PHP Feature Analysis

Ashiqur Rahman
School of Computing
Queen's Universtiy
Kingston, ON, Canada
E-mail: rahman@cs.queensu.ca

***Abstract*—Evolution of a programming language is necessary to keep the language useful. Deprecation, removal and feature enhancement are some essential parts of evolution. However, selecting features for evolution is not an easy task. Feature analysis is an approach that can help in this process. It finds usage frequency of different language features in existing source files. In this project, we attempt to analyse the features of PHP, a general purpose web scripting language. PHP is very rich in features, but all are not equally useful, some features have alternative forms, and some are problematic. Therefore, programmers try to avoid unnecessary, confusing and problematic features in their programs. We analyse features of seven different categories. Our analysis finds usage frequency of various features, and shows differences exist in the usage of alternative forms of features, magic methods, magic constants, and functions. Attention of language engineers on those can help to improve the language in its future version.**

I. Introduction

The application area of computing is broad and evolving. Programming languages thus need to evolve to fulfil the changed needs of computing. Language engineers need to know which features programmers are using and which are not. Based on that they can decide how to improve a language in its future version. If a feature is popularly used, attempt can be taken to make it more efficient. On the other hand, a rarely used feature can be removed to keep the language structure lightweight, and to create more space for new features. If the language engineers believe an infrequently used feature has the potential to be very useful then they can try to understand why programmers are not using that, and can act accordingly.

In this project, we aim to investigate the appearance of various PHP features in PHP files. Though language feature analysis research exist in the literature, to the best of our knowledge, no one attempted to analyse the usage of PHP features before.

Feature analysis approach can help to find the usage of various features of a language. It takes existing code into account and analyse them to find in how many files, or how many times some selected features have been used. Insights from such analysis can help the language engineers to decide which features can be enhanced, or can be removed without affecting the users. If language engineers follow this approach, they are less likely to remove necessary features accidentally which can happen if they depend only on user feedback, or their own judgement.

We select PHP for feature analysis because it is one of the most popular web scripting language and rich in features. However, not all features are equally used or accepted among users. Users' discussion in many online websites and blogs reveal that [1], [2], [3], [4], [5]. Therefore, we get interest to explore PHP feature usage in source code to understand which features are good candidates for enhancement or removal.

Three open source PHP projects: Wordpress, phpBB and drupal are analysed in this study. Our analysis finds the usage of various PHP features in PHP files. It reveals PHP magic methods are not much used, all alternatives of a feature are not equally used and alternative form of a language construct without colon is preferred than the one with colon. However, there are few exceptions to these observations.

The results presented here is expected to help the language engineers to find the PHP features that need attention.

Besides, language developers who are interested to develop Domain Specific Languages to support web application development would be benefited. They would find this research interesting to know the usage frequency of various language features to understand their importance.

The work presented here can be extended in future in two ways: including more projects and analysing more features.

Paper Structure. Section II briefly introduces PHP and the features that were selected for analysis. Section III discusses our approach to feature analysis. Section IV and Section V present analysis and discussion on our research. Challenges faced during this project and limitations are outlined in Section VII and VIII respectively. After discussing related research in Section IX, conclusion and future work are given in Section X.

II. The PHP Language

PHP (Hypertext Preprocessor) is a general purpose web scripting language. It supports both functional and object oriented programming. PHP first appeared in 1995 [6]. Over the years it went through many evolutions and now PHP7 is available at the time of writing this report.

PHP is a server side scripting language. The end users are delivered plain HTML code after executing PHP code in the server. A PHP file can contain pure PHP code as shown in Figure 1, or PHP code embedded within HTML as shown in Figure 2. PHP code must be marked with special tags known as PHP tags.

```
<?php
$a = 1;
$b = 2;
function Sum()
{
global $a , $b ;
$b = $a + $b ;
}
Sum();
echo $b ;
?>

Output:
3
```

Fig. 1. PHP code

```
<html>
<head>
<title>Hello World</title>
</head>
<body>
<?php echo '<p>Hello World</p>' ; ?>
</body>
</html>
```

Fig. 2. PHP code embedded in HTML

PHP is rich in many language features. It provides lot of built-in funcitons. Many functions are available through extensions as well. A complete list can be found in [7].

A. Selected Features

Considering the duration of this project it is not feasible to analyse all PHP features. In this research, we are primarily concerned about some selected general programming features. The features are mainly selected by our own choice after studying PHP documentation [7], Beginning PHP 5.3 [8], w3school.com [9], and several websites and blogs [1], [2], [3], [4], [5].

