "asdasdsd~ie"
3. Usage password_hash() (@password_hash_salt_option): "salt" option is not secure and deprecated now
4. Usage mb_ereg_replace() (@mb_ereg_replace_e_modifier): Usage of "e" modifier in mb_ereg_replace is deprecated: ""msre"
5. Usage assert() (@assert_on_string): You should avoid using string code: "'false'"
6. Usage parse_str() (@parse_str_without_argument): Call to parse_str() without second argument is deprecated
7. Usage define() (@define_case_insensitive): Case-insensitive flag of define() is deprecated, use original constant name
Peak memory usage: 1.928 MB
```

Figure 13: phpcf sample results [13]

2.4 Mining Software Repositories Tools

Data mining is one of the most relevant topics in current day's research areas. Based on the knowledge gather from data mining from repositories, data miners are able to predict models and patters. This information could be used in various domains. [14]

Candoia uses Mining Software Repositories tools inside are ecosystem. The platform uses MSR as built apps. These apps can "build once, run everywhere". [15] Following is a figure of how candoia works.

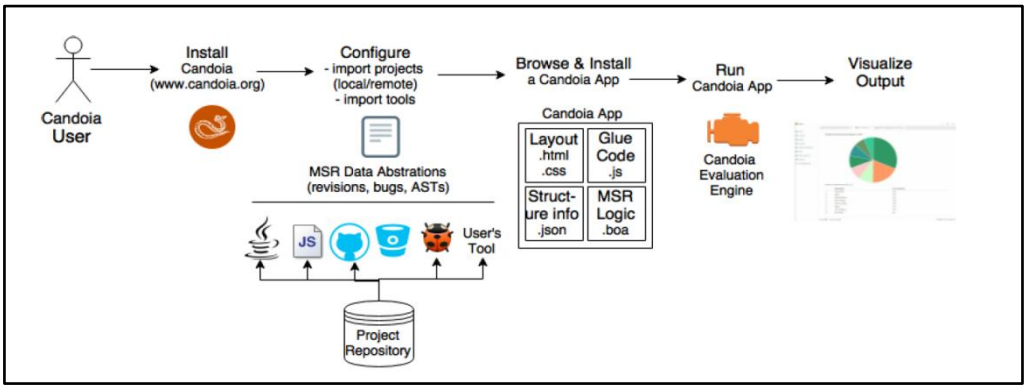

Figure 14: Candoia work flow [15]

2.5 Deprecated methods

There are many IDE's tools that find out deprecated and also there are scripts written by developers that are published. Depcheck is simple application script to find out deprecated methods on a file and suggest replacements. [16] Once a deprecated method had been found from a file if finds a replacement code from the latest version for a suggestion. In order to check through a full PHP project, the script file can be modified to iterate through the project PHP files and detect deprecated methods. The class file is able to read from a text file or a CSV for deprecated methods which can be updated as the release comes.

```
<?PHP
include('depcheck.class.php');
```

```
$fileName = ( !empty($_REQUEST['f']) ) ? $_REQUEST['f'] : '';

$dpc = new depcheck($fileName);
$dpc->checkFile();

?>
<html>
    <head>
        <title>PHP deprecated function checker</title>
    </head>
    <body>
<?PHP
if( $dpc->errorFlag === true ){
?>
<div style="margin: 10px 0; color: red;">Error!!!<br><?PHP echo
$dpc->errorMessage;?></div>
<?PHP
}
?>
<div>Running PHP version <?PHP echo PHP_VERSION;?></div>
        <div>Using deprecated csv file '<?PHP echo $dpc-
>depFile;?>'</div>
        <div>Checking file '<?PHP echo $dpc->fileName;?>'</div>
        <div style="margin-top: 10px;">Results:<br><?PHP echo
$dpc->resultMessage;?></div>
    </body>
</html>
```

Figure 15: PHP Deprecated Methods Finding Script

2.6 PHP built in lint command

In order to find syntax errors through the command line, PHP has introduced the lint command (php -l). The programmers could run this on a terminal interface and it will output the syntax error and file information on the terminal. Following is a script which was developed to iterate through project folder and find PHP and PHTML files.

```
#!/bin/bash

FOR file in “find .”
DO
    EXTENSION = "${file##*.}"
    IF [ "$EXTENSION" == "php" ] || [ "$EXTENSION" == "phtml" ]
    THEN
        RESULTS =”php -l $file”
```

```
        IF [ "$RESULTS" != "No syntax errors detected in $file" ]
        THEN
            echo $RESULTS
        FI
    FI
done
```

Figure 16: PHP Lint script

Based on the literature review findings, author was able discover that there are few extension and tools to identify specific areas related to code standers and errors. These solutions were focused on fixing or identifying specific problem areas. In order to use them on projects, developers needed to have special IDEs which supported those tools or knowledge on how to execute them manually using the terminal. This research is focused on providing a tool that covers all the aspects related to code validation. The solution would be implemented using features taken from PHP code sniffer and code fixer.

CHAPTER 03: METHODOLOGY

Main objective of this research is to provide a tool to help the developers easily find out system code compatibility in the latest programming language versions. Read the project code from the local drive or from the repository making the tool portable and accessible.

The project scope is limited to one programming language and its releases due to the research time constraint. The methodology of the solution will be specific to PHP language, but this research can be extended to support many other programming languages in the future. This section provides a conceptual overview of the approach, discussing the representation and generation of compatibility reports from PHP code.

3.1 System

The high level of information flow view is represented in figure 3.1. The user will be given the option to select the project location either from local drive or from the repository. Once the project file location is added, user need to select the version that need to be checked, user will be given the available PHP releases to select. Once the required information are given user will be able to execute the checker which would inspect the code and generate the findings on the dashboard. The user will be able to make changes to the project code and re-run the testing. Once the issues are fixed the user has the option to generate a deployable image using the tool.

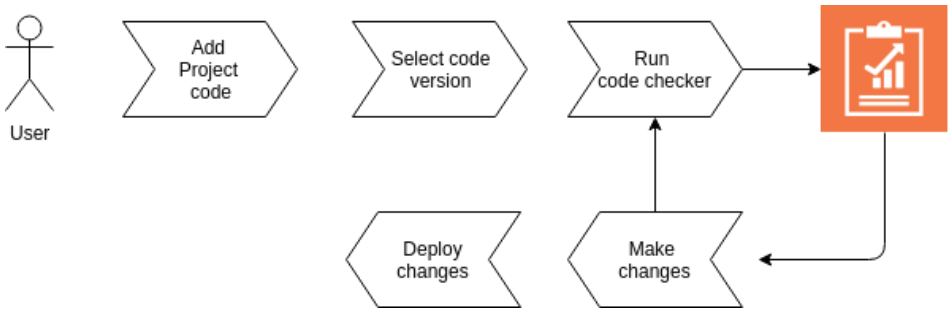

Figure 17: Proposed work flow

3.1.1 High level architecture

Figure 18 illustrate the top tier architecture in the code compatibility tool. As shown on the figure, the user will be able to provide the code either from local drive or git repository. The provided code will be checked depending on the programming language version that the user selected. The project code will be inspected based on the version and custom rules set by the user. Based on the rules, the tool will generate a report on the application dashboard and a deployable image. The report will have multiple types of notifications, syntax errors, deprecated, warnings, and notice. Users also have the option to add coding standards to be verified during the inspection. Inspection process uses the code sniffer to identify errors on the code, this would be executed internally inside the container using execution calls. The output is available on the dashboard displayed as graphs and tables. The use will be able to edit the identified files and re-run the checking process. The files will update inside the container without affecting the original project code. Once the correction process is completed the user will able to generate a output to compare the difference between the original project code and updated code. The user also has the option to create a docker base image with the updated code along with selected programming language version.

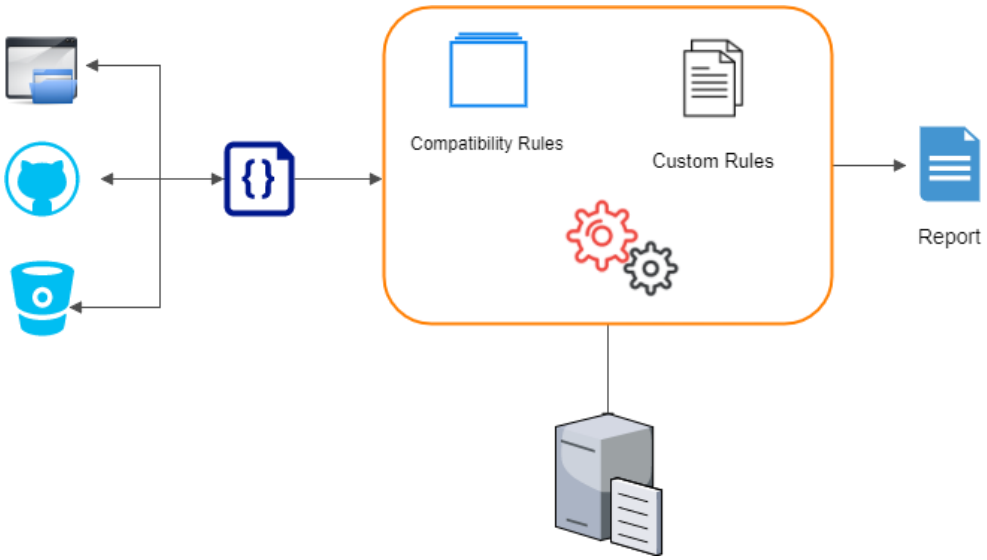

Figure 18: Proposed System Architecture

3.1.2 Components architecture

Figure: 19 illustrate component architecture of the solution. Compatibility tool consists of multiple components working together to generate the desired output. The main two components are the application and docker processes.

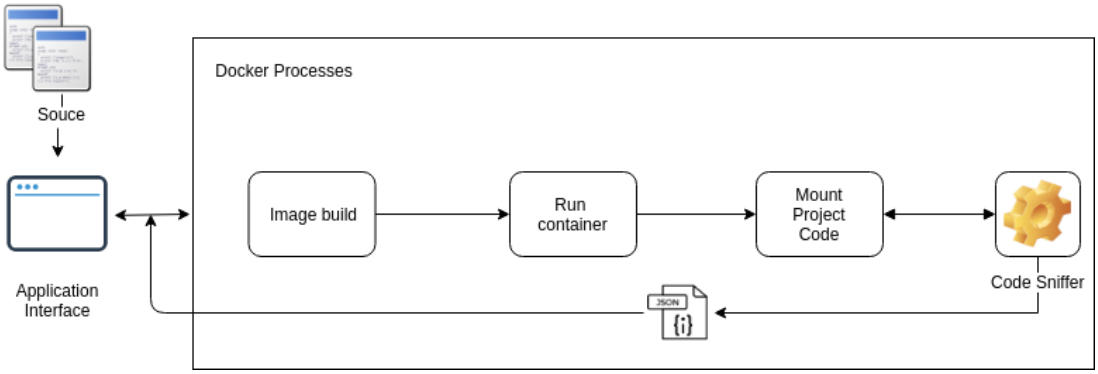

Figure 19: Components architecture

3.1.2.1 Application interface

All the user interactions will be handled through the application interface. Docker processes will be triggered from the application interface based on the user interactions. The application aims to be user friendly fulfilling the requirements. The application is intended work on multiple platforms, the tool can be used across windows, apple or any linux os.

3.1.2.2 Docker Processes

The application is depended on docker, it uses docker image build and containers to mount the project code base and execute the code sniffer rules. Since this is based on docker containers the users have full access to containers, allowing them to customize. The docker container consists of required extensions for PHP, open ports and nginx server, so that the users could access the project from the browser if they desire.

3.2 Input output and processing

Figure: 20 illustrate the abstract input and output of the proposed tool. Code compatibility tool verify the provided code along with the rule set assigned to the code sniffer. Once the verification process completed the output will be written as json to a file. This would be used as the source to generate the report and dashboard data.

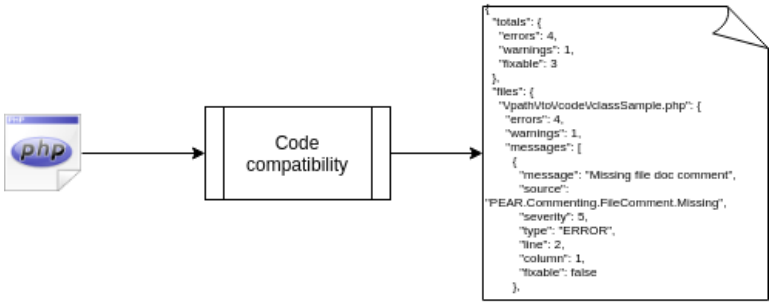

Figure 20: system input output

Following table is a sample input and output of the compatibility tool. When a user triggers a verification process from the application, it will iterate through the mounted project code and write the findings to the results.json file. The application will use the json data to notify the user on the findings and generate graphs and tables on the dashboard.

Table 2: Compatibility checker Input and result output

Context	Value
Input	/camera-hut/codepool/app/Sales/Orders/Block/order.php

Output	results.json output file contains the following output.
	
	{
	"totals": {
	"errors": 4,
	"warnings": 1,
	"fixable": 3
	},

In order to get the results.json file, the application executes multiple processes. Figure: 21 illustrates the activity flow of the application.

01 - The user will be given an option to start the proceedings. Starting the readiness check will trigger a prerequisites check. The application depends on docker installation, during the readiness check, it will check for existing docker installation and its version.

02 - The Docker installation will be triggered based on the operating system the installation process may vary.

03 - The user will be able to provide the project information such as project name, project folder location, required PHP version. This information will be gathered and used for the docker image build and container startup.

04 - Docker image build up and container will be started on this step. The output of this process will be visible on the application and written written to a log file.

05 - During this step user will be able to access the started container along with project information. The project code will be mounted and will provide a count of total php / phtml files found on the project.

06 - This step will execute the code sniffer rules set, providing steam of data to the JSON output file.

07 - Once the execution completes, the JSON file will hold the results. This files will be use to generate graphs and tables on the report.

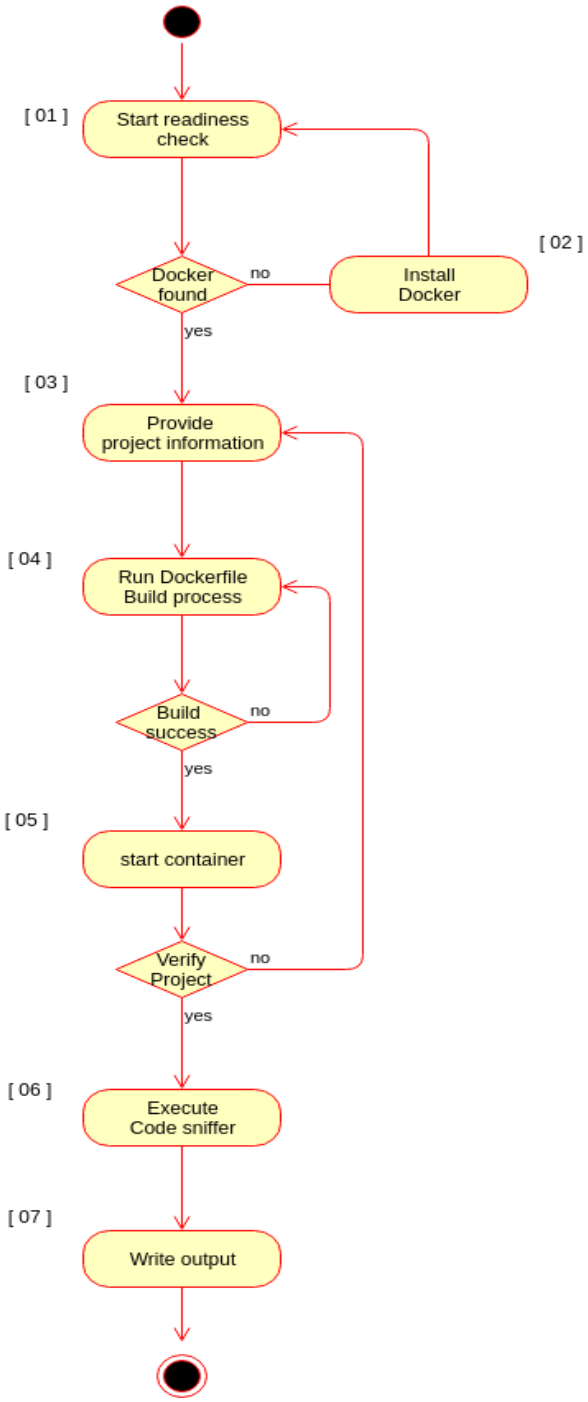

Figure 21: Activity flow

3.2.1 Custom rule sets

One of the main features of the compatibility tools is allowing the users to add custom rules sets. There are coding standards such as PSR1, PSR2, PEAR which the

developer commonly uses on IDE such as phpstorm, VS code, sublime etc. This tool provides the same capability as adding multiple coding standards and custom checkers to code sniffer. Following is a sample rule set which could be used in the proposed system.

