UNIVERSITY OF AMSTERDAM

PHP re-factoring: HTML templates

Name of candidate:	Dimitrios Kyritsis
E-mail:	jim9ky@yahoo.gr
Date:	September 2013

Host organization:	Centrum Wiskunde & Informatica (CWI)
URL:	http://www.cwi.nl/
Address:	Science Park 123, 1098 XG Amsterdam
Contact persons:	Mark Hills - Mark.Hills@cwi.nl
	Jurgen J. Vinju - Jurgen.Vinju@cwi.nl

Abstract

For every software developer the goal, when he creates software, is that the final result of his efforts meets its initial requirements. The software should operate as it was meant to, without any or at least without significant inconveniences. However, experienced developers have another goal, to write simple code. Their experience has taught them that the simpler the code is, the easier will be its maintenance. For web application developers the situation is the same. A way for them to write simpler web applications with easily maintainable code is to use template systems. Thereby, they can separate business logic from its presentation making their lives way easier, especially if they have to collaborate with web designers. Developers and designers will then have different fields of concern and the changes that they make will not affect each other's work.

This thesis will describe our efforts to create an application which can automatically transform normal code into uses of template systems. In the beginning we will enumerate the factors that were taken into account for the creation of this application, along with a rationale for them. Subsequently, we will describe the steps and the decisions that were taken during the creation of our application. Finally, there will be a validation phase which will show if our application can really offer the advantages that was created for. The results will reinforce our research and will demonstrate the initial reasons for creating it.

Contents

Table of Contents

Figures

Listings

Tables

Preface

This project is the master thesis for the degree in Master of Science in Software Engineering at the University of Amsterdam. The drafting of this thesis was a difficult task which would not have come true without the help of some people. At first, I would like to thank my two supervisors, Mark Hills and Jurgen Vinju who helped me a lot with their knowledge on the topic of code analysis and transformation. They were always willing to help me when a new problem would rise. Thereafter, I would like to thank my family for their ethical and financial support. Finally, I would like to thank all these people that made my life in Amsterdam a beautiful experience.

Chapter 1 - Introduction

Martin Fowler in his book “Re-factoring**- Improving the Design of Existing Code**” [1] defined re-factoring as “the process of changing a software system in such a way that it does not alter the external behaviour of the code, yet improves its internal structure”. Code re-factoring is a way to make existing code cleaner by improving its design and additionally prevents the appearance of bugs. But what if the code doesn't look at all as it was before after its transformation, yet its results are the same and perhaps better? And what if the code supports more features than before? Is this still considered as re-factoring?

The goal of this thesis is the implementation of a tool that automatically re-factors specific PHP code into uses of template systems. More specifically our tool should automatically restructure PHP generated HTML code, into code that will use a template engine (the Smarty template engine in our case) to process web-templates. The result in our browser should be similar before and after the use of Smarty, while in specific cases some security issues that existed will be overcome.

The structure of our thesis is as follows: Chapter 2 contains the problem statement and motivation for this study. This chapter helps the reader understand why we wrote this thesis along with the difficulties that we faced during its preparation. Chapter 3 refers to the context (including technologies and concepts that are important for the understanding of our actions) and to related work. In Chapter 4 we describe the research method that we followed during our research. In Chapter 5 we describe our prototype, its abilities and its weaknesses and finally Chapter 6 contains the validation of our research, explaining why it is useful. It also describes the future work that needs to be done.

We hope that our research on the topic of template systems and code re-factoring will give value to this thesis and will make this project a remarkable endeavour. We also hope that the implemented tool will become a notable prototype, which after some future work will be able to offer all the advantages that was implemented for.

Chapter 2 - Problem statement and Motivation

2.1 Overview

Generally there is a great deal of interaction between PHP and HTML. More precisely PHP can generate HTML code and HTML can pass information to PHP. Fragments of HTML can also be intermingled with PHP. This provides a way of outputting HTML. Another way is to generate HTML through PHP by using the "echo" and "print" commands of PHP followed by the HTML code. When creating web applications using PHP and HTML web-developers and designers usually have to collaborate with each other (at least when they are not the same person). This collaboration might sometimes create problems especially in the case of code maintenance. For example if one day the programmer (web-developer) has to make a change in the application logic of the program, he might have to affect the presentation too. The reason is that both the application logic (PHP) and presentation (HTML) co-exist in the same files. The same thing might occur when the designer has to make changes in the presentation logic. A way to overcome this problem is to use a template engine.

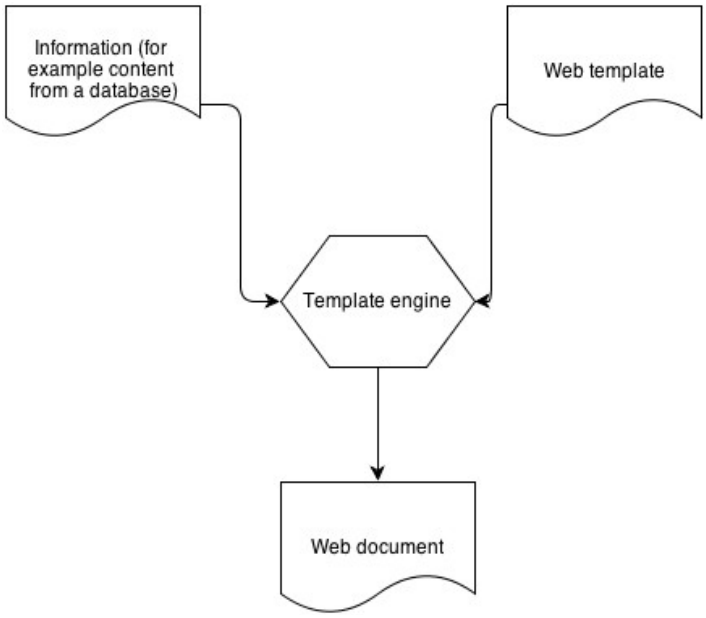

Figure 2.1: Web Template System

A template engine is part of a template system. Its aim is to produce web documents by combining the information that receives from the processing of web templates and content information (for example data from a database). Web templates constitute the other part of a template system and are the means to accomplish the separation of application logic from presentation [3]. In general this separation can provide solutions to many problems and improve web application development along with security.