Forms, database, multimedia, authentication, compression and archive related features were not considered due to the time needed for analysing them. Besides, features which are specially for debugging purposes were avoided. It is because developers would probably remove them before publishing their final code.

The selected features are categorized into the following seven groups.

- Output Functions
- Control structures with two different forms
- Magic Methods
- Magic Constant
- Internal Functions
- Special Functions
- Operators

Below is a brief description of the features that we put in each category.

1) Output Functions: We put four PHP functions in this category which are used to show output to users.

echo(): It prints multiple arguments.

print(): It takes a single argument.

printf(): This function produces output according to a particular format. An example of using printf() is given in Figure 3.

```
<?php
$num1 = 2;
$num2 = 3;

$sum = $num1 + $num2;
printf("Sum of %d and %d is %d", $num1, $num2, $sum);
?>

Output:
Sum of 2 and 3 is 5
```

Fig. 3. Use of printf()

vprintf(): This function can be used to display array value as a formatted string. Figure 4 shows an example of using vprintf().

```
<?php
vprintf ( "%04d-%02d-%02d" ,
 explode ( '-' , '1988-8-1' ));
?>

Output:
1988-08-01
```

Fig. 4. Use of vprintf()

2) Control structures with two different forms: PHP provides two different forms of *if, while, for, foreach* and *switch* statements. One form has a colon associated with it and another form is without any colon. Figure 5 gives an example of two alternative forms of the if statement.

```
<?php
if ($a ==5)
  echo "a is equal to 5";
?>


<?php
if ($a == 5):
echo \a is equal to 5";
endif;
?>
```

Fig. 5. Example of alternative forms of if

3) Magic Methods: PHP provides the following magic methods which are exclusive to classes. They must be defined inside a class and they are called automatically based on some particular event. Name of each magic method starts with __. Figure 6 gives an example using magic methods.

__construct(): It is a special method to define the constructor.

__destruct(): This method releases the resources held by an object.

__call(): If an object calls a method that is not implemented in its class then automatically the *__call()* method is called if implemented.

__callStatic(): This one is similar to the *__call()* method but static.

__set(): If an object tries to write data to inaccessible properties then *__set()* is called.

__get(): When an object tries to read data from inaccessible properties then *__get()* is called.

__isset(): If isset() is called on an inaccessible property then the magic method *__isset()* is called automatically.

__unset(): If unset() is called on an inaccessible property then the magic *__unset()* method is called automatically.

__sleep(): This method is used when an object of a class is to be serialized. It returns an array containing a list of class members that need to be serialized.

__wakeup(): To un-serialize an object this method needs to be used. It is opposite of the *__sleep()*.

__tostring(): It defines the behaviour of a class when the class is treated as a string.

__invoke(): Objects with the *__invoke()* method defined in the class can be called like a function.

__set_state(): This method is called automatically for classes exported by var_export().

__clone(): Once cloning of an object is done then *__clone()* is called to change any necessary properties of the cloned object.

__autoload(): When a script tries to create an object but doesn't get the class in the script then it can call the *__autoload()* to get the class definition.

__debugInfo(): Though it is used for debugging purposes, we have considered this method for the sake of completeness of analysing magical methods. It is called by var_dump() method and can show the properties of an object.

4) Magic Constant: A large number of predefined magic constants are provided by PHP. Below are seven of them which change based on in which script they are placed, and where they are placed inside a script. Figure 7 shows example of using magic constant.

__FILE __: It returns the full path of the .php file that calls the *__FILE __* constant.

__DIR __: Directory of a file can be found using this constant.

__FUNCTION __: It gives the function name.

__CLASS __: It gives the name of the class with namespace.

__METHOD __: Class method name is given by this constant.

__LINE __: The current line number of a file can be retrieved using *__LINE __:*.

```
<?php
class MyClass
{
public $userName ;

public function __construct ( $userName )
{
$this -> userName = $userName ;
}

public function __toString ()
{
return $this -> userName ;
}
}

$myOb = new MyClass ( 'Bob' );
echo $myOb ;
?>

Output:
Bob
```

Fig. 6. Example of magic method

```
<?php
echo "File is '" . __FILE__ . "'.\n";
echo "Directory is '" . __DIR__ . "'.\n";
?>
```

Fig. 7. Example of Magic constant

__NAMESPACE __: This constant gives the name of the current namespace.

5) Internal Functions: In this project we use the TXL grammar of PHP developed by Eshkevari et al. [10]. In that grammar the following functions are described as internal function.

new: It is used each time a new instance of a class is created.