```
<?xml version="1.0"?>
<ruleset name="User defined" namespace="ProjectName\Rules">
<!-- Define the error message type  -->
<rulegroup>
<rule ref="Conditions to be checked">
  <type>error</type>
 </rule>
 <rule ref="Conditions to be checked">
  <type>warning</type>
 </rule>
<rulegroup>
<rule ref="Forbidden_Functions">
  <properties>
   <property type="array" value="delete=>unset,print=>echo" />
  </properties>
</rule>
</ruleset>
```

Figure 22: Custom rules

3.2.2 Type of notification outputs

There are errors, warnings and notice along with severity level as notification to the user. Depending on the type of notification and severity level the results will be categorized and visualized on the dashboard. Also the summary of the total verification process will be visible to the user.

3.2.3 Deployment process

If the user wishes to extract the updated files or deploy the updated content to the server, compatibility tool provide the option to generate a deployable image along with updated PHP version. It will also provide the user, difference between the initial container and the updated container files. This is helpful to identify which files were updated and what was updated.

CHAPTER 04: IMPLEMENTATION

This section describes the implementation aspects of the compatibility tool with programming language releases. This is to provide proof of concept provided on the chapter 03: Methodology. In relation to the previous chapter, this section provides the scope of the implementation, dependencies, implementation technology and outcomes of the compatibility tool.

4.1 Scope of implementation

In PHP programming language currently there are 3 supported versions (7.1, 7.2. and 7.3). Still there are many projects running on PHP version 5.4, 5.5, 5.6 and 7.0 versions, those needs to update to be compatible with latest supported versions. Upgrading from 5.4 to 5.6 or 5.6 to 7.0 are considered major changes, since there are significant amount of changes done from PHP. [9]

The compatibility checker provides an easy to use tool where users will be able to select the wanted PHP version and execute the code sniffer to find out the errors and warning on the project. Following are the deliverables of the developed tool.

- Identify deprecated methods and syntax errors on the PHP project compared to the selected version.
- Listing of available PHP versions that are stable and supported from the vendore.
- Compatibility checker is developed using electron, a JS framework which is used to create native applications. It supports multi platforms such as windows, apple and linux based operating systems.
- Simple user friendly tool which can be used without any prior experience.
- Provide user friendly graphs and tables in the report.
- Multiple re-run of the code sniffer is possible and the reports are updated along with the changes.
- Once the code sniffer completes the execution, the users are able to view the results on the dashboard. This includes the graphs and tables.
- On the results output, it shows the error severity level file line number and option to edit.

4.2 Compatibility checker

The compatibility checker is a JS based application which uses node modules to communicate with docker API. Docker is used to create images along with required environment variables and start a container to run the compatibility checker. PHP code sniffers are used to check the project code and generate an output in JSON format. Once the iteration / verification through project code is completed, by using the results json file the reports on the dashboard will be populated. The user will be able to update the related php / phtml files shown on the results. The users have the option to create a deployable docker image along with updated code and desired PHP version.

4.2.1 Application implementation

This tool uses the electron framework to build the system as a native application. Electron was developed by GitHub as an open source library for creating cross-platform desktop applications. It uses HTML, CSS, and JS for creating application views. This library achieves this by using Chromium and Node.js combining into single runtime application which can be built and packaged multiple platforms such as Mac, Linux and Windows. [17] There are many node modules used during the development of this application. React, Webpack, Dockerode, material-ui, etc. Following is a package.json file along with all the dev and prod dependencies.

```
"dependencies": {
    "@material-ui/core": "3.9.2",
    "@material-ui/icons": "3.0.2",
    "chartist": "0.10.1",
    "classnames": "2.2.6",
    "concurrently": "^4.1.0",
    "dockerode": "^2.5.8",
    "electron": "^4.1.4",
    "electron-is": "^3.0.0",
    "electron-is-dev": "^1.1.0",
    "history": "4.7.2",
    "perfect-scrollbar": "1.4.0",
    "prettier": "1.16.4",
    "prop-types": "15.7.1",
    "react": "16.8.1",
    "react-chartist": "0.13.3",
    "react-dom": "16.8.1",
```

```
    "react-google-maps": "9.4.5",
    "react-router-dom": "4.3.1",
    "react-scripts": "2.1.5",
    "react-swipeable-views": "0.13.1",
    "wait-on": "^3.2.0"
  },
  "devDependencies": {
    "@babel/core": "^7.4.2",
    "@babel/plugin-proposal-class-properties": "^7.4.0",
    "@babel/preset-react": "^7.0.0",
    "babel-loader": "^8.0.4",
    "babili-webpack-plugin": "^0.1.2",
    "css-loader": "^2.0.2",
    "electron-packager": "^13.0.1",
    "file-loader": "^3.0.1",
    "html-webpack-plugin": "^3.2.0",
    "mini-css-extract-plugin": "^0.5.0",
    "postcss-cssnext": "^3.1.0",
    "postcss-import": "^12.0.1",
    "postcss-loader": "^3.0.0",
    "postcss-nested": "^4.1.1",
    "postcss-pxtorem": "^4.0.1",
    "style-loader": "^0.23.1",
    "url-loader": "^1.1.2",
    "webpack": "^4.28.2",
    "webpack-cli": "^3.1.2",
    "webpack-dev-server": "^3.1.14"
  }
}
```

Figure 23: package.json

Fully completed package.json file will be available in the appendix. Code compatibility application uses the port “localhost:8080”. This is commonly used by other services such as nginx, apache. If specific port is already taken the users can easily change the main.js file and added unique port. Electron uses the createWindow function to startup the chromium browser.

```
function createWindow() {
  // Create the browser window.
  mainWindow = new BrowserWindow({
    width: 1024,
    height: 768,
    show: false
  })

  // and load the index.html of the app.
  let indexPath

  if (dev && process.argv.indexOf('--noDevServer') === -1) {
    indexPath = url.format({
```

```
      protocol: 'http:',
      host: 'localhost:8080',
      pathname: 'index.html',
      slashes: true
    })
  } else {
    indexPath = url.format({
      protocol: 'file:',
      pathname: path.join(__dirname, 'dist', 'index.html'),
      slashes: true
    })
  }
```

Figure 24: Electron init code

Figure: 25 illustrate the initial screen the users would see. The users will be given the option to start the readiness check. This will check the system for docker installation. Docker is mandatory software for the code compatibility tool.

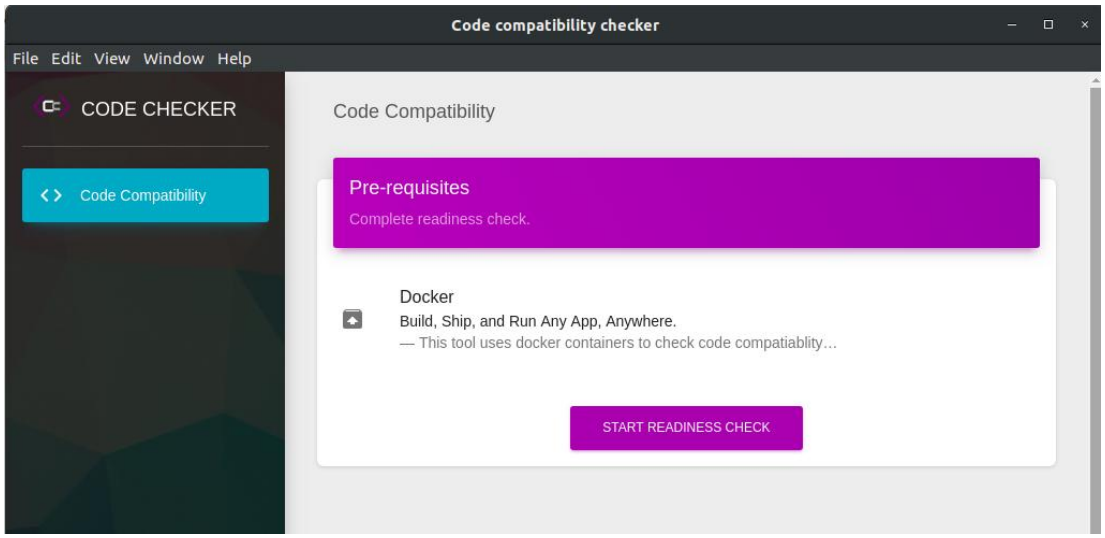

Figure 25: Readiness check

List of system calls are carried out to check if the docker is already installed. The tool will look for the docker installation and the current version to verify the compatibility. Following code sample triggers the system calls, base on the output user will be able to proceed to next step. If the docker installation was not found the user will provided with an option to install docker on the PC.