In the case of security, template systems can insulate the templates from the PHP (the case that we are dealing with), creating a controlled separation of presentation from business logic. Template engines also have security features like security filters, that can enforce restrictions on templates, preventing malicious users to pursue attacks (for example XSS attacks that we will discuss later).

2.2 The difference of using and not using a template engine

To illustrate the operation of template systems and the difference between using and not using them, we will show a simple example. This example consists of two parts. In the first part we can see the code of a simple PHP program, created without the use of a template engine, while in the second part we can see the code of the same program created with the assistance of the Smarty template engine(we will provide more information about the Smarty template engine in the next chapter). In **Listing 2.1** we can see the first case:

```
<html>

<head>
<title>Encapsed case and concatenation</title>
</head>
<body>
<?php

$lang= "PHP";
$lang2= "Rascal";
echo  "<h1>I love $lang and $lang2!</h1><br>";
$lang= "C";
print "<b>I love " . $lang . " and $lang2</b>";

?>
</body>
</html>
```

Listing 2.1: PHP program created without the use of a template engine

The above is a simple PHP program that we created to test our tool. We can see that the application logic (variable assignment) is combined with the presentation (the HTML inside the "echo" and "print" commands). In **Listing 2.2 and 2.3** we can see the second case:

```
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$lang = 'PHP';$smarty->assign('val1',$lang);
$lang2 = 'Rascal';$smarty->assign('val2',$lang2);
$lang = 'C';$smarty->assign('val3',$lang);
$smarty->display('showTemplate.tpl');
?>
```

Listing 2.2: PHP program created with the use of the Smarty template engine- The logic

```
{* Smarty *}
<html>

<head>
<title>Encapsed case and concatenation</title>
</head>
<body>

<h1>I love  {$val1}  and  {$val2} !</h1><br>
<b>I love {$val3} and  {$val2} </b>

</body>
</html>
```

Listing 2.3: PHP program created with the use of the Smarty template engine – The presentation

The second case consists of two programs. **Listing 2.2** is a PHP program which encloses the business logic of the original program. Lines 1-11 serve as a link between our system and Smarty (they represent the locations where important Smarty files exist in our system) and will be displayed in every transformation that will be made later by our tool. **Listing 2.3** represents the template, which encloses the presentation logic. The variable assignment is made by the programmer in the program of **Listing 2.2** (lines 13-16). Then these variables can be used by the designer in the program of **Listing 2.3** (lines 9-10). That means that unlike the case of **Listing 2.1** these two types of developers don't need to interfere with each others' work.

2.3 Research Question

Our thesis answers the following research question :

- **Is it possible to automatically transform hand-crafted HTML into uses of template systems?**

However, for a sufficient answer we had to face challenging tasks like the below ones:

- The HTML is usually generated using print statements scattered all over the code and not by generating the whole HTML code in once.
- HTML generation has to take account of the control flow.
- Transformation to templates requires parsing the HTML code, which might also require dealing with HTML that does not conform to any specific standard (or maybe with broken HTML code).
- If some of the tags are given in a more dynamic fashion (for example returned from function calls or assigned to the same variable with different values on different paths), data flow analysis will be needed to correctly determine what to generate.
- The above point sometimes make it difficult to keep the number of templates down to a minimum.
- PHP commands like “print” or “echo” could be given in an “eval()” PHP function. Additionally, emitted values of HTML tags could be given directly in form posts and thus will be unknown in the code. This could even happen with tags generated from normal functions.

In our project we were able to overcome successfully some of those problems. For the remaining, future work will be needed.

2.4 Motivation

The motivation behind this thesis project was mainly the advantages that can be provided by the separation of business logic from the presentation logic and the security that is granted by the template engines [2]. More precisely the separation of the two forms has the following advantages:

1. The presentation logic (templates) and the business logic (data model) represent two different entities.
2. Designers and developers can work in parallel without being involved in each others work. This reduces inconveniences and communication costs. For example, a designer can work independently on the layout of a website without any disturbance from the programmer, who is responsible for the website's logic.
3. Designers can “break” templates into sub-templates and then reuse them whenever they want. Except the reuse,this technique provides also simpler and “cleaner” code.
4. The application's maintenance becomes easier. If the designer wants to make a change in the layout,

he only needs to change the template, not the whole program. This also applies for the programmer. In general, changing a program is much riskier than changing a template. So possible interaction between developers and designers in the same file makes it even riskier.

5. Template systems usually have features that can provide security to web applications (security filters).
6. The overall code of the application (both template and data model code) will be more flexible after using a template system, because of the separation of responsibilities.

Chapter 3 - Context and related work

3.1 Overview

In this chapter we will present the context of our thesis and some related work. More precisely we will discuss about the technologies which we used for the creation and the testing of our prototype (and the reason we used them), along with some concepts that are important for the understanding of our actions. In the end of the chapter we will show previous related work of other researchers on similar topics.

3.2 HTML

HTML stands for Hypertext Mark-up Language. It is a mark-up language which has been used by the World Wide Web (WWW) since 1990 and is widely regarded as the standard publishing language of it. With HTML a user can create platform independent hypertext documents, that is documents that are portable from one platform to another. These kind of documents are SGML (SGML stands for Standard Generalized Mark-up Language, an ISO-standard technology for defining generalized mark-up languages for documents) documents [6]. They contain generic semantics that are appropriate for representing information from a wide range of domains. Some examples are hypertext news, mail, database query results, menus of options and documents with in-lined graphics[1].

In general HTML is a mark-up language that interacts a lot with PHP. As we stated in Chapter 2 PHP can generate HTML, and HTML can pass information to PHP. This interaction and especially HTML generation by PHP is the issue that concerns us[2].

3.3 PHP

PHP stands for PHP: Hypertext Preprocessor and it is an HTML-embedded scripting language widely used by web developers all over the world[3]. Much of its syntax is borrowed from C, Java and Perl. The goal of the language was to allow web developers to write dynamically generated pages. In general PHP focuses on server-side scripting, a technique which involves embedding scripts in HTML source code (client's requests). These requests are executed thereafter on the server before the plain HTML result is sent back to the browser[4]. That means that PHP differs from client-side scripting languages because its code is executed on a server, generating HTML which is then sent to the client. The code which is sent will be unknown to the client and only the results will be shown[5].