clone: It creates a new copy of a given object. The copy can perform all operations of the original object without affecting the original.

isset(): It is used to check whether a given variable is set and is not NULL.

empty(): If a variable does not exist or FALSE then that can be determined using the empty().

intval(): Returns the integer value of the given variable.

eval(): A string can be evaluated as a PHP code using the eval(). Figure 8 shows an example of using eval() taken from [7].

exit(): exit() terminates the current script and can optionally print a message.

include: It is used to insert the content of one .php file into another.

require: require and *include* are identical except their techniques to handle error situations.

include_once: Same as *include* but stops a script from being included more than once.

```
<?php
$string = 'cup' ;
$name = 'coffee' ;
$str = 'This is a $string with my $name in it.' ;

echo $str . "\n" ;
eval( "\$str = \" $str \";" );
echo $str . "\n" ;
?>

Output:
This is a $string with my $name in it.
This is a cup with my coffee in it.
```

Fig. 8. Use of eval()

require_once: Same as *require* but does not include a script more than once.

use: This operator assigns aliases to names of classes, interfaces and other namespaces.

6) Some Special Functions: PHP has many special functions. Analysing them is a time consuming task. Below are few special functions that we find interesting to investigate.

unset(): It destroys a given or a set of variables.

create_function(): This is also known as anonymous function [11] and used to create functions dynamically. An example of using the create_function() taken from [7] is shown in Figure 9.

```
<?php
$newfunc = create_function
('$a,$b', 'return "ln($a) + ln($b)
 = " . log($a * $b);');
echo "New anonymous function: $newfunc\n";
echo $newfunc(2, M_E) . "\n";
?>

Output:
New anonymous function: lambda_1
ln(2) + ln(2.718281828459) = 1.6931471805599
```

Fig. 9. Use of create_function()

explode(): It breaks a string into an array. An example has been shown in Figure 10.

```
<?php
$listOfUsers = "BOB,ALICE,SAM";
$userArray = explode( ",", $listOfUsers);
print_r $userArray
?>

Output:
Array ( [0] => BOB [1] => ALICE [2] => SAM)
```

Fig. 10. Use of explode()

implode(): It is opposite of *explode()*, returns a string from an array. Figure 11 depicts an example of using the implode().

declare(): The declare function can be used for two purposes: one for declaring the encoding of a php script, and another for defining the frequency at which tick function is called.

```
<?php
$userArray = array( \BOB, ALICE, SAM");
$userList = implode( \,", $userArray );
echo $userList
?>

Output:
BOB, ALICE, SAM
```

Fig. 11. Use of implode()

fmod(): It is used to get the modulo of float values.

array_shift(): It removes the first element of an array and returns that element.

array_unshift(): Inserts new elements at the beginning of an array.

array_push(): Insert elements at the end of an array.

array_pop(): The last element of an array can be deleted using *array_pop()*.

array_splice(): It can remove portion of an array and can replace them with new elements. An example is given in Figure 12.

```
<?php
$userArray = array( "Bob" , "Alice" , "Sam" );
array_splice ( $userArray , 2);

print_r $userArray
?>

Output:
Array ( [0] => BOB [1] => ALICE [2] => Alice)
?>
```

Fig. 12. Use of array_splice()

array_diff():array_diff() can be used to get the differences between two arrays.

list(): Values of an array can be assigned to a list of variables using the list() function. Figure 13 shows an example of using the list().

7) Operators: We do not consider general purpose operators and those that have known common uses such as +, - , $->$ etc. We consider some special operators and those that have alternatives.

Heredoc operator(<<<): Heredoc can be used to write multi line document string. <<< represents heredoc. In addition, an identifier is needed to mark the start and end of each document. An example of using heredoc is given in Figure 14.

Array Keys (=>:) When creating an array it can be initialized with array keys. An instance of using array key has been shown in Figure 15.

as: Assigns each array element to a variable. Figure 16 shows an example of using the as operator.