```
let cmd = (is.windows()) ? 'docker-bash' : 'docker-shell';
    let commands = require('../../scripts/'+cmd);
```

```
for (var i = 0; i < Object.keys(commands).length; i++) {
  var commandKey = Object.keys(commands)[i];
  var commandVal = Object.values(commands)[i];

  var parts = commandVal.split(/\s+/g);
  var spawn = spawnSync(parts[0], [parts.slice(1), {
    shell: true
  }]);
```

Figure 26: Docker checker

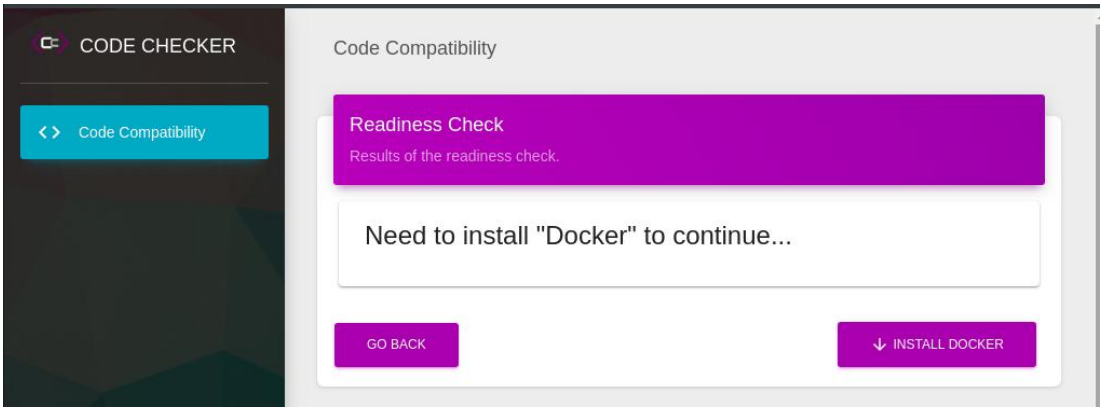

Figure 27: Docker Install option

Docker installation will trigger an installation script on the user's machine, depending on the OS platform installation process would vary. If it's a linux base system application will trigger a installation script which will install docker, for windows and apple OS users the user needs to approve the installation and setup docker. Figure 4.3 illustrates the screen output of successful readiness check.

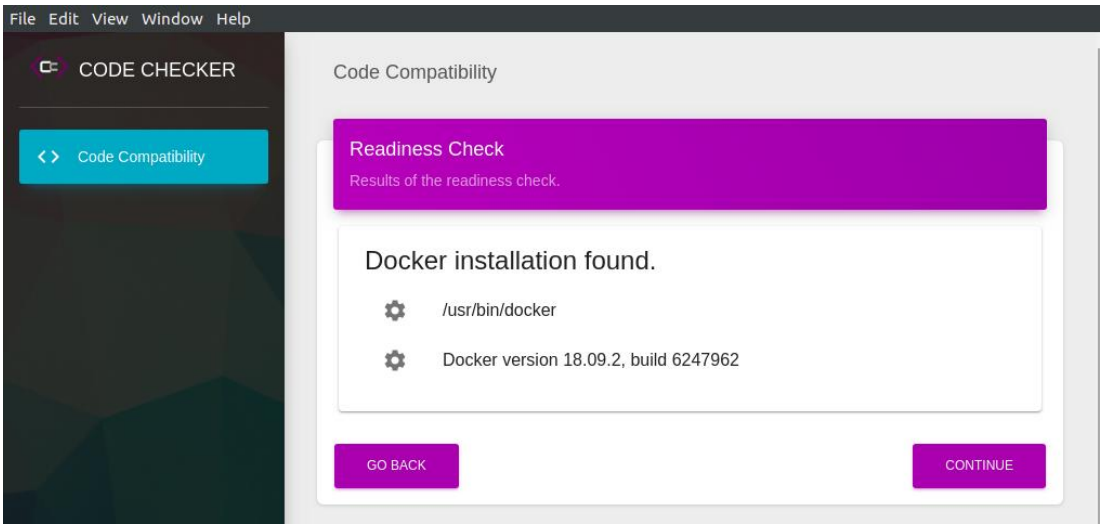

Figure 28: Docker information

4.2.1.1 Dockerode

Dockerode is used to communicate with docker. In the application it uses Docker API to create a docker image from the provided Dockerfile. [18] Dockerode proved the following features which are used in this application.

- **streams** - stream output are used to log the communication and execution of process from application to docker.
- **stream demux** - Supports optional demultiplexing.
- **run** – this library provides seamless run commands inside the container.
- **interfaces** - Features callback and promise based interfaces.

Following code sample is from the image build up process which uses the user inputs added from the docker build form. The user needs to provide.

- PHP version to check
- Project name - this will be used to create the image tag.
- Project path - This will be mounted to container /var/www/html/public

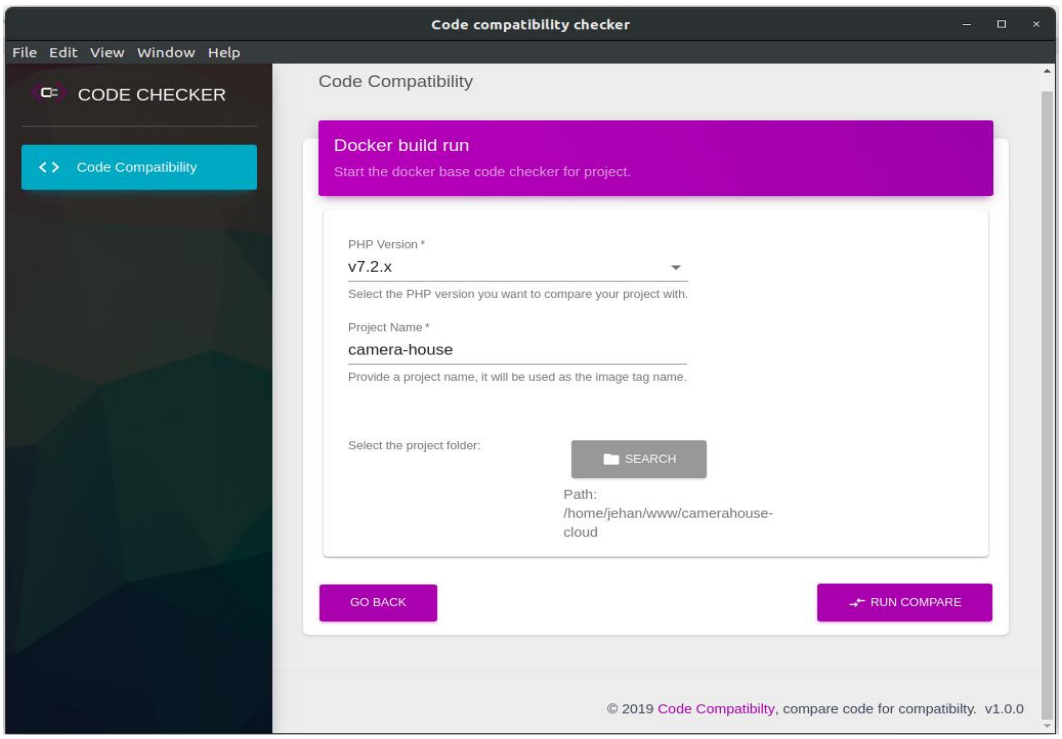

Figure 29: Docker project info

```
componentDidMount() {
    //stats
    const { activeStep, stopReRun, projectName, phpVersion } =
this.state;

    if(activeStep === 0 && stopReRun === false) {

      if (fs.existsSync(dockerBuildLog)) {
        var writeStream = fs.createWriteStream(dockerBuildLog, {
flags : 'w' });
      }
      //Docker build proces
      docker.buildImage('./Dockerfile.tar.xz', {
        t: projectName,
        rm: true,
        buildargs: {
          "buildtime_version":phpVersion
        },
      }, function( err, stream) {
        if (err)
          return;

        stream.on( 'error', function( error ) {
          this.appendOutput('stderr: <'+error+'>' );
        }.bind(this));

        stream.pipe(writeStream);

        stream.on( 'data', function (dataSet) {
          this.appendOutput(dataSet);
        }.bind(this));

        stream.on( 'end', function() {
          this.done();
        }.bind(this));
      }.bind(this));
    }
  }
```

Figure 30: Docker image create code sample

The above code sample uses the “Dockerfile.tar.xz” to build the docker image. The “Dockerfile.tar.xz” consisites for the following files:

- Dockerfile, consists of the environment set up and configuration related the docker container.
- Nginx configuration file to setup the default server on the docker container. Default file will contain the listening port 80, fastcgi_pass socket configuration and project path. This is added to help out the developers to view the site after the upgrade.
- Supervisord.conf is added to execute multiple services on the docker container. Starting and ending points are defined as ENTRYPOINT and

CMD in the dockerfile. It is general recommendation would be to separate areas running one service per container. That service can create multiple running processes such as apache, nginx, PHP etc. [19]

- Composer.json file is used to set up the code sniffer and PHPCompatibility along with application dependencies.
- Code_check.xml, this includes the rule sets to be monitored during the code scanning process.

The outputs of the process are written to the logs and visible to the user through the application. Once the image is successfully built, it will trigger another function to create a container and start up the process.

Following is an output taken from the terminal running the command “docker ps”. It outputs the currently running docker container on the machine. It shows the container ID, image name, open ports to the host.

```
jehan@jehan-Aspire-E5-575G:~/www/webtest/codepool$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                   NAMES
17a3c1a6c289        camerahouse/app     "supervisord"       6 minutes ago       Up 5 minutes        0.0.0.0:9000->80/tcp    elated_poitras
```

Figure 31: Container terminal output

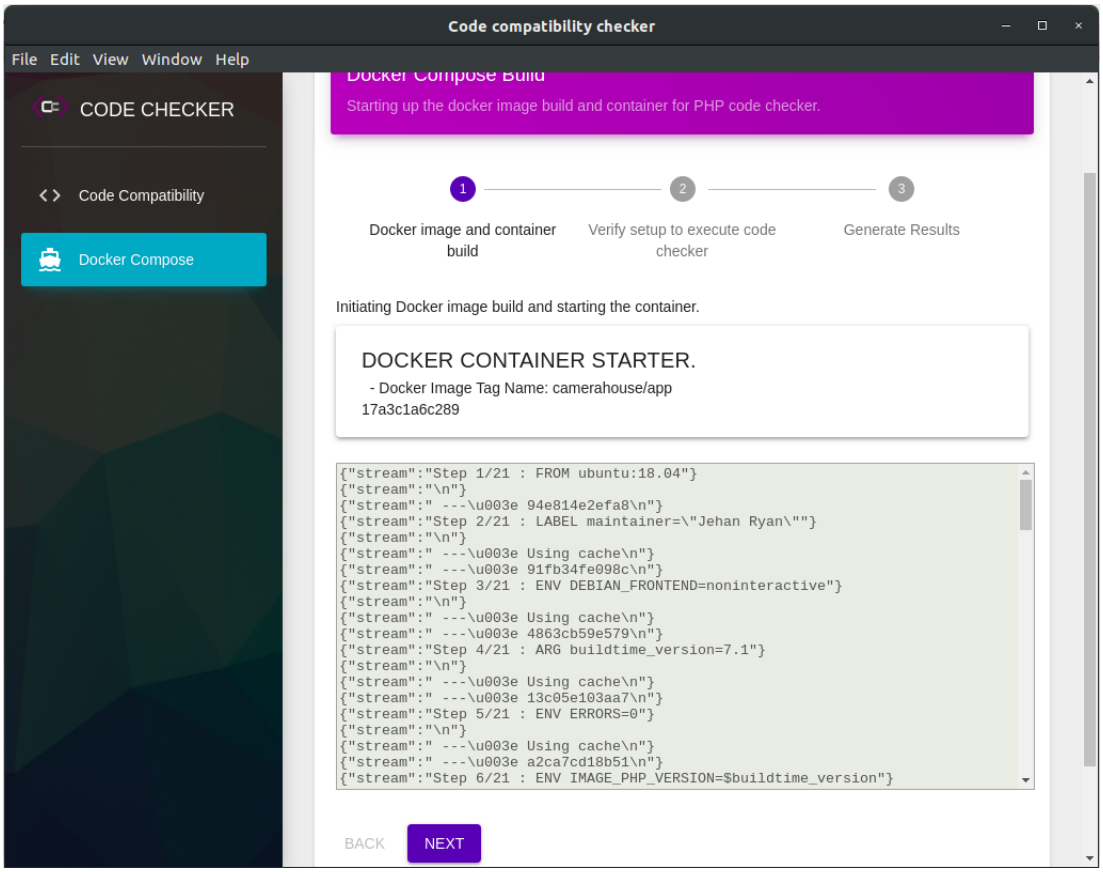

Figure 32: Docker starting view

The user will be able to see the responses and errors on the application. This allows user to re correct any issues during the docker build. The code checker docker process is a three step process.

1. Docker image and container build up.
2. Verify the setup for code sniffer.
3. Generation of results based on the code sniffer output.

All the communication with docker is carried out using dockerode node module. In the verification view the user will be able to verify the setup prior to results generation.

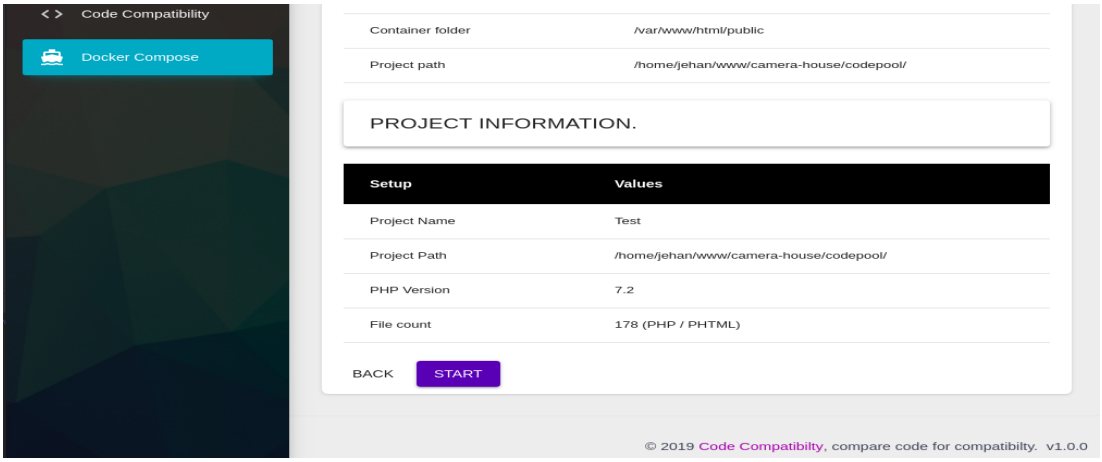

Figure 33: Docker and Project information

The application uses two dockerode functions to generate the above output. “Container.inspect()” and “container.exec()”. Container inspect is used to get the running container information. Container exec lets the user run commands on the docker container. The docker information is fetched using the inspect function and the total php/phtml file count is captured from executing the exec function. After verifying the information the user can start the code sniffer process.

4.2.1.2 Docker

Compatibility checker tool uses “ubuntu 18.04” docker instance, along with PHP, nginx, composer, git and required PHP extensions. The application uses a Dockerfile to setup these installations. The “dockerfile” along with other setup scripts are compressed and added to application directory to be picked up during the image build. PHP version is passed on as a variable to the docker file providing flexibility.