As we stated before, PHP can generate HTML code. A way that this can be accomplished is by using the “echo” and “print” commands of PHP followed by the HTML code. In our case we will try to separate the PHP code from the HTML (the presentation from the business logic) with the objective to make our program simpler and safer. In order to do this, we will first parse the PHP code and subsequently with the help of Rascal (see **Chapter 3.5**) we will re-factor it into using Smarty, a widely used template engine.

1 http://tools.ietf.org/html/rfc1866
2 http://php.net/manual/en/faq.html.php
3 http://nl1.php.net/manual/en/faq.general.php
4 http://nl1.php.net/manual/en/faq.html.php
5 http://www.php.net/manual/en/intro-whatis.php

3.4 Smarty

Our main goal is to analyse hand-crafted HTML code and then transform it automatically to code which uses a PHP template engine. The template engine that we chose to use is Smarty. It is a widely used template engine and its main task is to separate the presentation from the application logic of a PHP program, making it look "cleaner". More precisely it cuts off the PHP code from the presentation and it provides a simpler tag-based syntax. This helps especially web-designers by allowing them to skip learning the PHP syntax, but mainly it simplifies cases where they have to maintain HTML mixed with PHP code. Apart from this, Smarty provides tools for managing the presentation. A useful tool is for example template inheritance. That means that if a project consists of a huge number of templates, Smarty can keep template maintenance simple using this feature[6]. Finally Smarty can provide security for our applications with the use of escaping content filters.

In general Smarty doesn't aim to replace PHP. It is just a tool to separate presentation from application logic[7]. However some of its advantages like:

- clean tag-based syntax
- easy to maintain
- its flexibility
- the security it offers
- free and open source
- sufficient documentation

made it the proper solution for our project.

3.5 Rascal

Rascal is a domain-specific language developed and tested in CWI (Centrum Wiskunde and Informatica) in Amsterdam. Its field is source-code analysis and manipulation (SCAM) or meta-programming as it is widely known. Rascal has 3 specific goals [4]:

- to diminish the complexity of integrating analysis and transformation tools
- to offer the developers a safe and interactive environment where they can do their experiments in the field of code analysis and transformation
- to offer the developers and programming experts an easy to learn and use language

Rascal will be the main language with which we will do the analysis and the re-factoring of our hand crafted PHP programs. More precisely we will use Rascal's concepts like data structures, pattern matching, switch and visit statements to fulfil our goal. Additionally for the parsing and the analysis of the PHP programs we will use the Rascal PHP analysis project, a project developed for analysing PHP software[8].

3.6 Parsing and ASTs

Parsing in computer science can be defined as the separation of a computer program into easily processed components which are analysed for correct syntax. Thereafter, these components are defined by attached tags. Parsing in Computer Science is usually done by the compiler which needs to parse a program

6 http://www.smarty.net/about_smarty
7 http://www.smarty.net/why_use
8 https://github.com/cwi-swat/php-analysis

before compiling it.

On the other hand abstract syntax trees (ASTs) are tree representations of data which are able to capture the essential structure of the input (a computer program for example) in a tree form, while skipping minor syntactic details. ASTs can be created by parsers and are used in source-code analysis and manipulation (SCAM) [5].

These two concepts were used practically for the creation of our prototype. Initially, with the help of Rascal and the PHP Analysis tool we parsed the code of our programs and subsequently we used the AST representation of the code to do the re-factoring. Below we can see how a simple PHP program is represented as an AST after its parsing with the PHP Analysis tool. This program takes two integers as variables and then assigns their sum in a third variable:

```
<?php

$x=20;
$y=30;
$z= $x + $y; ?>
```

Listing 3.1: PHP code to be parsed

Below we can see the AST of the above program:

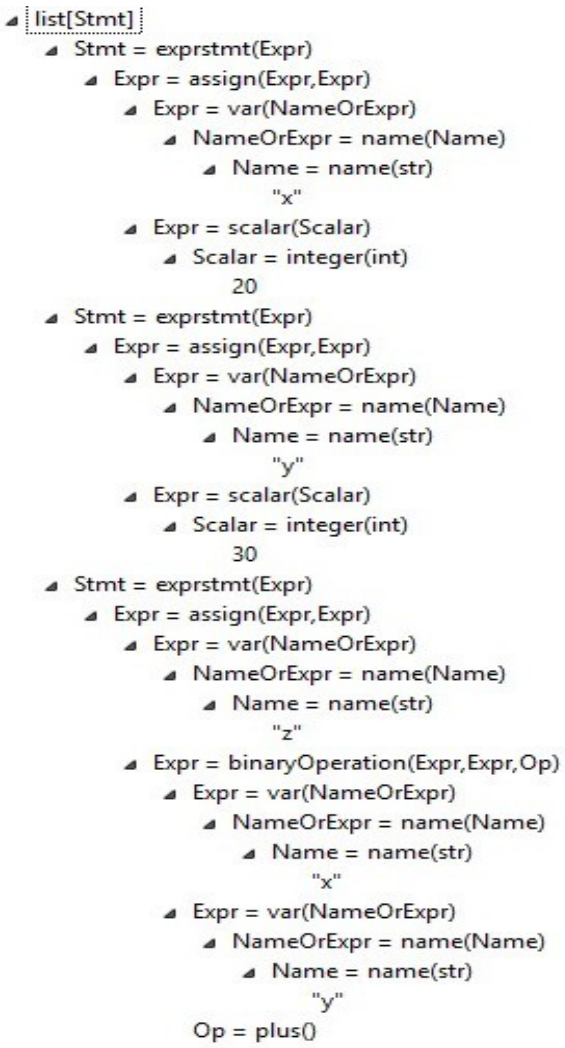

Figure 3.1: AST of the PHP program

3.7 XSS Attacks

XSS stands for Cross-Site Scripting and is a form of attack in web applications. It enables attackers to sneak malicious scripts into web-pages by exploiting possible vulnerabilities that might exist [7]. XSS can be distinguished between two types:

- **First-order or reflected XSS** – In this type of XSS, the attacker usually convince a simple user to "click" on a specially crafted URL address that contains malicious HTML/JavaScript code. By clicking this URL the user allows the malicious script to be executed into his browser. The result of this action could be the theft of sensitive user data by the attacker.
- **Second-order or persistent XSS –** The second type of XSS is the most dangerous one. In this type of XSS, the attacker can store his malicious code that has a form of a message, into a server that hosts a public forum. The malicious code can then be displayed permanently into this public forum posing a threat to the multiple users who visit it. Persistent XSS is considered more dangerous than reflected XSS because the attacker can harm multiple users without the need to trick them first.