Equal (==): The equal operator only checks whether the left and right hand values are equal but ignores their types.

```
<?php
$userArray = array(\BOB, ALICE, SAM");

list( $firstPerson, $secondPerson,
     $thirdPerson ) = $userArray;

echo $firstPerson . \ < br/ > ";
echo $secondPerson . \ < br/ > ";
echo $thirdPerson . \ < br/ > ";
?>

Output:
BOB
ALICE
SAM
```

Fig. 13. Use of list()

```
<?php
echo <<<EOT
User Name is "$name"
Email is "$email"
Address is "$address"
EOT;
?>
```

Fig. 14. Heredoc (<<<)

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);
?>
```

Fig. 15. Initializing array =>

```
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as $value) {
    echo $value;
}
?>

Output:
1234
```

Fig. 16. as operator

Identical (===): The identical operator checks both the value and the type of the operands.

Not equal (!=): It returns true if the value of the operand at left hand is not equal to the value of the operand at right hand.

Not Equal (<>): Like !=, it also returns true if the value of the left operand is not equal to the value of the right operand.

Not identical (!==): It returns true if either the values are different or the types are different.

Logical (and): It performs logical And operation.

Conditional (&&): It performs conditional And operation.

Logical (or): If either the left or the right operand is true it returns true.

Conditional (||:) It also returns true if either the left or the right hand operand is true.

Scope Resolution Operator(::): Scope resolution operator is used to access static, constant and overridden properties of a class. Figure 17 shows an example taken from [7].

```
<?php
class MyClass
{
protected function myFunc () {
echo "MyClass::myFunc()\n" ;
}
}

class OtherClass extends MyClass
{
// Override parent's definition
public function myFunc ()
{
// But still call the parent function
parent :: myFunc ();
echo "OtherClass::myFunc()\n" ;
}
}

$class = new OtherClass ();
$class -> myFunc ();
?>

Output:
MyClass::myFunc()
OtherClass::myFunc()
```

Fig. 17. Scope resolution operator (::)

Ternary Operator(? :): It is a shorthand alternative that acts like an if-else construct. Figure 18 gives an example of ternary operator.

```
<?
$age = 17;
$msg1 = \Cannot apply for driving licence";
$msg2 = \Can apply for driving licence"";
echo ( $age > 17 ) ? $msg1 : $msg2;
?>

Output:
Cannot apply for driving licence
```

Fig. 18. Ternary operator (?:)

III. Approach

We chose the following three open source PHP projects to find the usage of the selected PHP features. Availability of these projects and previous analysis on them by Eshkevari et al. [10] encouraged us to use them in our project.

- **WordPress [12]:** It is one of the top blogging and content management system. WordPressis free, open source, and written in PHP. More than 60 million websites are supported by WordPress. Till now four versions of WordPress have been released [13].

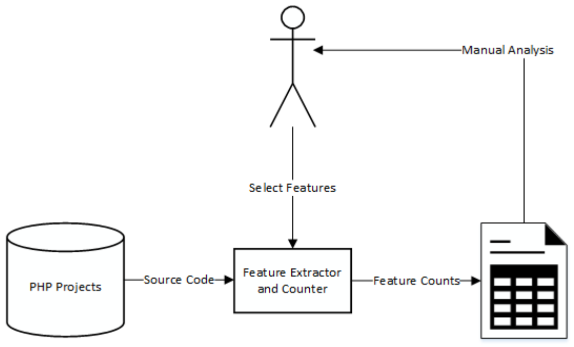

Fig. 19. Feature Analysis Process

- **Drupal [14]:** It is another free and open source content management framework. Drupal is written in PHP using Symfony [15]. Its latest release is Drupal 8 [16]. Drupal has over 100,000 active contributors.
- **phpBB [17]:** *phpBB* stands for php Bulletin Board. It is an internet forum package and written using PHP. Multiple databases are supported by phpBB such as MySQL, Oracle, SQLite etc. So far three versions of phpBB have been released [18].

Figure 19 shows an overview of the process that was followed to analyse the PHP projects. To extract and count the desired features from the projects an extractor was developed using TXL. It uses the same PHP grammar that was developed and used by Eshkevari et al. [10]. Figure 20 presents a snippet of the developed extractor. After getting the extracted results they were analysed manually.

```
include "php.grm"

function main
    match [program]
PHPProgram [program]
...............................

construct EqualityOps[EqualityOp*]
_ [^ PHPProgram]
   construct _ [number]
_ [length EqualityOps] [putp "EqualityOps: %"]

%....Find != ......
construct NotEquals[EqualityOp*]
_ [getNotEqual  each EqualityOps ]
construct _ [number]
_ [length NotEquals] [putp "NotEquals: %"]

...............................
end function

function getNotEqual
CurrentEqualityOp[EqualityOp]
replace [EqualityOp*]
PreviousEqualityOps[EqualityOp*]

deconstruct CurrentEqualityOp
'!=
by
PreviousEqualityOps[.CurrentEqualityOp]
end function