```
docker.buildImage('./Dockerfile.tar.xz', {
        t: projectName,
        rm: true,
        buildargs: {
          "buildtime_version":phpVersion //PHP version taken from
the form
        },
      }, function(err, stream) {
        if (err) return;
```

```
        stream.on('error', function(err) {
          this.appendOutput('stderr: <'+err+'>' );
        }.bind(this));
```

Figure 34: Docker image build

Following is a code sample taken from the Dockerfile used for the application. It uses the environment variable passed as ${IMAGE_PHP_VERSION} to set up the required PHP version for the docker container. The contents of the dockerfile will be available on the appendix.

```
RUN get install -y
    software-properties-common \
    && LANG=C.UTF-8 \
    && add-apt-repository ppa:ondrej/php

RUN apt-get update apt-get install -y \
       curl zip unzip git supervisor sqlite3 \
       nginx \
       php${IMAGE_PHP_VERSION}-fpm \
       php${IMAGE_PHP_VERSION}-cli \
       php${IMAGE_PHP_VERSION}-pgsql \
       php${IMAGE_PHP_VERSION}-sqlite3 \
       php${IMAGE_PHP_VERSION}-gd \
       php${IMAGE_PHP_VERSION}-curl \
       php${IMAGE_PHP_VERSION}-memcached \
       php${IMAGE_PHP_VERSION}-imap \
       php${IMAGE_PHP_VERSION}-mysql \
       php${IMAGE_PHP_VERSION}-mbstring \
       php${IMAGE_PHP_VERSION}-xml \
       php${IMAGE_PHP_VERSION}-zip \
       php${IMAGE_PHP_VERSION}-bcmath \
       php${IMAGE_PHP_VERSION}-soap \
       php${IMAGE_PHP_VERSION}-intl \
       php${IMAGE_PHP_VERSION}-readline \
       php${IMAGE_PHP_VERSION}-xdebug \
       php-msgpack php-igbinary \
    && mkdir /run/php
```

Figure 35: Dockerfile contents

Dockerfile uses the supervisord.conf to keep the nginx and php processes running during the container startup. This is essential since the user have no access to starting up these services from the application. In order to run the specified PHP version, it is passed as a variable to the supervisord.conf file. Following code sample is taken from the supervisord.conf files used on the application. [25]

```
[supervisord]
nodaemon=true

[program:php-fpm]
command = php-fpm%(ENV_IMAGE_PHP_VERSION)s
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0

[program:nginx]
command = nginx
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

Figure 36: supervisord.conf file

4.2.1.3 PHP Code sniffer / Rule sets

PHP code sniffer is used to check the project code with selected PHP version. Code sniffer is mainly used on IDS such as Phpstorm, VS code, Sublime etc. compatibility checker application will be using the terminal based execution. Code sniffer is mainly used to keep a coding standard a cross developers. PEAR, PSR2, PSR12, PSR1 are used to keep track of standards. Compatibility checker application focus more on the syntax errors and deprecated errors which could be more critical. This application uses an extension called “PHPCompatibility” with code sniffer, which identify the deprecation errors and adds to the results output.

“PHPCompatibility” and Code sniffer both are added to container using composer install. This will be picked from the dockerfiles and added to the container for installation and execution. Users can easily modify the “composer.json” file and add more extensions to the code sniffer tool. Following is the code sample taken from the composer.json file. It triggers the installation of “squizlabs/php_codesniffer” and “phpcompatibility/php-compatibility”. The users can verify this by running the following code on the container.

```
{
    "name": "phpcompatibility/code-checker",
    "description": "code checker required php_codesniffer
extensions",
    "version": "1.0.0",
      "authors" : [
        {
            "name": "Jehan Benjamin",
            "email": "ryan.jehan@gmail.com",
```

```
            "role": "Developer"
        }
    ],
      "license": "MIT",
    "require": {
        "squizlabs/php_codesniffer": "*",
            "phpcompatibility/php-compatibility": "*"
    },
    "prefer-stable" : true,
}
```

Figure 37: Composer package.json

This application uses the custom rule set to get retrieves the desired results. Syntax and compatibility checker are enabled while code comments and coding standards are ignored. Users can easily include or exclude these from the code_checker.xml file. Also the users will be able to add parameters, exclude folders, and ignore specific functions. The code_checker.xml is fully customizable, allowing users to use the setup for unique projects. The compatibility checker application focus more on the generic use of code sniffer and provided a generic set of rules.

```
<?xml version="1.0"?>
<ruleset name="Code Checker">
   <description>My rules for PHP CodeSniffer</description>
   <arg name="extensions" value="php,phtml"/>
   <!-- Run against the PHPCompatibility ruleset -->
   <rule ref="PHPCompatibility"/>
   <!-- Exclude Composer vendor directory. -->
   <exclude-pattern>*/vendor/*</exclude-pattern>
   <rule ref="Generic.PHP.Syntax"/>
</ruleset>
```

Figure 38: Custom rule set

4.2.1.3 Notification and reports

Based on the results found during the code sniffer process, the application will use the results.json file to generate notification types and reports. The user will be able to view charts based on the total number of files searched with number of error found on files, the application will differentiate the errors based on the type of error and severity level.

$$\text{syntax errors file percentage} = 100 \text{ x } \frac{\text{num of error files}}{\text{total num of files checked}}$$

$$\text{warnings file percentage} = 100 \text{ x } \frac{\text{num of warning files}}{\text{total num of files checked}}$$

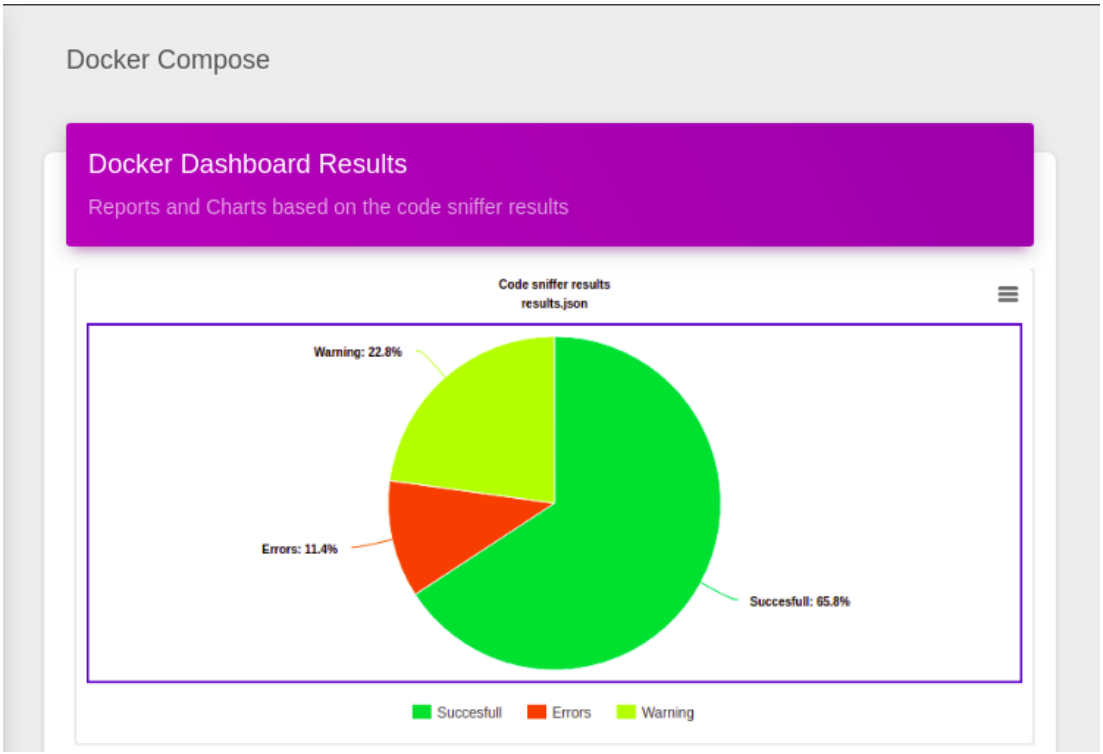

Figure 39: Pie chart

The charts will be using the percentage values to create pie / column charts on the application. The defect files would be visible on the application categorized into syntax errors, deprecated errors and warnings. Users will be able to click on the file name and access the file inside the container and fix the issues. The application allows the user to re-run the code sniffer again and validate the fixes.

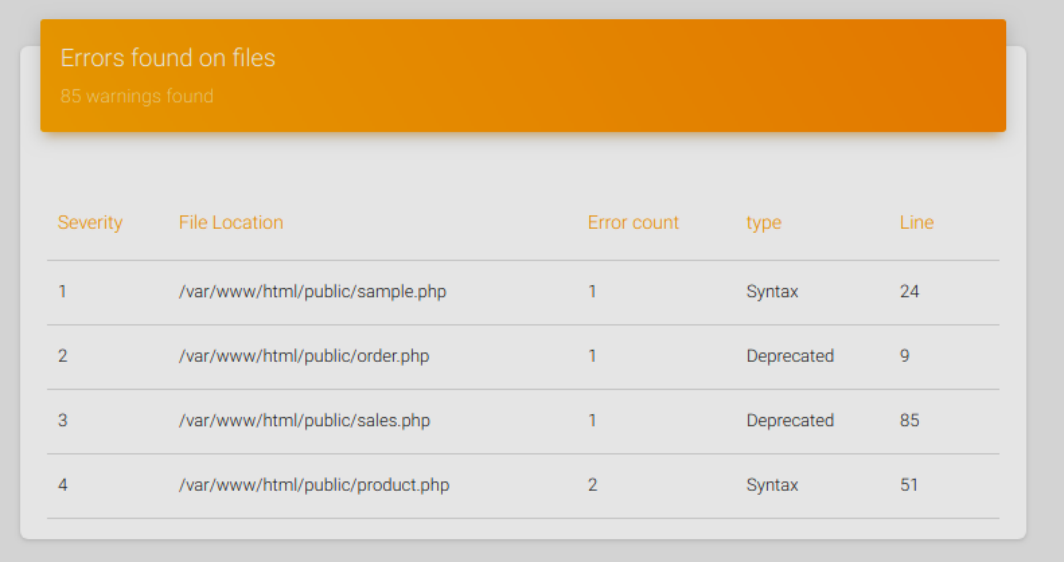

Figure 40: Error notification

The users will be able to view the list of errors on the dashboard after a successful compatibility check. The errors will be listed based on the severity level. Users can access the specified file by clicking on the file location link. The file will be open from the docker container, the developer will be able to fix the prompted issue and re-run the checker. Error section displays the syntax errors, deprecated errors and fatal errors found during the checker process.