Concerning to our thesis, Smarty supports the use of escape filters. These filters can prevent user inputs that contain scripts, preventing possible XSS attacks. Our tool will automatically transform the original PHP files that don't check for script import to web templates that use escape filters.

3.8 Related Work

Some general and interesting work on re-factoring can be found in Fabian's Bannwart and Peter's Muller work [9], where they present a technique that guarantees that the re-factoring is applied correctly. More precisely they divide each re-factoring into three stages: (1) Determination of the re-factoring's essential applicability conditions, (2) determination of the re-factoring's correctness conditions and (3) a formal proof that each application of the re-factoring preserves the external behaviour of the program, provided that the program satisfies the re-factoring's essential applicability conditions and correctness conditions.

In [10], the paper provides an extensive overview of existing research in the field of software re-factoring. This research is compared and discussed based on a number of different criteria. More specifically, the authors of this paper discuss about the following re-factoring activities:

- The identification of the software that should be re-factored.
- The determination of the suitable re-factoring.
- The guarantee that the applied re-factoring will not affect the software's behaviour.
- The application of the re-factoring.
- The evaluation of the re-factoring.

They also discuss about various re-factoring techniques and about previous work on program preservation proofs. The two previous papers, whereas they are not very focused on the topic of our research, they are noteworthy because of the methodologies that they provide.

Work which is more relevant to ours can be found in [8]. In this paper the research focuses on two tools which can automatically track and correct HTML generation errors in statements that print string literals (constant prints). Furthermore, in [11] the research focuses on a tool which automatically locates and fixes HTML validation errors in PHP-based Web applications. The two last papers face the same problem (repairing HTML generation errors in PHP applications). However they use different methods. In [8] they use instrumentation to map HTML output back in the program, while in [11] they use a heuristic algorithm

for the same purpose. In this thesis, we also deal with HTML generation in PHP applications. However, we are not trying to correct HTML generation errors, neither HTML validation errors with our tool. Instead, our goal is to re-factor the code into uses of the Smarty template engine.

Similar work can be also found in [14]. In this paper the authors discuss about fault localization in dynamic Web applications. Their tool automatically generates tests that expose failures, and then automatically localizes the faults responsible for those failures. Within the tested cases they are included those, where the output is written out using the print and echo statements (HTML generated through PHP).

Finally in [12] it is presented a framework which is mainly concerned with the decomposition of legacy Web applications to the MVC Architecture[9], by identifying software components to be transformed into Java objects. The MVC architecture consists of three parts, (1) the "model" (the code that handles the data), (2) the "view" (the code that displays that data) and (3) the "controller" (the code that handles the interaction between model and view and also functions outside the scope of either). In the topic of template systems, the main purpose of their use is the separation of concerns (the application from the presentation logic). Our task has many similarities with [12] however, the goal and the technologies that are used are different.

In general our project differs from previous work because it deals with code re-factoring into uses of template systems. The result of our research is a tool which automatically transforms HTML code (generated through PHP print-echo commands), into using the Smarty template engine. Successful results automatically mean easier application maintenance, faster code execution and more secure applications [13].

9 http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Chapter 4 - Research method

4.1 Overview

This chapter describes our research method. The procedure that we followed is:

1. We created simple PHP programs to test cases with the prototype.
2. We started creating our prototype, testing in parallel our hand-crafted PHP programs with it, from the simplest to the most complex ones.
3. In between these two first phases we wrote more PHP programs with more complex features and tested them with our prototype.
4. We validated our research by testing existing PHP applications.

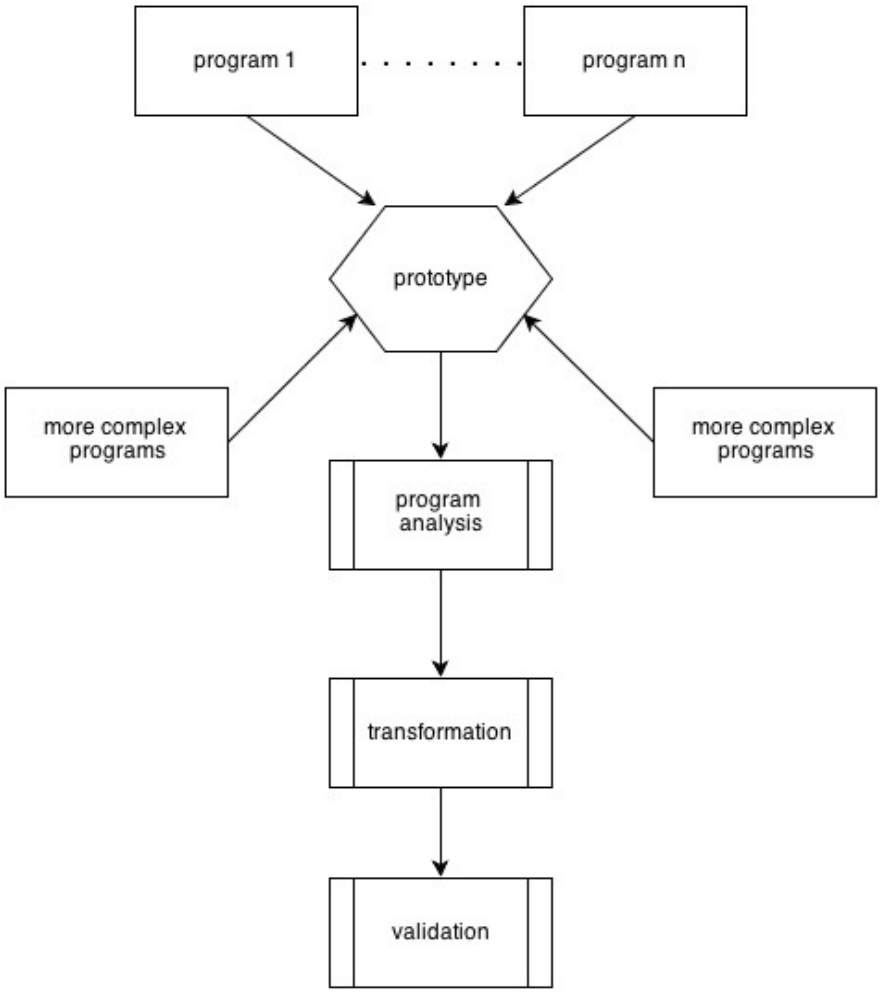

Figure 4.1: Research method

4.2 Writing simple PHP programs

The goal of this phase is to simulate with the help of our programs the desired behaviour of mixed PHP and HTML code. Later we will re-factor these programs with our prototype into a desired form that will keep up with our research question. We will start with simple cases and then we will move on to more challenging ones. By doing this, we will systematically add more complex features to our prototype while overcoming challenging tasks of our research.