...............................
```

Fig. 20. A snippet of the extractor written in TXL

IV. Analysis

Table I gives an overview of our three analysed projects: WordPress, Drupal and phpBB. Total number of .php files was over 1, 700 in the analysed projects.

TABLE I
Overview of the Analysed Systems

1.	#Projects	3
2.	#Files	1,740
3.	#Classes	443.
4.	#Files containing classes	321
5.	#Functions	7,144
6.	#Files containing functions	528

A. Output Functions

Four PHP output functions were analysed echo(), print(), printf() and vprint(). We did not consider those which do not display the output or used mainly for debugging, for example sprintf(), fprintf() etc.

Before the analysis we expected that echo and print would be the most popularly used output statements. They frequently appear in the examples of most websites. However, this assumption was partly true. As it is clear from Figure 21, after echo, C style printf() is the second highest output statement that appear in different files of the analysed projects. On the other hand, vprint() was not found in any of the files.

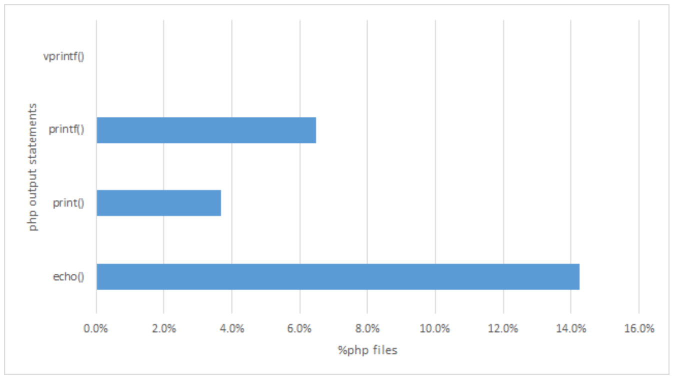

Fig. 21. Output statements

Summary: *Programmers often need to give formatted output.*

B. Control structures with two different forms

Five PHP statements if, while, switch, for and foreach have two different forms. They can be used with colon or without colon. From Figure 22, it is apparent that statements without colon are more popular than statements with colon.

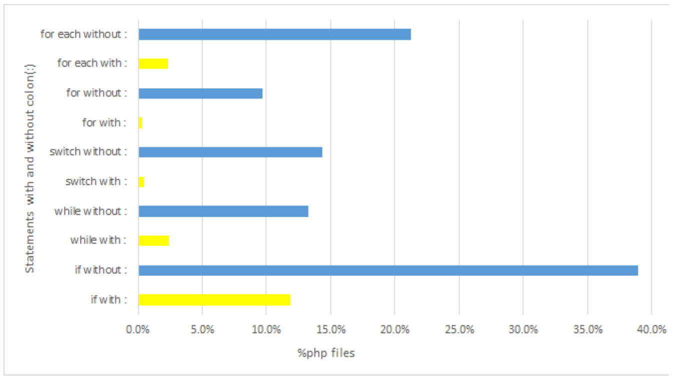

Fig. 22. Statements with colon or without colon

In case of for and switch the colon forms were found in less than 0.5% .php files. Though if with colon was found in nearly 11% files, it is roughly one-fourth of the total percentage of files that use if without colon. The colon forms of foreach and while was found in around 2% files which is very low compared to their without colon forms.

Summary: *Statements without colon forms are more popular than their alternative forms with colon.*

C. Magic Methods

Use of magic methods is related to the object oriented concept implemented in applications. Figure 23 illustrates the use of magic methods in the selected projects.

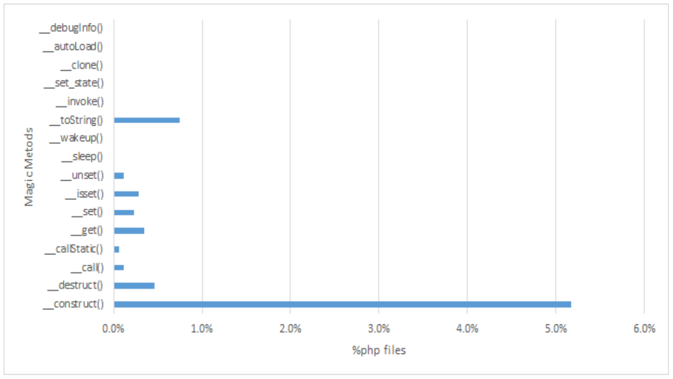

Fig. 23. Magic methods

In the analysed projects total 443 classes were implemented in 321 .php files, which represents approximately 18% of the total .php files. However, nearly half of the magic methods were never used in any file. Except __construct() that was implemented in approximately 5% files, other magic methods were implemented in less than 1% files. Among them __toString() was found in nearly 0.7% files, whereas, others were found in less than 0.5% files.

Summary: *Except few methods, programmers tend to use magic methods rarely.*

D. Magic Constant

PHP supports many magic constants, but in this study, only seven of them were selected for usage frequency analysis. They were selected based on the feedback provided in different websites. Figure 24 shows the use of the selected magic constants. Among these seven features, four features appeared in less than 0.5% files.

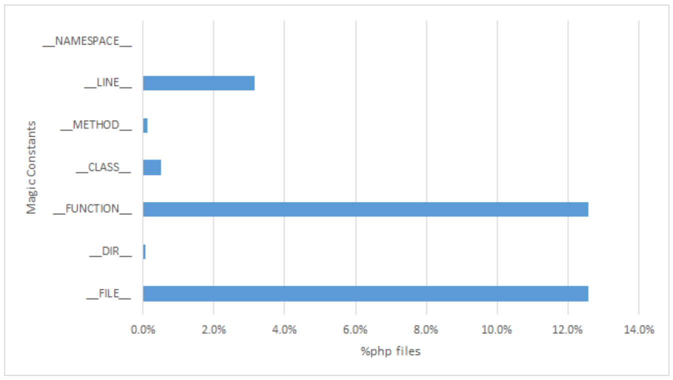

Fig. 24. Magic constants

It was interesting that though __FUNCTION __appeared in more than 12% files, __METHOD __appeared in less than 0.1% files. Similar observation is true for __FILE __and __DIRECTORY __
.

Summary: *Some magic constants are very popularly used.*

E. Internal Functions

Among the internal functions, it was expected that functions related to file inclusion and object creation would appear in high frequency. Our assumption was true as we can see in Figure 25.

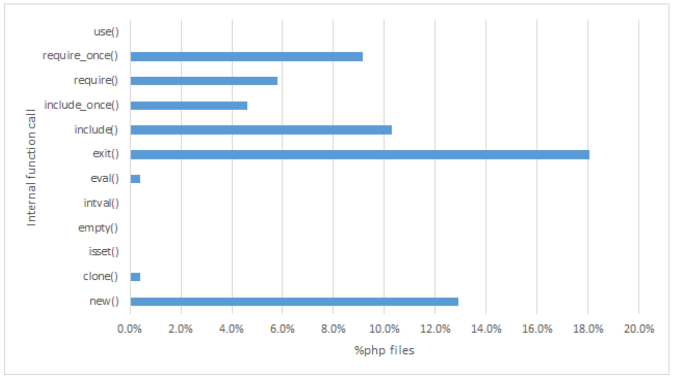

Fig. 25. Internal function