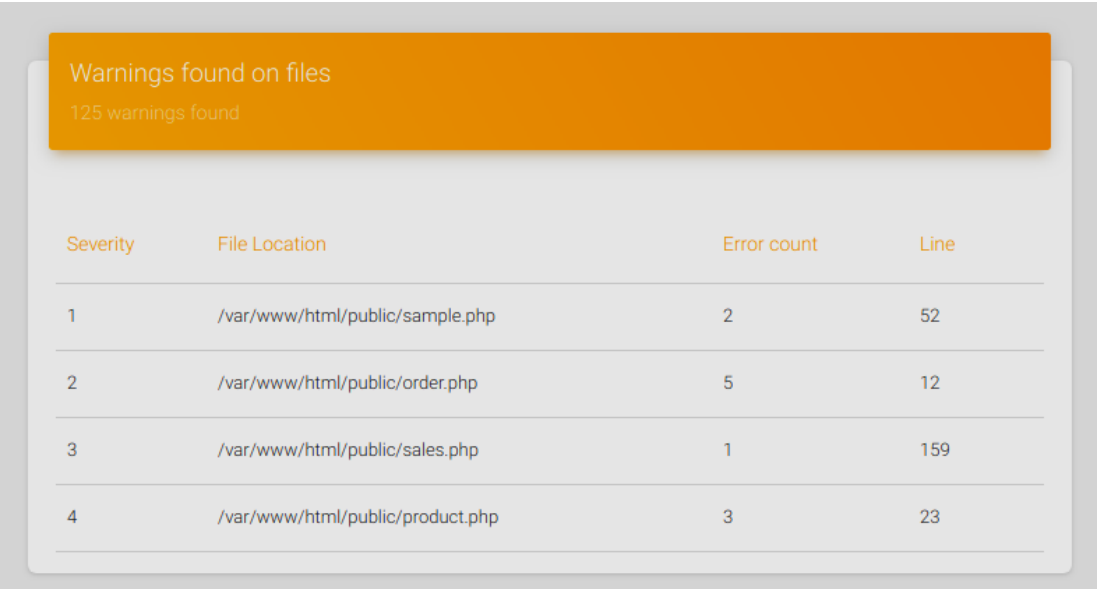

Figure 41: Warning notification

Under the warning section, the users will be able to view the list of warnings after a successful compatibility check. The warnings will be listed based on the severity

level. Users have the option to access the specified file by clicking on the file location link. The file will be open from the docker container, the developer will be able to fix the prompted warning and re-run the checker. The users can set custom rules to ignore specific types of warnings and set to ignore low level of severity errors and warnings.

4.2.1.4 Verification and deployment

The application provides the user to execute a diff command on the container. The application will generate a diff report from the results fetch from the container. This report will include all the updates done on the container.

```
$ docker diff CONTAINER ID

#sample output of the command
C /var/www/html/public/app/Sales/Block/order.php
C /var/www/html/public/app/Product/Block/inventory.php
C /var/www/html/public/app/Email/Module/order.php
C /var/www/html/public/app/Invoice/Module/shipment.php
```

Figure 42: Container diff

The application provides the option to create a new image from a container's changes. The new image with the changes can be deployed. This option helps the developers to work collaboratively. The results.json file will be available, along with project file updated list.

AWS Deployment

Amazon ECS is used to run Docker applications in AWS environments. It uses scalable clusters for docker containers. User will be able to start a docker supported application on an Amazon ECS cluster along with load balancer and execute test runs on the sample application. [20]

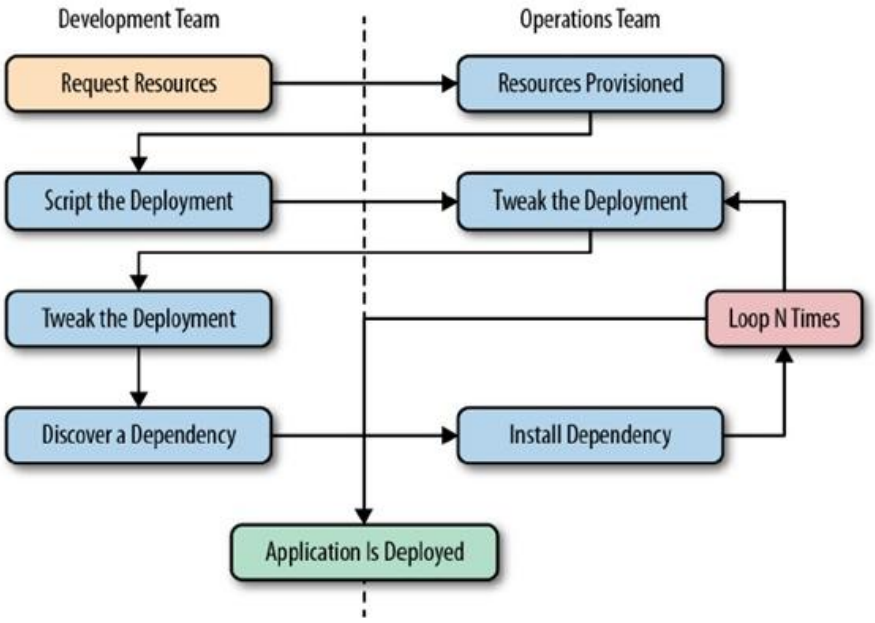

Figure 43: Docker deployment process [21]

Kubernetes Deployments

It provides a portable and extensible platform to manage containerized workloads and services. Kubernetes provide the feature to setup configurations or automation. Kubernetes is a rapidly expanding ecosystem with a widely supported range of tools. [22] Docker containers can be easily deployed to kubernetes' orchestration systems.

CHAPTER 05: EVALUATION

This section discusses the effectiveness and validation of the compatibility checker application. It verifies the deliverables based on the requirements scope. Section 5.1 focuses on the usability of the application across other tools and IDEs. Section 5.2 discusses on the projects and sample data sets and PHP code sniffer capturing errors and warnings on the results JSON. Section 5.3 presents the reporting and information validation based on the sample data set.

5.1 Usability

One of the main objectives of the research is to provide a user friendly, multi platform application. The application follows a simple step process during the readiness check and docker container build process, hiding the complex setup and configuration from the users. Users can easily keep track of the current process using the application interface. Also every docker base process calls are recorded in logs.

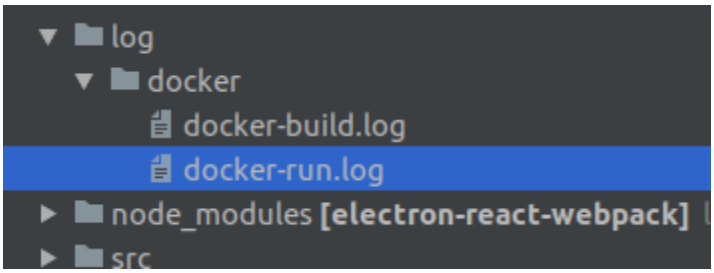

Figure 44: log files

Following are some screen captures from the code compatibility application. Provide an easy to use step by step process.

5.1.1 Readiness check

During the readiness check the application search for docker installation in the user's machine. The application uses "spawnSync" child process to trigger arbitrary command execution. The application is checking for docker installation and version for verification.

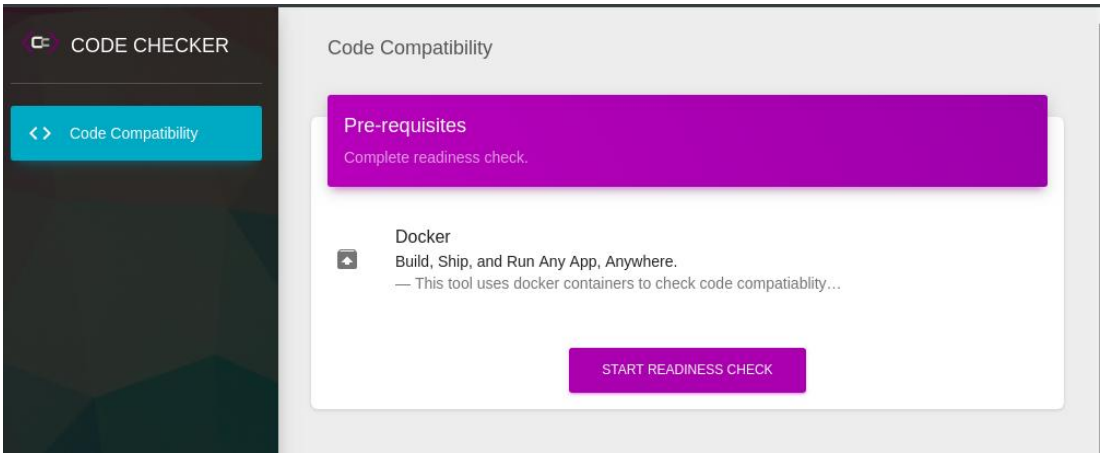

Figure 45: Application landing page

The Pre-requisites page is the initial page that the user will be able to view. It is used to inform the user on the system pre-requisites and start the initiate the code compatibility process. The will be able to view only one view, dashboard and docker compose will be available as the user progress.

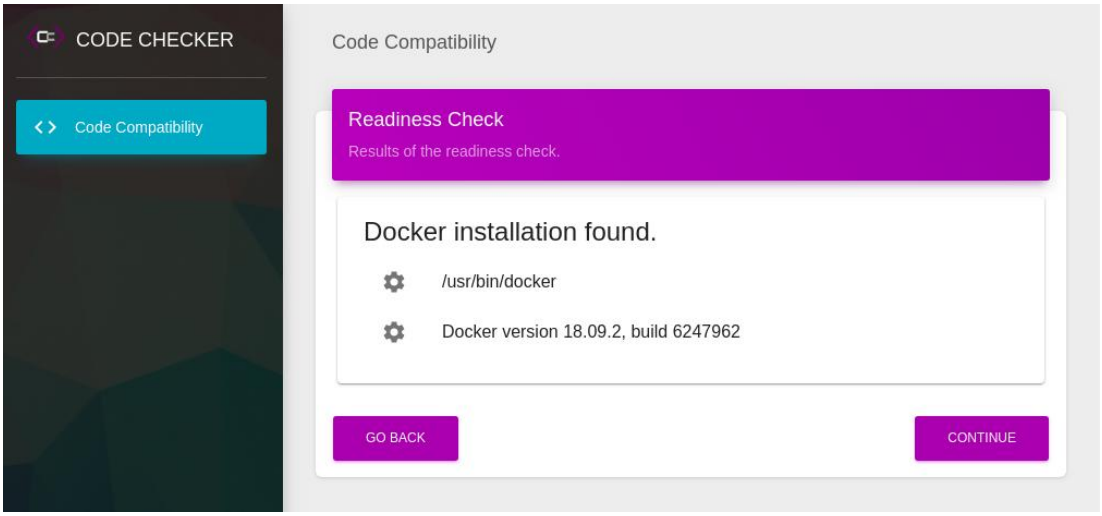

Figure 46: Docker information

Once the user starts the readiness check the next step is to check the docker installation and the version. If this step is successful user will be able to proceed to next step. Adding the project name, select the required PHP version and provide a project source path. The project name will be taken as the docker image tag name. During the readiness steps user always have to option to go back to the previous step if needed. It hides the system calls and complexity on the application from the user.

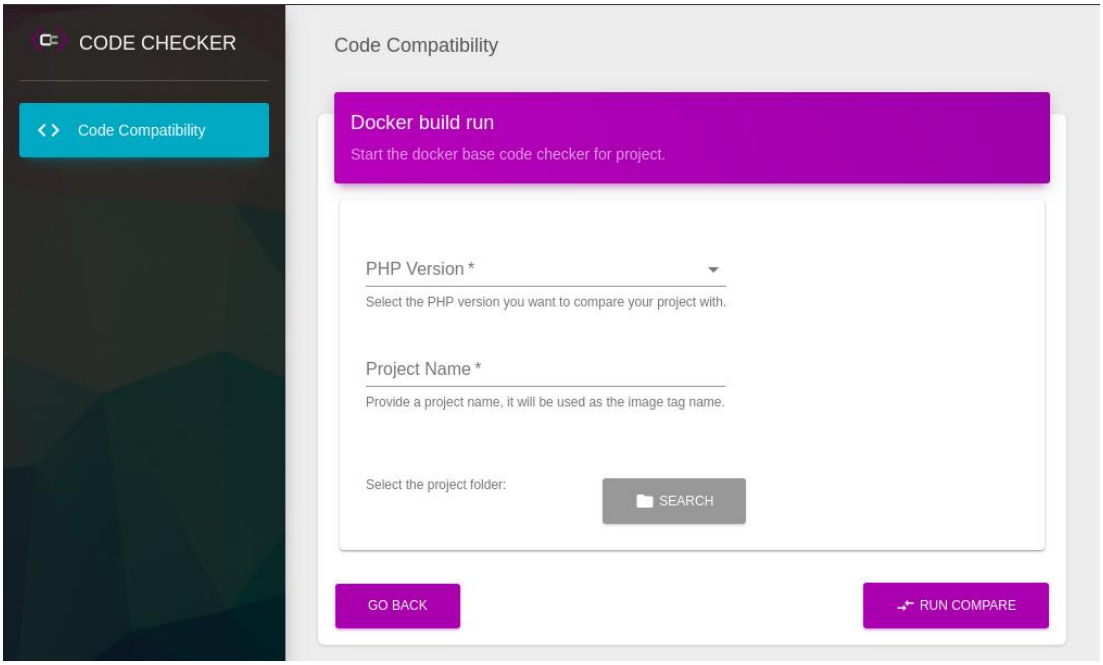

Figure 47: Project info form

5.1.2 Docker build, run process.

Code compatibility application depends on the docker features of creating and running containers. It uses a Dockerfile to setup the image and build the container based on the user input. The project name, project source location and PHP version shown on figure: 47 is captured from the form. It uses the port “9000” on the hosted machine to communicate. This can be altered by the user.