4.3 Creating our prototype

We set out to implement our prototype with the hypothesis that the results will fulfil our research question, namely that is possible to automatically transform hand-crafted HTML into uses of template systems. For its implementation we used “Rascal - Meta Programming Language"[10] along with the Rascal PHP Analysis project[11], a project which focuses on developing tools for analysing PHP software. With the PHP Analysis project we parsed our hand-crafted PHP programs (see 4.1) and then restructured their code into code that uses the Smarty template engine[12].

4.4 Writing more complex PHP programs

From a certain point the implementation of our prototype progressed along with the creation of more complex PHP programs. We chose this layered way of the development for our prototype because of the assumptions that we made during its creation. More precisely our technique was to start with lots of assumptions about what our prototype can achieve and then weaken them as we proceeded. As an example, we started by assuming that all the HTML in our code is built into a single string, so we tried to handle these sorts of cases with our prototype. Then we increased the complexity by assuming that the HTML is built into multiple strings. Again we tried to adapt our prototype to handle this kind of cases and so on. In the end we didn't tackle the full complexity of PHP but we covered a sufficient number cases.

4.5 Validation of our research

We validated our research by testing with our prototype existing hand-crafted PHP applications and checking if the features that we had implemented were displayed correctly. We also examined whether the separation of concerns that our prototype offers can really help when creating or maintaining web applications.

10 http://www.rascal-mpl.org
11 https://github.com/cwi-swat/php-analysis
12 http://www.smarty.net/

Chapter 5 - The prototype

5.1 Overview

This chapter describes our prototype, the tool that automatically transforms PHP generated HTML code into uses of template systems. The size of the tool is 892 SLOC (source lines of code) and is written in Rascal. As it was mentioned in Chapter 4 we started to create our tool with the hypothesis that it will be capable to fulfil its goal. However, the complexity of the PHP language is big so we first started with simple cases and then we proceeded in more complex ones. During the implementation we faced many problems and some times we had to turn back and arrange the dependencies that were risen up. In the end we created a tool that does not cover the full complexity of PHP, but can successfully re-factor a sufficient number of cases. The procedure is always the same, parsing our PHP programs with the PHP analysis tool that we mentioned before, examining their AST and then trying to re-factor their code with Rascal.

5.2 The simplest case

Our first assumption and at the same time the first case that we dealt with, is that all the HTML that is generated in our code, is built in a single string. For this case we wrote a simple PHP program with an "echo" command that would generate some HTML. Our tool should traverse this program and search for this "echo" command. Then it should evaluate the content of this command. In this particular case it is a string literal. When the evaluation ends it should create a template and put the string inside it. At the same time it should create another PHP program that would call this template. The results before and after the transformation should be similar. In **Listing 5.1** we can see the code of the original program:

```
<?php
   echo "<html><head><title> printsAndEchos </title>
   </head><body> <h1> Say Hello! </h1></body></html>";
?>
```

Listing 5.1: the original PHP program – case 1

In **Listing 5.2** and **5.3** we can see the result of the transformation:

```
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$smarty->display('showTemplate.tpl');

?>
```

Listing 5.2: Transformation: PHP - case 1

```
{* Smarty *}
<html><head><title> printsAndEchos </title>
</head><body> <h1> Say Hello! </h1></body></html>
```

Listing 5.3: Transformation: Template - case 1

The “display” command in the transformed PHP program (**Listing 5.2**) indicates to our browser to show the content of the Smarty template.

5.3 Scattered “print-echo” commands

The second assumption is that the “print” or “echo” PHP commands are scattered around the program. That means that the HTML is built in multiple strings. In that case, we should take into account the control flow of the program and follow the previous process. For example for the program of the **Listing 5.4** according to the control flow, our prototype will traverse the program. At first it will find an "echo" command. It will evaluate its content and will find that is a string literal. This string literal will be put inside a template. Then it follows a "print" command. Its content will also be evaluated and the string literal will be put inside the same template. This procedure will go on until the end of the program. In **Listing 5.4** we can see the code of the original program:

```
<?php
   echo ("<html><head><title> printsAndEchos </title></head>");
   print "<body> <h1> Say Hello! </h1>";
   //......
?>

<?php
   //......
   echo "</body>";
   print "</html>";
?>
```

Listing 5.4: the original PHP program – case 2

In **Listing 5.5** and **5.6** we can see the result of the transformation:

```
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$smarty->display('showTemplate.tpl');

?>
```

Listing 5.5: Transformation: PHP - case 2

```
{* Smarty *}
<html><head><title> printsAndEchos </title>
</head><body> <h1> Say Hello! </h1></body></html>
```

Listing 5.6: Transformation: Template- case 2

At this point the re-factoring goes fine without any problems. These cases might be simple, but they showed us that we can have successful results.

5.4 Assigning and printing variables

The next level for our tool is to assign and print variables. The tool should assign the existing variables of the original program to values in the transformed PHP program. Then, like the previous cases, should evaluate the content of the "echo-print" commands. This time the result of the evaluation will be a combination of a variable and a string (for example $y= variable and
= string). Our tool should then correlate each variable with the assigned value and put these values inside the template. The same will happen with the string literals. In **Listing 5.7** we can see the code of the original program:

```
<?php

$y=5;
$z=-10;
$personal_info = array(
    'name' => 'Dimitris',
    'lname' => 'Kyritsis',
    'age' => '26'
);
$sum=$y+$z;

echo "$y<br>";
echo "$z<br>";
echo "$personal_info[name]<br>";
print "$sum";

?>
```