However, among the rest of the internal functions, except exit(), other functions were called very infrequently (in less than %0.5 files), and one third of them was never called in the selected projects.

Summary: *Few internal functions are frequently used.*

F. Special Functions

Figure 26 shows the percentage of php files in which the selected special functions appeared.

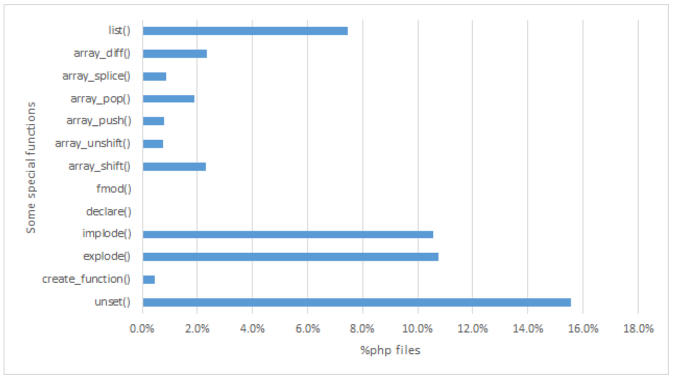

Fig. 26. Some special functions

Except fmod() and declare(), all other selected special functions were found in use in the analysed projects. Among the selected functions, unset() was at top with appearance in more than 15% php files, followed by implode and explode in more than 10% files.

There is very little difference in the number of files in which implode() and explode() appeared. From the percentage of files in which they appeared it seems conversion between array and string is a very frequent task.

Summary: *Array related special functions have good use to the programmers.*

G. Operators

Logical *and, or, as, heredoc,* and an alternative of *not equal (<>)* have very low usage count among the operators that we analysed. Figure 27 presents the percentage of files in which they appeared.

Though some operators can be used as alternatives, such as <> and !=, programmers tend to adhere to a single form.

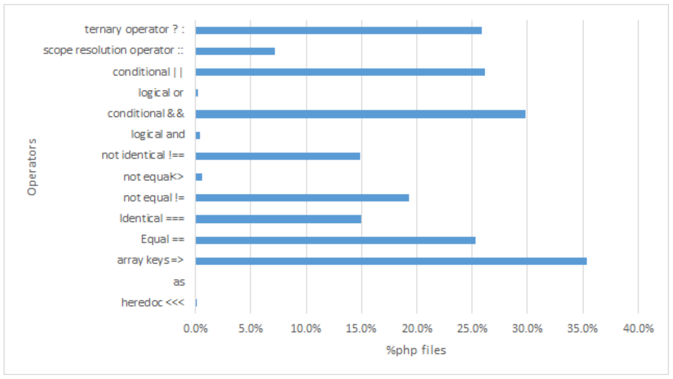

Fig. 27. Operators

Summary: *Instead of switching between alternative operators for the same task, programmers tend to adhere to a particular one.*

V. Discussion

In this project we analysed PHP features, and found useful information about their usage. The insights obtained can help the language engineers to make decisions about improving the language further.

Making the final decision about the features totally depend on language engineers. In no way, the purpose of this report is to say that some particular features were not used at all, or rarely used so those must be removed from the language.

Though language engineers can make such decisions, many other useful decisions can also be made from the presented analysis. For example, low usage of magic methods may mean that programmers are not much familiar with them. In that case, programmers need to encourage more through promoting those particular features. It may also mean that those particular features require more clear descriptions in the documentation.

The observations presented here can also reveal opportunities for improvement. As an instance, low usage of a feature may also mean that users do not find that particular feature efficient enough in their application, or their syntax is not handy to use.

The function eval() is an example which has bad reputation. Programmers are sometimes worried about the use of eval(). Even in the PHP documentation it is mentioned as a dangerous [7] feature. However, this analysis confirms that though not very frequent, programmers use eval() in programs. In such scenarios, language engineers can think deeply about removing, or limiting the effect of such features.

VI. Lessons Learned

In this project we attempted to find the usage of various PHP features from three existing web application written in PHP. The analysis presented here is expected to help the evolution of the PHP language. Below, is a brief discussion about the lessons we learned through this project.

- **Feature analysis is also applicable to PHP**
  There is no doubt that there are features in PHP language which are not frequently used. Those can be identified by applying feature analysis technique, and can be considered for enhancement or removal.
- **Some manual tasks seem difficult to automate in the feature analysis process**
  Though we used an automated technique to extract features, selecting the features that need to be analysed and analysis of the extracted results require human effort.