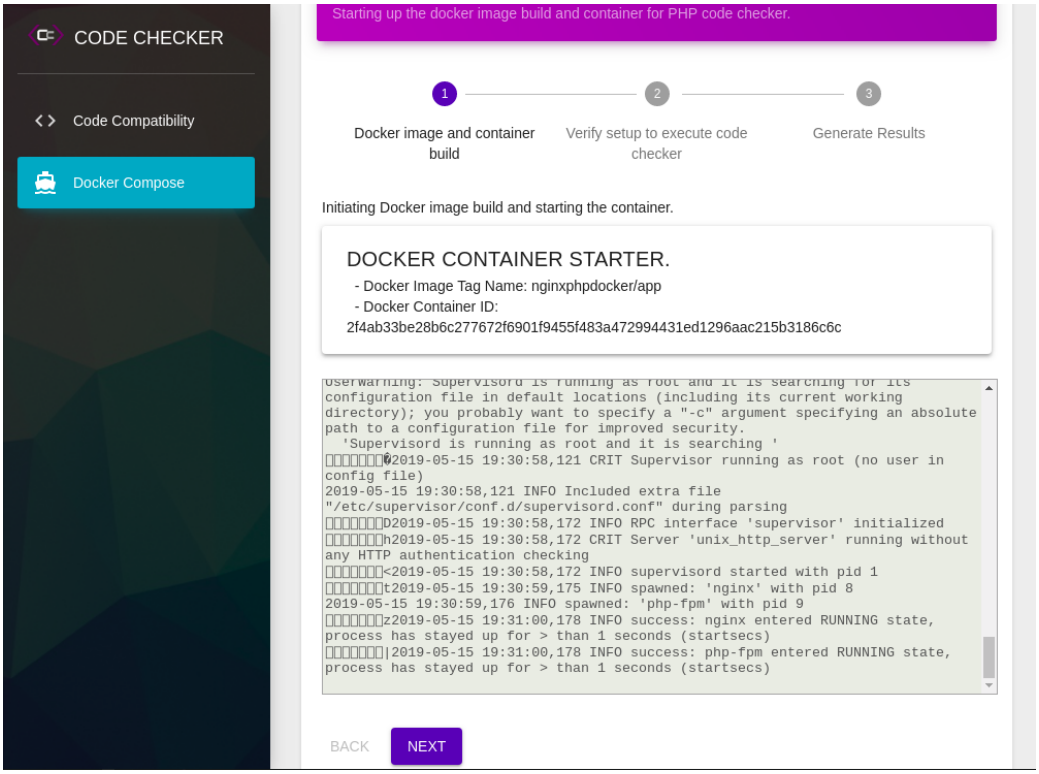

Figure 48: Docker image run and container start

On this step the docker image is built using the Dockerfile and the variable values provided by the user. Once the image is successfully created, container creation will be started. Mounting the project source with container and creating a port binding will triggered next.

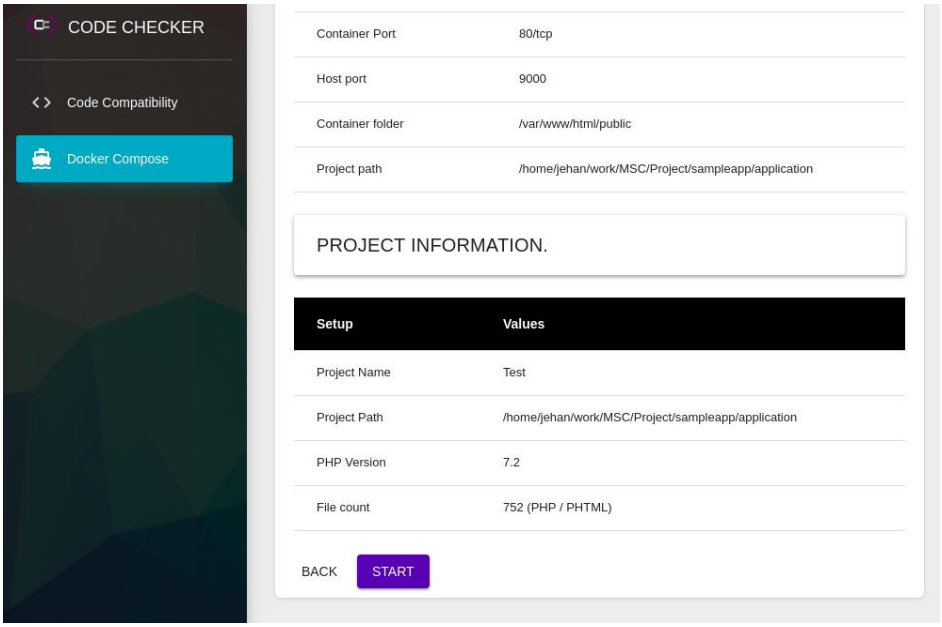

Figure 49: container and project info

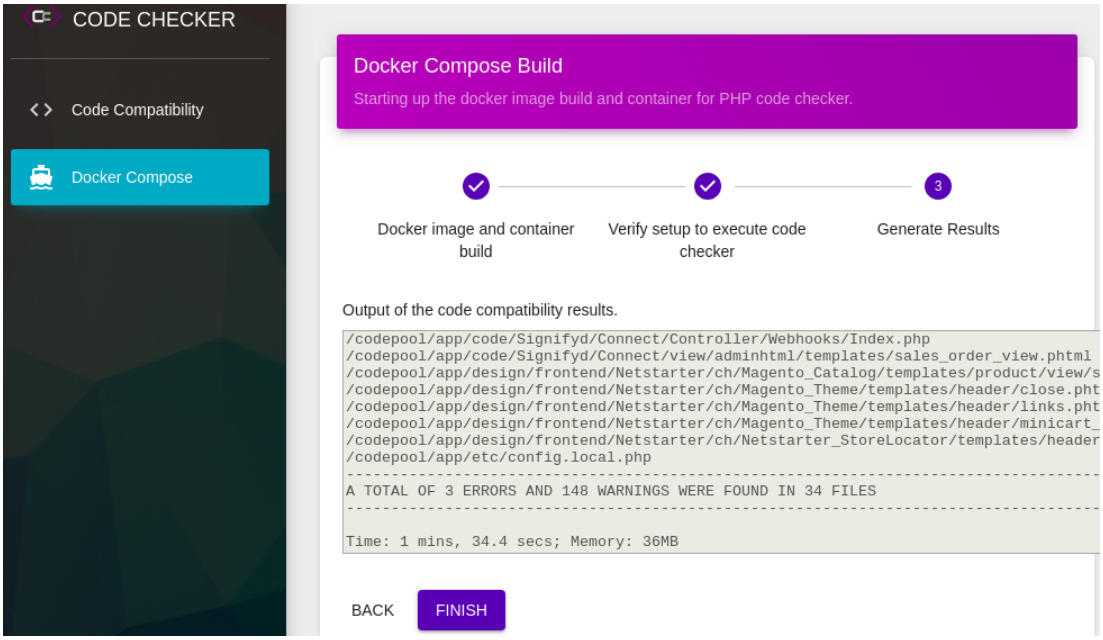

Figure 50: Docker executes and results

During this step docker executions will be triggered, PHP code sniffer checker results summary will be loaded on the application. Full results will be added to results.json file inside the docker container. JSON file contents will be used to generate the reports and notification on the dashboard.

Phpstorm is a widely used commercial IDE used by developers. Setting up codesniffer in Phpstorm requires multiple steps. It requires the user to have an idea about the IDE and how to use code sniffer plugin. Running multiple code checker from the IDE is not possible. [23] Since Phpstorm is a paid software developers may tend to look for alternatives.

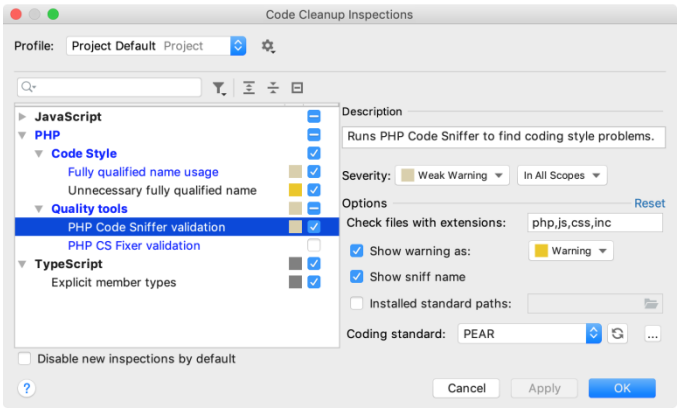

Figure 51: Phpstorm code sniffer setup

Running PHP CodeSniffer in Visual Studio Code is another option for the developers. The setup process is similar to the phpstorm. This also required the user to have previous knowledge about the IDE and code sniffer setup. [24]

The application uses docker command to setup the image and start the containers. Some developers may struggle to mount source location and open up ports to communicate with the containers. Also passing arguments and setting up variable would be challenging work for a beginner. Following are some docker terminal commands which are used from in the compatibility application. In the application its implemented using docker API calls.

```
$ docker build --build-arg buildtime_version=7.2 -t
nginxphpdocker/app:latest -f dockerTest/application/Dockerfile
dockerTest/app/

$ docker run -d --rm -e IMAGE_PHP_VERSION=7.2  -p 8080:80 -v
$(pwd)/application:/var/www/html/public nginxphpdocker/app:latest
```

Figure 52: Docker build commands

Compatibility checker application is developed using electron a JS based framework which uses the chromium browser create a native application. This tool will be built using webpack, it provides a JS based API that could be used in Node.js runtime. [25]

```
"scripts": {
 "prod": "webpack --mode production --config
webpack.build.config.js && electron --noDevServer .",
 "start": "webpack-dev-server --hot --host 0.0.0.0 --
config=./webpack.dev.config.js --mode development",
 "build": "webpack --config webpack.build.config.js --mode
production",
 "package": "npm run build",
 "postpackage": "electron-packager ./ --out=./builds"
},
```

Figure 54: webpack build scripts

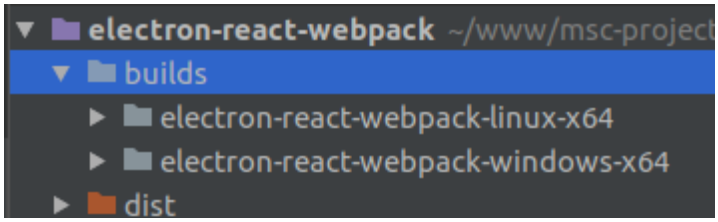

Figure 53: platform related builds

5.2 Application results evaluation

In order to conduct an unbiased evaluation of the results for the application, the results were captured from fifteen systems that are currently in production. Table 5.1 illustrates the types of PHP projects that were selected for the testing.

Table 3: Sample projects information

Project type	PHP Versions	Num of Projects
Magento 1.x	5.6	2
Magento 2.0.x	7.0	2
Magento 2.1.x	7.0 ~ 7.1	3
Orange HRM	7.1	1
CRM - Vtiger	5.6	1
Wordpress blogs 4.x	5.6 ~ 7.0	3
Wordpress blogs 5.x	7.1	1
Laravel 5.6	7.1	2

5.2.1 Code sniffer results

All the selected projects were analyzed using the compatibility checker application using PHP version 7.2. This version was selected due to its stability, support and relatedness. Code sniffers with custom rules were executed from terminal and through the application to compare the output and execution time. During the execution of code sniffer, the author noticed the execution time and results are same.

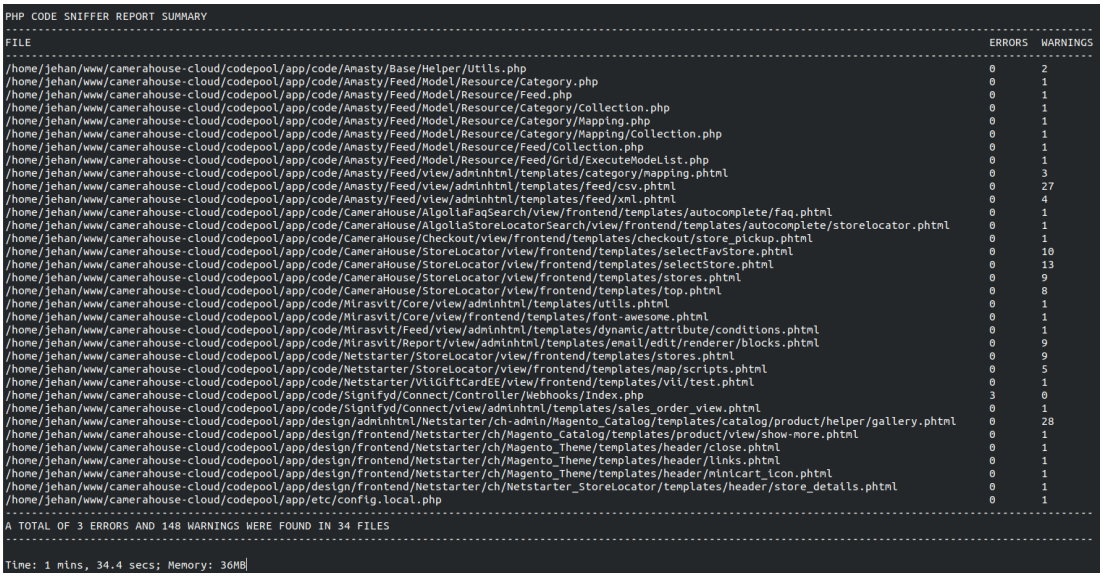

Figure 55: Terminal summary output

Above is the terminal output taken from summary generation. It has taken 1mins, 53.57secs and Memory: 36MB to generate the summary. Same figures were identified from the application output during the summary report generation.