Listing 5.7: the original PHP program – case 3

In **Listing 5.8** and **5.9** we can see the result of the transformation:

```php
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$y = 5;$smarty->assign('val1',$y);
$z = -10;$smarty->assign('val2',$z);
$personal_info = array('name' => 'Dimitris','lname' => 'Kyritsis','age' =>
'26');$smarty->assign('val3',$personal_info);
$sum = $y + $z;$smarty->assign('val4',$sum);
$smarty->display('showTemplate.tpl');
?>
```

Listing 5.8: Transformation: PHP – case 3

```
{* Smarty *}
{$val1}  <br>  {$val2}  <br>  {$val3.name}  <br>  {$val4}
```

Listing 5.9: Transformation: Template - case 3

In the Listing 5.9 we can see that the original variables are assigned to values and then are passed to the template (Listing 5.8). These values are used as "holes" that can pass information from the PHP program to the template. Our tool is able to print many kinds of variables. In the previous program we can see numbers(positive and negative), strings and simple arrays(with keys and values). Furthermore it can handle variables from external sources (using $_GET and $_POST[13]) booleans and arrays with variables as keys. The cases that it cannot handle are variable functions[14], variable variables[15], recursive, dynamic and multi-dimensional arrays[16]. Further work is needed for our tool to be able to handle these more complex cases.

5.5 Variables : Type juggling and references

The fourth case that we dealt with was type juggling[17] and variable assignment by reference[18]. In these two cases after we added the necessary code for our tool that enabled it to support these two operations, we let the control flow of the program to do the remaining work (like the previous case: **5.4- Assigning and printing variables**). In **Listing 5.10** we can see the code of the original program:

13 http://www.php.net/manual/en/language.variables.external.php
14 http://php.net/manual/en/functions.variable-functions.php
15 http://php.net/manual/en/language.variables.variable.php
16 http://php.net/manual/en/language.types.array.php
17 http://php.net/manual/en/language.types.type-juggling.php
18 http://www.php.net/manual/en/language.variables.basics.php

```
<html>
<head><title> Type Juggling and Ref assignments </title></head>
<body>

<?php
$name = "Dimitris";
echo "The old name is $name";
$name = 'George';
$newName=array('Joe', 'Jack');
$newname=&$newName;
echo "<br>New name is $name or $newname[0]";
$name += $_GET['name'];
print "<br>After type juggling \$name: ";
echo "$name";

?>
</body></html>
```

Listing 5.10: The original PHP program- case 4

In **Listing 5.11** and **5.12** we can see the result of the transformation:

```
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$name = 'Dimitris';$smarty->assign('val1',$name);
$name = 'George';$smarty->assign('val2',$name);
$newName = array('Joe','Jack');$smarty->assign('val3',$newName);
$newname = $newName;$smarty->assignByRef('val4',$newname);
$name += $_GET['name'];$smarty->assign('val5',$name);
$smarty->display('showTemplate.tpl');

?>
```

Listing 5.11: Transformation: PHP - case 4

```
{* Smarty *}
The old name is  {$val1}  <br>New name is
{$val2}  or  {$val4.0} <br> After type juggling
$name:  {$val5|escape:'htmlall'}
```

Listing 5.12: Transformation: Template - case 4

Above we can see that the first variable takes as a value a string literal and subsequently this value changes to another string literal. We have also an array assignment and a variable which is a reference to this array. Finally the first variable ($name) changes again and a value is added to it dynamically through a URL (we will discuss about this, as well as the escape:'htmlall' that we can see inside the template in a next chapter). The procedure here is like of the previous chapter, except that now our tool instead of outputting the “assign” command in the transformed PHP will output “assignByRef” when the variable is a reference. It will also use “+=” instead of “=” to support type juggling.

For the case of type juggling, when the variable $name has the value 'George', if we pass through the URL something like: *http://..../showTemplate.php?name=44* the value 'George' will be replaced by the number 44 (the value from a string is now an integer). However if we try to pass something like: *http://..../showTemplate.php?name=Jim* the value 'George' will be replaced by the number 0. Back to our transformation, the "print-echo" content is always evaluated according to the control flow like before. As we mentioned in **chapter 5.4** variable functions, variables variables and recursive and multi-dimensional arrays are not supported.

5.6 Printing mixed string literals with variables and the case of concatenation

After dealing with generated HTML of a form of single variables or string literals, it is time to take care of more complex cases. For example, as we saw in **5.4-Assigning and printing variables** the content of the "print- echo" commands was a mixture of a variable and a string. Our prototype, when evaluating the content of the above commands, distinguishes each case and treats them accordingly. Another usual case is when the content of the "print-echo" commands uses concatenation. Concatenation belongs to binary operations, thus we have to evaluate the left and right part of the operation.

For example if we have this piece of code: *"Hello" . $name,* we should evaluate the left side of the "." (*"Hello"*) and then the right side (*$name*). However the situation is not always so simple. In more complex cases like the following: *"Hello" . $name . "and $name2",* the left side of the binary operation is a binary operation (*"Hello" . $name*) and the right side is a mixed string with a variable (*"and $name2"*). Our tool during the evaluation of the left side it uses recursion and evaluates anew the new binary operation (*"Hello" . $name*). Now the left side of the concatenation is a string and the right side is a variable. These two parts are then handled as we described in the previous chapters. When it finishes with the left side our tool will evaluate the right side (*"and $name2"*), which is a string mixed with a variable. The contents will be put inside the template and the re-factoring will end successfully. In **Listing 5.13** we can see the code of the original program:

```
<html>

<head>
<title>Encapsed case and concatenation</title>
</head>
<body>
<?php

$lang= "PHP";
$lang2= "Rascal";
echo  "<h1>I love $lang and $lang2!</h1><br>";
$lang= "C";
print "<b>I love " . $lang . " and $lang2</b>";

?>
</body>
</html>
```

Listing 5.13: the original PHP program – case 5

In **Listing 5.14** and **5.15** we can see the result of the transformation:

```php
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$lang = 'PHP';$smarty->assign('val1',$lang);
$lang2 = 'Rascal';$smarty->assign('val2',$lang2);
$lang = 'C';$smarty->assign('val3',$lang);
$smarty->display('showTemplate.tpl');
?>
```

Listing 5.14: Transformation: PHP - case 5

```
{* Smarty *}
<h1>I love  {$val1}  and  {$val2} !</h1><br>
<b>I love  {$val3}  and  {$val2} </b>
```

Listing 5.15: Transformation: Template- case 5

5.7 Dealing with security issues: the $_GET and $_POST variable

In **Chapter 3** we gave some information about XSS attacks. A usual way for attackers to do an XSS attack is by exploiting dynamic fields in HTML by using malicious scripts. However, PHP has ways to prevent these attacks. One of these ways is the htmlspecialchars() function[19]. The use of this function allows us to convert special characters to HTML entities. So if an attacker tries to inject a malicious script as a part of a string the use of this function will return the string with the script as a part of it. For example the code: `print htmlspecialchars("<script>alert('hey')</script>");` , will output the string: `<script>alert('hey')</script>` , and not a pop up window with the message `'hey'`. In Smarty we can achieve this feature by using escape filters[20].