- **PHP feature analysis should be targeted on parts of PHP**
  PHP is rich in features. To do this project the complete PHP documentation was downloaded from [7], that is over 5,000 pages long document. Analysing all of them together is a big task though not impossible. However,

separate analysis on different feature groups of PHP will be easy to complete.

- **PHP is dynamic as a programming language**
  PHP language engineers work actively to make it more convenient by removing and adding new features over time. In the PHP documentation [7] we found evidences of addition, deprecation and removal of language features.
- **TXL is a very handy tool to extract features from source code**
  The extractor we developed for this project worked very fast and generated results within a couple of seconds.

VII. Challenges Faced

Some challenges were faced during this project. Below is a brief discussion on them.

- **Short duration of the project**
  The main challenge we faced during this project is its short duration. Though we had a plan to analyse more features, due to limited time that was not possible
- **Selecting the features to be analysed**
  PHP has lot of features. Selecting some of them for analysis was a difficult task.
- **Confusing naming convention of PHP**
  PHP does not follow any strict naming convention [6]. It sometimes confused us.

VIII. Limitations

Though we analysed general language features of PHP, we did not include all of them. PHP has many features, and it is time consuming to analyse all of them. Considering the duration of this project we attempted to analyse only some selected general programming features. For the same reason, we avoided special features such as form, database and xml related features.

However, analysis of other features would not be very difficult. The extractor developed for the analysis purpose can easily be extended to cover more features.

Features were selected mainly based on our own choice, and users' discussion in different websites and blogs. No specific rule was followed to select them. Other researchers may find different features more interesting to investigate than the ones we selected. The only solution to this problem is to do a complete analysis of all PHP features.

We analysed only three projects. This is very low in number. It would not be surprising if analysis on other projects gives different results. Some more projects are needed to analyse to make more concrete decisions. However, that can also be done very easily with the developed tool.

IX. Related Work

Martin et al. performed Make language feature analysis in [19]. The data they obtained through the feature analysis process shows various make features with their usage frequency, and differences in feature usage between automake and manually created make files. Besides, the presented data shows evidence that few obsolete make features are still in use. This work is closely related to ours, but we perform feature analysis on PHP language.

In [10] Eshkevari et al. performed static and dynamic analyses to investigate type changes of PHP variables. Three open source PHP projects Wordpress, phpBB and drupal were analysed. The authors found dynamic type change is rarely used in PHP, and can be easily converted to static type. In order to parsing the PHP files, they developed a TXL grammar for PHP [20]. We also investigate the same PHP projects that were used in [10], and use the same grammar that was developed by Eshkevari et al. for parsing PHP files. However, our goal is completely different.

Nguyen et al. analyzed four open source PHP projects in [21]. They discovered a special form of refactoring applicable to dynamic web applications. However, our objective is only finding the usage frequency of various PHP features in PHP projects.