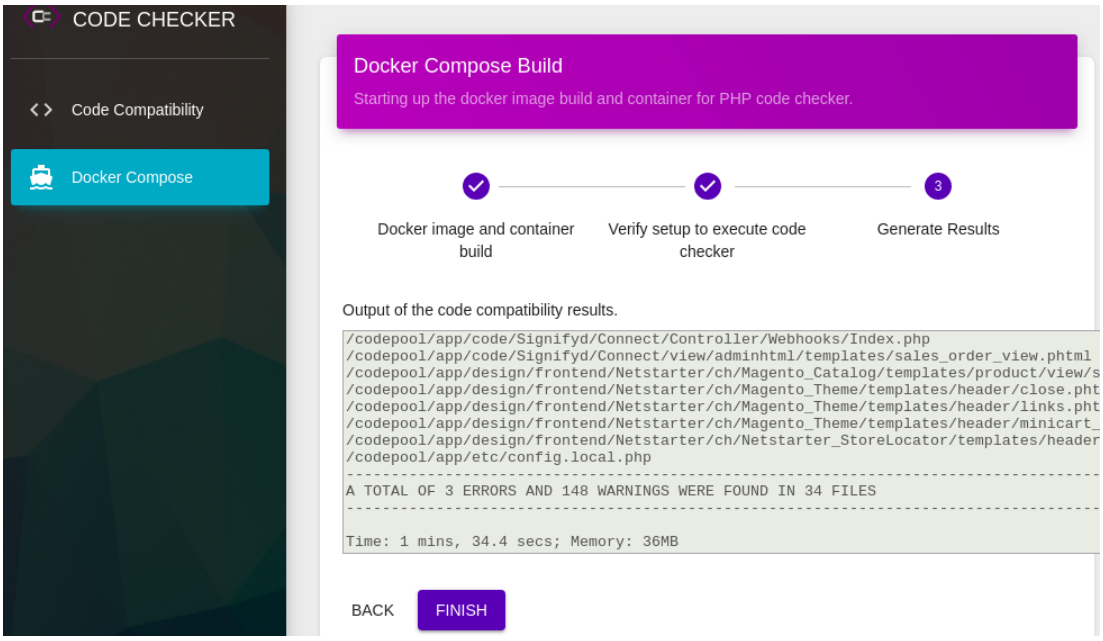

Figure 56: report summary on application

Following table is a comparison of the code sniffer summary report generation. The application uses both summary and full JSON based report to generate reports for the user.

Table 4: Compatibility results summary

PHP versions compared	Time to complete	Memory	Errors	Warnings
5.6 -> 7.2				
Magento 1.x	19mins, 12.13secs	94.01MB	305	310
Magento 1.x	18mins, 35.54secs	94MB	307	250
CRM - Vtiger	21mins, 15.23secs	58MB	159	296
Wordpress	15mins, 04.30secs	73.59MB	72	498
7.0 -> 7.2				
Magento 2.0.x	3mins, 44.17secs	41MB	7	171
Magento 2.0.x	3mins, 41.89secs	39MB	4	149
Magento 2.1.x	1mins, 53.57secs	36MB	3	148
Wordpress blogs 4.x	52.32secs	12MB	2	123
Wordpress blogs 4.x	42.28secs	11.23MB	2	112
7.1 -> 7.2				
Laravel 5.6	43.54secs	9.5 MB	1	92
Laravel 5.6	21.01secs	4 MB	0	25
Wordpress blogs 5.x	32.24secs	4.2 MB	0	35
Orange HRM	54.25secs	29 MB	2	185
Magento 2.1.x	33.54secs	3.2 MB	0	21
Magento 2.1.x	22.29secs	2.5 MB	0	11

The projects tend to differ from the platforms, code architecture and the functionality. Magento is one of the leading ecommerce platform providers, one of the largely used PHP base systems; Wordpress is one of the popular content

management systems. Laravel is a free, open-source PHP web framework used on many projects. Orange HRM and Vtiger CRM were selected to cater the need of having unique type of systems to conduct the testing.

5.2.2 Evaluation of the test results

Compatibility results from 5.6 to 7.2

During the evaluation total number of four projects were checked, these systems were built using PHP 5.6. Using the compatibility application the code was check for compatibility with 7.2. It was noticed that execution of time on these reports were high compared to the other versions.

Table 5: Compatibility results summary for 5.6

5.6 -> 7.2	Execution time.	Report size	Error	Warning
Magento 1.x	19mins, 12.13secs	94.01MB	305	310
Magento 1.x	18mins, 35.54secs	94MB	307	250
CRM - Vtiger	21mins, 15.23secs	58MB	159	296
Wordpress	15mins, 04.30secs	73.59MB	72	498
Avg.	**18mins, 38secs**	**79.9 MB**	**210.75**	**338.5**

From these results it was noticed that there are higher number of errors and warnings were generated. During these inspections the following issues were commonly identified.

- Static calls to non-static methods
- password_hash()
- capture_session_meta
- Use of LDAP deprecated methods

Based on the findings, we can assume that the next project with similar build would take around 18mins to generate a result. Depending on the size of the project expected error count would be around 210, warning count would be around 338.

Compatibility results from 7.0 to 7.2

In order to evaluate the compatibility of the code in the selected project in we monitored the execution time, file size, error and warning count. Below tables show the results along with the averages.

Table 6: Compatibility results summary for 7.0

7.0 -> 7.2	Execution time.	Report size	Error	Warning
Magento 2.0.x	3mins, 44.17secs	41MB	7	171
Magento 2.0.x	3mins, 41.89secs	39MB	4	149
Magento 2.1.x	1mins, 53.57secs	36MB	3	148
Wordpress blogs 4.x	52.32secs	12MB	2	123
Wordpress blogs 4.x	42.28secs	11.23MB	2	112
Avg.	**2mins, 18secs**	**27.85MB**	**3.6**	**140.6**

By analyzing the error reports, the following deprecated errors were found used in the project code:

- each() function
- create_function() function
- __autoload() method
- assert() with a string argument
- Unquoted strings

There were considerable amount of warnings were found during the testing phase with an average of 140. Execution time is very less compared to the 5.6 averaging around 2mins.

Compatibility results from 7.1 to 7.2

PHP version 7.1 and 7.2 are still largely at use, most of the features are available for both 7.1 and 7.2 versions, with the introduction of 7.3 and upcoming PHP version 7.4 and version 7.1 will be outdated soon after. The compatibility testing was carried out for the selected project on 7.2 version.

Table 7: Results summary for 7.1

7.1 -> 7.2	Execution time.	Report size	Error	Warning
Laravel 5.6	43.54secs	9.5 MB	1	92
Laravel 5.6	21.01secs	4 MB	0	25
Wordpress blogs 5.x	32.24secs	4.2 MB	0	35
Orange HRM	54.25secs	29 MB	2	185
Magento 2.1.x	33.54secs	3.2 MB	0	21
Magento 2.1.x	22.29secs	2.5 MB	0	11
Avg.	**34secs**	**13.53MB**	**0.5**	**61.5**

Error count and execution time are low compared to the other version check results. Still there are considerable amount of warnings found on the projects.

5.3 Reporting and error information verification

During the evaluation process, it was identified that the pie chart information is misleading. Initial chart designs were to display the percentage along with the labels (successful, errors, warnings). In the evaluation process out of the fifteen projects, selected a project which had 11725 number of php and phtml files out of them only 3 files were found with errors and 148 had warnings. The selected system is a Magento 2.1.3 based project.

$$\text{syntax errors file percentage} = 100 \text{ x } \frac{\text{num of error files}}{\text{total num of files checked}}$$

When the figures are applied to the equation, it gives an error percentage of 0.025586354 and warning percentage of 1.262260128. The original pie chart design shows only up to one decimal place and the value is rounded up according to that. The pie chart show as errors as 0.0% which is misleading.

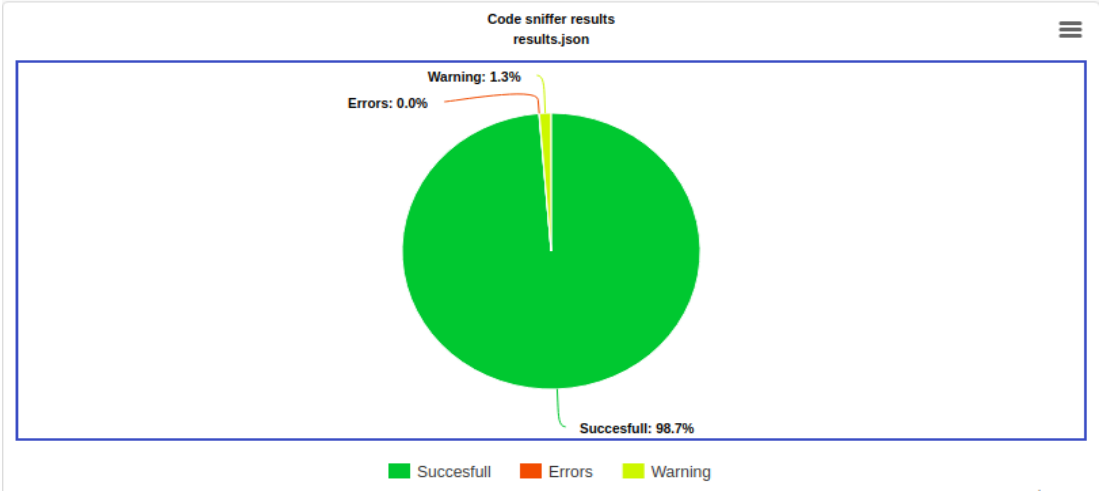

Figure 57: misleading information

The pie chart view is updated to show the label, error count and the percentage. Provide a more accurate output to the user.

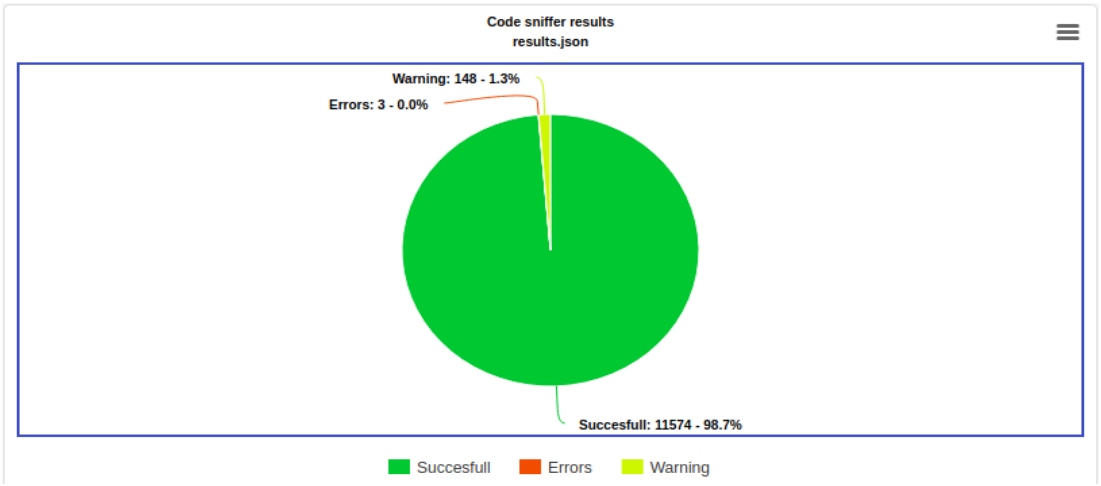

Figure 58: Updated pie chart view

CHAPTER 06: CONCLUSION AND FUTURE WORK

This chapter present the conclusions of the researched areas describe on the thesis. This section re-visits the objectives on chapter 1 and how the implementation delivers the expected outcome. This section also includes the achievements, problems encountered and future work.

Summary

The phase “Don't fix it if it ain't broken” should not be applied to systems anymore. The programming languages will keep on evolving and keep on releasing updated versions. In order to get the updated features and fixes, it is advised to stay updated with the latest supported version. Checking version changes and running code testers would be time consuming. Compatibility checker is an easy to use application which provides docker based solution that verifies the code with the selected PHP version.