A way to pass values dynamically to a PHP program is by using the predefined $_GET or $_POST variable. This way a user can pass values through a form and even from a URL. However, in cases where the developer hasn't used the htmlspecialchars() function, when echoing a string that possibly contains a $_GET or a $_POST variable, their use might prove really dangerous. An attacker can pass a script through the URL or a form and exploit a web application. Our tool will deal with this kind of cases by attaching an escape filter to any $_GET or $_POST variables inside the created template. In **Listing 5.16** we can see the code of the original program using a $_GET variable:

```php
<?php

$name = $_GET['name'];
$age = $_GET['age'];
echo "Welcome $name. You are $age years old <br>";

?>
```

Listing 5.16: The original PHP program- case 6

19 http://php.net/manual/en/function.htmlspecialchars.php
20 http://www.smarty.net/docsv2/en/language.modifier.escape.tpl

In **Listing 5.17** and **5.18** we can see the result of the transformation:

```
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$name = $_GET['name'];$smarty->assign('val1',$name);
$age = $_GET['age'];$smarty->assign('val2',$age);
$smarty->display('showTemplate.tpl');
?>
```

Listing 5.17: Transformation: PHP - case 6

```
{* Smarty *}
Welcome {$val1|escape:'htmlall'}.
You are {$val2|escape:'htmlall'} years old <br>
```

Listing 5.18: Transformation: Template - case 6

If we type in our browser the location of the new PHP file with this extension: `http://../showTemplate.php?name=Jim&age=26<script>alert('attacked')</script>`, the message will be: `Welcome Jim. You are 26<script>alert('attacked')</script> years old`. Our efforts to pass a script through the URL will fail due to the escape filter (The pop up window with the message `'attacked'` will not show up).

At this point we will start describing more complex cases. More specifically we will discuss about how our tool can deal with loops. It was a difficult task as we faced many boundaries and limitations that should be overcome by future work.

5.8 The if statement

The "if" statement is one of the most important features of PHP. We are also interested in it since it is essential for the proper operation of our tool. More specificaly, there might exist "echo-print" commands inside the body of the "if" statement that we have to put inside our templates. If so, we should put the "if"- "elseif"- "else" conditions inside the template too. However, sometimes there are no "echo-print" commands inside the "if" body. The algorithm of our tool checks the body of the "if" statement (which might also be another "if" statement or a loop in general) and if it finds "echo-print" commands it puts the conditions and the results into the template. If not it ignores the whole statement.

Inside the template file, the condition of the "if" statement can contain constants, variables, binary operations, and unary operations. It cannot contain any functions as we didin't support them in our tool. The body of the 'if" inside the template will be the content of the "echo-print" commands as we described them in the previous chapters or loops (our tool can transform the "foreach" and the "while" loop). The same applies in the case of the "elseif". Finally for the "else" the procedure is the same except that there is not any condition. In **Listing 5.19** we can see the code of the original program:

```php
<?php

$i= '0-50';
$j= '51-100';
$z= '101-150';
$x=rand(0, 150);

if ($x <= 50)
    echo "<b>The number is $x : and exists between $i</b><br>";
elseif  ( $x > 50 && $x <= 100)
    echo "<b> The number is $x : and exists between $j </b>";
else
    echo "<b> The number is $x : and exists between $z </b>";
?>
```

Listing 5.19: The original PHP program- case 7

In **Listing 5.20** and **5.21** we can see the result of the transformation:

```php
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$i = '0-50';$smarty->assign('val1',$i);
$j = '51-100';$smarty->assign('val2',$j);
$z = '101-150';$smarty->assign('val3',$z);
$x = rand(0,150);$smarty->assign('val4',$x);
$smarty->display('showTemplate.tpl');
?>
```

Listing 5.20: Transformation: PHP - case 7

```
{* Smarty *}
{if {$val4} < 50}<b>The number is {$val4} : and exists between {$val1}
</b><br>{elseif {$val4} > 50 && {$val4} < 100}<b> The number is {$val4} : and
exists between {$val2} </b> {else} <b> The number is {$val4} : and exists
between {$val3} </b> {/if}
```

Listing 5.21: Transformation: Template - case 7

The expressions inside the “if-elseif”condition could be more complex containing for example many binary operations. Also, as we mentioned before, inside the “if-elseif-else” body could be nested infinite “if” statements like in common PHP programs.

5.9 The foreach loop

The “foreach” loop is another important feature of PHP as it provides us with an easy way to iterate over arrays. Like the previous case of the “if” statement, inside the body of the “foreach” loop there might also exist "echo-print" commands which we need to put inside our templates. Our tool is able to handle both the below syntaxes of this loop:

- `foreach (array_expression as $value)`
  `statement`

- `foreach (array_expression as $key => $value)`
  `statement`

However it cannot trasform code which contain multi-dimensional or dynamic arrays in the "foreach" condition.