X. Conclusion and Future Work

The main aim of this project was to find the usage frequency of various PHP features. An extractor was written in TXL for that purpose, and analysis was performed on three open source PHP projects WordPress, Drupal and phpBB. Insights obtained through the feature analysis process have been presented in this report.

Among the selected features we found some features are popularly used, some have very low usage frequency, and some are not used at all. The observations presented here, such as low usage of magic methods, programmers' preference to a single form of an operator when multiple are available and other would be helpful to language engineers to enhance the language.

Due to the large size of PHP feature set it was not possible to analyse all of them. We mainly focused on general programming features. However, the developed extractor can be extended in future to add more features for analysis. Besides, more projects could be analysed to get more concrete insights.

Acknowledgement

We are really grateful to Dr. James R. Cordy, Professor, School of Computing, Queen's University, who is also one of the authors of [10], for permitting us to use the TXL grammar of PHP in our project, and all his valuable suggestion to make this project successful.

References

[1] D. Embry. (2017, March) Why PHP Sucks. [Online]. Available: https://webonastick.com/php.html
[2] L. Munroe. (2017, March) PHP: a fractal of bad design. [Online]. Available: https://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design/
[3] S. Scriven. (2017, March) Problems with PHP. [Online]. Available: http://toykeeper.net/soapbox/php_problems/
[4] A. Pagaltzis. (2017, March) In which I write about PHP for the first and the last time. [Online]. Available: http://plasmasturm.org/log/393/
[5] E. Martin. (2017, March) What i don't like about PHP. [Online]. Available: https://bitstorm.org/edwin/en/php/
[6] Wikipedia. (2017, March) PHP. [Online]. Available: https://en.wikipedia.org/wiki/PHP
[7] The PHP Group. (2017, March) Documentation. [Online]. Available: http://php.net/docs.php

[8] M. Doyle, *Beginning PHP 5.3*. John Wiley & Sons, 2011.
[9] w3schools.com. (2017, March) PHP 5 Tutorial. [Online]. Available: https://www.w3schools.com/php/
[10] L. Eshkevari, F. Dos Santos, J. R. Cordy, and G. Antoniol, "Are php applications ready for hack?" in *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 2015, pp. 63–72.
[11] Wikipedia. (2017, March) Anonymous function. [Online]. Available: https://en.wikipedia.org/wiki/Anonymous_function
[12] WordPress.Org. (2017, March) Meet WordPress. [Online]. Available: https://wordpress.org/
[13] Wikipedia. (2017, March) WordPress. [Online]. Available: https://en.wikipedia.org/wiki/WordPress
[14] Drupal. (2017, March) Drupal. [Online]. Available: https://www.drupal.org/
[15] Wikipedia. (2017, March) Symfony. [Online]. Available: https://en.wikipedia.org/wiki/Symfony
[16] ——. (2017, March) drupal. [Online]. Available: https://en.wikipedia.org/wiki/Drupal
[17] phpBB. (2017, March) phpBB. [Online]. Available: www.phpbb.com
[18] Wikipedia. (2017, March) phpBB. [Online]. Available: https://en.wikipedia.org/wiki/PhpBB
[19] D. H. Martin, J. R. Cordy, B. Adams, and G. Antoniol, "Make it simple: An empirical analysis of gnu make feature use in open source projects," in *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*. IEEE Press, 2015, pp. 207–217.
[20] Txl. (2017, March) The Txl Programming Language. [Online]. Available: http://www.txl.ca
[21] H. A. Nguyen, H. V. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Output-oriented refactoring in php-based dynamic web applications," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, 2013, pp. 150–159.