The objectives identified on the chapter 01 are fulfilled in the implementation deliverables. Users are now able to check for issues during upgrade from one programming version to another. The compatibility tool identifies the deprecated methods, syntax errors and warnings on the project. Based results users will be able to make decisions, edit the specific files and make the code compatible with the selected version.

The application consists of integrations between electron, docker, PHP code sniffer, composer and node JS plugins. These components were mapped to create a working solution discussed on the implementation section as a proof of concept. Users have the ability to execute the application and check for errors without any previous knowledge of the application. This system is not totally focused on developers. Due to its usability QA engineers, project managers and even clients would be able to run the application and check for issues.

The compatibility application could be used in multi platform environments. This is one of the key areas since there are many developers working with different type of platforms. This application uses docker containers which could be used by

developers to easily get the updated code and make a deployable image. Docker containers are supported on AWS and Kubernetes which are widely used server infrastructure providers.

One of the areas focused during the evaluation process was the execution results and time spent on generating the results. The testing processes were carried out for fifteen projects that are currently in production. Depending on the PHP version and framework there projects were categorized and monitored. During the evaluation it was noticed that upgrading from PHP version 5.6 to 7.2 directly is less feasible, based on the results. The average numbers of errors were 210 along with warning average of 338. Average execution time comes around 18 mins, which affects the usability of the system, since user will be seeing a loading screen for 18 mins.

Also one additional option for developers would be to plug in a database and run the PHP project from the host machine. This is possible due to docker container is consists of nginx and PHP extensions and composer.

6.1 Problems encountered

The application is consists of many integrations, code sniffer, dockerode node js module, react, streams, etc. building a common communication between these integration were time consuming. Each integration has its own limitations customizing and extending the functionality needed a learning curve. PHP Codesniffer alone does not capture system errors and deprecated errors, needed to add PHPCompatibility and customize the rule set to find syntax errors.

During the evaluation it was noticed that it takes an average around 18mins to complete the compatibility check on 7.2 for a projects developed using 5.6 version, this may vary based on the number of files and project size.

6.2 Future work

Currently the application mainly focuses on the deprecation errors and syntax errors. There are many other types of errors which need to be identified during the codesniffer process. Functional and performance aspects of the application were not

evaluated during this research. Large amounts of files and JSON result output would slow down the performance and usability. In order to cater this in future we can implement the progress Information option provided by the code sniffer or break files into smaller chunks and run the compatibility checker. Building up an image container takes considerable amount of time, due to the number of installation and updates. Optimizations of this process were not considered during the research topic. This application mainly focuses on PHP programming language but with the use of docker containers we are able add any type of programming languages with its dependencies to be verified.

REFERENCES

[1] (2015) PLE 2015 Programming Language Evolution. [Online]. https://2015.ecoop.org/track/PLE-2015-papers

[2] P. JB King. (1999, Augest) Heriot-Watt University. [Online]. https://2015.ecoop.org/track/PLE-2015-papers

[3] A. Shaleynikov. (2017, October) Hackernoon. [Online]. https://hackernoon.com/top-10-programming-languages-in-2017-2f22e918fbfd

[4] J. Mueller. (2015, May) NewRelic Blog. [Online]. https://blog.newrelic.com/2015/05/05/programming-language-version/

[5] T. Brunner, N. Pataki, and Z. Porkolab, "Tool for Detecting Standardwise Differences in C++ Legacy Code," in *IEEE 13th International Scientific Conference on Informatics*, Slovakia, 2015, pp. 57-62.

[6] (2015, March) Compatibility Guide for JDK 8. [Online]. http://www.oracle.com/technetwork/java/javase/8-compatibility-guide-2156366.html

[7] J. Neal. (2017) Polyfill.io. [Online]. https://polyfill.io/v2/docs/

[8] P. Klau. (2016, November) CSS-Tricks. [Online]. https://css-tricks.com/polyfill-javascript-need/

[9] (2019) PHP Official site. [Online]. https://www.php.net/manual/en/migration70.php

[10] L. C. Kenneth and L. A. Kenneth, *Programming Languages: Principles and Practices*, 3rd ed., Marie lee, Ed. Boston, United States of America: Course Technology, 2012.

[11] G. Sherwood. (2016, July) PHP_CodeSnifferwiki. [Online]. https://github.com/squizlabs/PHP_CodeSniffer/wiki

[12] H. C. Gall, "LGTM - Software Sensing and Bug Smelling," in *IEEE*, Szeged, 2012, pp. 3-4.

[13] wapmorgan. (2019) phpclasses. [Online]. https://www.phpclasses.org/package/10211-PHP-Find-deprecated-functions-and-variables-in-PHP.htm

[14] K. K. Chaturvedi, V. B. Singh, and P. Singh, "Tools in Mining Software Repositories," in *13th International Conference on Computational Science and Its Applications*, Ho Chi Minh, 2013, pp. 89-98.

[15] N. M Tiwari, G. Upadhyaya, and H. Rajan, "Candoia: A Platform and Ecosystem for Mining Software," in *ACM 38th IEEE International Conference on Software Engineering Companion*, Austin, 2016, pp. 759-761.

[16] D. Smith. (2015, April) phpclasses. [Online]. https://www.phpclasses.org/package/9084-PHP-Find-deprecated-functions-and-suggest-replacements.html

[17] (2019) electronjs. [Online]. https://electronjs.org/docs/tutorial/about

[18] apocas. (2019) npm js. [Online]. https://www.npmjs.com/package/dockerode

[19] (2019) docker. [Online]. https://docs.docker.com/config/containers/multi-service_container/

[20] (2019) aws. [Online]. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

[21] Ajeetraina. (2019) collabnix. [Online]. https://collabnix.com/a-docker-deployment-workflow/

[22] (2019) kubernetes. [Online]. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[23] (2019) jetbrains. [Online]. https://www.jetbrains.com/help/phpstorm/using-php-code-sniffer.html

[24] (2019) tommcfarlin. [Online]. https://tommcfarlin.com/php-codesniffer-in-visual-studio-code/

[25] (2019) webpack. [Online]. https://webpack.js.org/api/node/

[26] (2016, June) Informationq. [Online]. https://www.informationq.com/computer-language-and-its-types/

[27] (2017) octoverse.github. [Online]. https://octoverse.github.com/2017/

[28] Zend. (2019) zend. [Online]. http://files.zend.com/help/Zend-Studio/content/semantic_analysis_preferences.htm

[29] K. Adams et al., "The HipHop Virtual Machine," in *OOPSLA*, Portland, 2014, pp. 777-790.

[30] M. G. Schneider and J. L. Gersting, *Invitation to Computer Science: C++ Version*, 4th ed.: Thomson Learning EMEA, 2006.

APPENDIX A

package.json

```
{
          "name": "electron-react-webpack",
          "version": "1.0.0",
          "description": "Code Compare Application. Coded by Jehan Ryan",
          "license": "MIT",
          "private": false,
          "repository": {
            "type": "git",
            "url": "https://github.com/jehanben/electron-react-webpack.git"
          },
          "homepage": "/",
          "bugs": {
            "url": "https://github.com/jehanben/electron-react-webpack.git/issues"
          },
          "author": {
            "name": "Jehan Ryan",
            "email": "ryan.jehan@gmail.com",
            "url": "https://github.com/jehanben"
          },
          "keywords": [
            "app",
            "boilerplate",
            "electron",
            "open",
            "open-source",
            "postcss",
            "react",
            "reactjs",
            "source",
            "webpack"
          ],
          "engines": {
            "node": ">=9.0.0",
            "npm": ">=5.0.0",
            "yarn": ">=1.0.0"
          },
          "main": "main.js",
          "scripts": {
            "prod": "webpack --mode production --config webpack.build.config.js && electron --noDevServer .",
            "start": "webpack-dev-server --hot --host 0.0.0.0 --config=./webpack.dev.config.js --mode development",
            "build": "webpack --config webpack.build.config.js --mode production",
            "package": "npm run build",
            "postpackage": "electron-packager ./ --out=./builds"
          },
          "dependencies": {
            "@material-ui/core": "3.9.2",
            "@material-ui/icons": "3.0.2",
            "chartist": "0.10.1",
            "classnames": "2.2.6",
            "concurrently": "^4.1.0",
            "dockerode": "^2.5.8",
            "electron": "^4.1.4",
            "electron-is": "^3.0.0",
            "electron-is-dev": "^1.1.0",
            "history": "4.7.2",
            "perfect-scrollbar": "1.4.0",
            "prettier": "1.16.4",
            "prop-types": "15.7.1",
            "react": "16.8.1",
            "react-chartist": "0.13.3",
            "react-dom": "16.8.1",
```

```
            "react-google-maps": "9.4.5",
            "react-router-dom": "4.3.1",
            "react-scripts": "2.1.5",
            "react-swipeable-views": "0.13.1",
            "wait-on": "^3.2.0"
          },
          "devDependencies": {
            "@babel/core": "^7.4.2",
            "@babel/plugin-proposal-class-properties": "^7.4.0",
            "@babel/preset-react": "^7.0.0",
            "babel-loader": "^8.0.4",
            "babili-webpack-plugin": "^0.1.2",
            "css-loader": "^2.0.2",
            "electron-packager": "^13.0.1",
            "file-loader": "^3.0.1",
            "html-webpack-plugin": "^3.2.0",
            "mini-css-extract-plugin": "^0.5.0",
            "postcss-cssnext": "^3.1.0",
            "postcss-import": "^12.0.1",
            "postcss-loader": "^3.0.0",
            "postcss-nested": "^4.1.1",
            "postcss-pxtorem": "^4.0.1",
            "style-loader": "^0.23.1",
            "url-loader": "^1.1.2",
            "webpack": "^4.28.2",
            "webpack-cli": "^3.1.2",
            "webpack-dev-server": "^3.1.14"
          }
        }
```

Dockerfile

```
FROM ubuntu:18.04

LABEL maintainer="Jehan Ryan"

ENV DEBIAN_FRONTEND=noninteractive

# Get build time evnironment for PHP
ARG buildtime_version=7.1

# ENV variables
# Update base image
# Add sources for PHP
# Install software requirements
#
ENV ERRORS=0
ENV IMAGE_PHP_VERSION=$buildtime_version

RUN apt-get update \
    && apt-get install  -y gnupg tzdata \
    && echo "UTC" > /etc/timezone \
    && dpkg-reconfigure -f noninteractive tzdata

RUN apt-get install -y software-properties-common \
    && LANG=C.UTF-8 add-apt-repository ppa:ondrej/php

RUN apt-get update \
    && apt-get install -y curl zip unzip git supervisor sqlite3 \
       nginx \
       php${IMAGE_PHP_VERSION}-fpm \
       php${IMAGE_PHP_VERSION}-cli \
       php${IMAGE_PHP_VERSION}-pgsql \
       php${IMAGE_PHP_VERSION}-sqlite3 \
       php${IMAGE_PHP_VERSION}-gd \
       php${IMAGE_PHP_VERSION}-curl \
       php${IMAGE_PHP_VERSION}-memcached \
       php${IMAGE_PHP_VERSION}-imap \
       php${IMAGE_PHP_VERSION}-mysql \
       php${IMAGE_PHP_VERSION}-mbstring \
       php${IMAGE_PHP_VERSION}-xml \
       php${IMAGE_PHP_VERSION}-zip \
       php${IMAGE_PHP_VERSION}-bcmath \
       php${IMAGE_PHP_VERSION}-soap \
       php${IMAGE_PHP_VERSION}-intl \
       php${IMAGE_PHP_VERSION}-readline \
       php${IMAGE_PHP_VERSION}-xdebug \
```

```
        php-msgpack php-igbinary \
    && mkdir /run/php

RUN curl --silent --show-error https://getcomposer.org/installer | php \
    && ./composer.phar global require wapmorgan/php-code-fixer dev-master \
    && apt-get -y autoremove \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/* \
    && echo "daemon off;" >> /etc/nginx/nginx.conf

# tweak php-fpm config
RUN sed -i -e "s/;cgi.fix_pathinfo=1/cgi.fix_pathinfo=0/g" /etc/php/${IMAGE_PHP_VERSION}/fpm/php.ini
RUN sed -i -e "s/error_reporting = E_ALL & ~E_DEPRECATED & ~E_STRICT/error_reporting = E_ALL/g"
/etc/php/${IMAGE_PHP_VERSION}/fpm/php.ini
RUN sed -i -e "s/error_reporting = E_ALL & ~E_DEPRECATED & ~E_STRICT/error_reporting = E_ALL/g"
/etc/php/${IMAGE_PHP_VERSION}/cli/php.ini
RUN sed -i -e "s/;daemonize = yes/daemonize = no /g" /etc/php/${IMAGE_PHP_VERSION}/fpm/php-fpm.conf

ADD default /etc/nginx/sites-available/default
ADD supervisord.conf /etc/supervisor/conf.d/supervisord.conf

CMD [ "supervisord" ]
```