The procedure that our tool follows to do the transformation of a "foreach" loop is the same as with the "if" statement. It checks the body of the "foreach" loop and if it finds "echo-print" commands, it puts the condition and the results into the template. If not it ignores the whole statement. Our tool can successfully handle cases when other loops or statements ("while-foreach" loops, "if" statement) are inside the body of the "foreach" loop. In **Listing 5.22** we can see the code of the original program:

```php
<?php

$a = array(1, 2, 3);

foreach ($a as $v) {
    echo "<li>Current value of \$a: $v.</li>";
}
print "<br>";
$a = array(
    "one" => 1,
    "two" => 2,
    "three" => 3
);

foreach ($a as $k => $v) {
    echo "<li>\$a[$k] => $v.</li>";
}

?>
```

Listing 5.22: the original PHP program – case 8

In **Listing 5.23** and **5.24** we can see the result of the transformation:

```php
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$a = array(1,2,3);$smarty->assign('val1',$a);
$a = array('one' => 1,'two' => 2,'three' => 3);$smarty->assign('val3',$a);
$smarty->display('showTemplate.tpl');
?>
```

Listing 5.23: Transformation: PHP - case 8

```
{* Smarty *}
{foreach from=$val1 item=val2} <li>Current value of $a: {$val2} .</li>
{/foreach}
<br> {foreach from=$val3 key=val5 item=val4} <li>$a[ {$val5} ] => {$val4} .</li>
{/foreach}
```

Listing 5.24: Transformation: Template - case 8

A case that it is not transformed by our prototype is when there is an increment or a decrement of a variable (for example `i++, i--`) inside the body of a “foreach” loop. If for example we have this sample code :

```
$a = array(1, 2, 3);
$i = 0;
foreach ($a as $v) {
    echo "\$a[$i] => $v.\n";
    $i++;
}
```

the result in our browser after the transformation will be the following:

```
$a[ 0 ] => 1 . $a[ 0 ] => 2 . $a[ 0 ] => 3 .
```

while it should be:

```
$a[ 0 ] => 1 . $a[ 1 ] => 2 . $a[ 2 ] => 3 .
```

This is happening because we haven't adjusted our tool to transform cases with increments or decrements inside a “foreach” loop. Future work is needed for this.

5.10 The while loop

The last case that our prototype will deal with is the “while” loop. Equally important than the two previous cases the “while” loop is very useful in PHP. However its representation in Smarty puts limitations on what we can do with it. The procedure of its transformation is almost similar to the “if” statement, since they have the same syntax:

- ```
  while (expr)
       statement
  ```

The limitations are also the same (see previous chapters). Another problem that we had to overcome was the correct transformation of increments and decrements which we did with success. However the representation of the “while” loop in Smarty brought us face to face with another limitation. For example this sample code:

```
$foo=3;
while($foo > 0){
      print "number[$foo] : $foo ";
      $foo--;
}
```

will be represented like this inside a Smarty template:

```
{while $val1  > 0} number[{$val1--}] : {$val1--} {/while}
```
```

and the corresponding results in the browser will be:

- Without the use of templates : `number[3] : 3 number[2] : 2 number[1] : 1`

and:

- With the use of templates : `number[3] : 2 number[1] : 0`

So a limitation when we re-factor code with “while” loops with our tool is that we cannot “echo-print” the increments or decrements more than once inside the body of the loop. Another limitation is that the value of the condition can only change because of increments or decrements ($i++, $i--) and not by other cases ($i*2 for example). Future work is needed to prevent this limitation. In **Listing 5.25** we can see the code of the original program:

```
<?php

$i=1;
$x=rand(51,100);

while($i <= 10){
    echo"$i : $x<br>";
    $i++;
    $x--;
}
?>
```

Listing 5.25: the original PHP program – case 9

In **Listing 5.26** and **5.27** we can see the result of the transformation:

```
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$i = 1;$smarty->assign('val1',$i);
$x = rand(51,100);$smarty->assign('val2',$x);
$smarty->display('showTemplate.tpl');
?>
```

Listing 5.26: Transformation: PHP - case 9

```
{* Smarty *}
{while $val1 <= 10} {$val1++} : {$val2--} <br> {/while}
```

Listing 5.27: Transformation: Template - case 9

5.11 Summary

The table below provides a list of PHP features and indicate how well our tool can handle them. It can be used as a summary of **Chapter 5**:

PHP features	Handled	Semi-handled	Unhandled
Types	Booleans, Integers,Floating point numbers, Strings, NULL, Type juggling	Arrays (see **Chapter 5.4 - Assigning and printing variables**)	Objects, Resources, Callbacks, Pseudo-types
Variables	Basic variables (handled types), Variables assigned by reference, Variables from external sources ($_GET, $_POST)	Basic variables(semi-handled types), Predefined variables ($_GET, $_POST are handled)	Variable variables, Variable functions
Constants	No	No	Yes
Operators	Arithmetic operators, Assignment Operators, Comparison Operators, Incrementing/ Decrementing Operators, Logical Operators, String Operators, Array Operators	No	Bitwise Operators, Error Control Operators, Execution Operators, Type Operators
Control Structures	if/elseif/else	while (see **Chapter 5.10 - The while loop**), foreach (see **Chapter 5.9 - The foreach loop**)	do-while (not supported in Smarty), for, break, continue, switch (not supported in Smarty), return
Functions	No	No	Yes
Classes and Objects	No	No	Yes

Table 5.1: List of PHP features which are handled, semi-handled or unhandled by our tool

These features were taken from the PHP website[21]. They don't represent the whole complexity of PHP, however they are the most important features that in our opinion should be handled by our tool. More specifically, in the category Handled we can find the features that are completely handled by our tool (we have a successful re-factoring). Semi-handled features are those which are not completely handled. For example, in the category Variables we can see that Arrays are semi-handled. If we look back in **Chapter 5.4 - Assigning and printing variables**, we can see that our prototype is able to handle simple arrays(with keys and values) and arrays with variables as keys. However it cannot handle dynamic and multi-dimensional arrays. Finally, the category Unhandled describes these features that are not handled at all by our tool. Future work is needed for these features. Successful treatment of the two last categories will automatically mean a more useful tool.

21 http://www.php.net/manual/en/langref.php

Chapter 6 - Evaluation and results

6.1 Overview

In this chapter we will make the evaluation of our research. As a basis for the evaluation we have the two following questions:

- Are the transformations made by our tool semantics-preserving?
- Does the separation of concerns (business logic from presentation logic) really facilitate the maintenance of web-applications?

By answering the first question (or at least to some extent), we will show that our prototype is successful and is possible to have more success after some future work. Thus, we will have answered successfully our research question. A satisfactory answer of the second question will justify the creation of our prototype. For this purpose, we will test an existing application called SchoolMate[22] with our tool, which is a PHP/MySQL solution for elementary, middle and high schools.

6.2 Semantics-preserving transformations, the method

The method which we will use to answer the first question is the following:

1. Initially, our tool will search for the 63 PHP files which form SchoolMate inside their directory.
2. Thereafter, it will do the re-factoring for each one of them, putting the generated files inside the root directory of our server.
3. Then the evaluation process will take place. More specifically we will test the original and the transformed programs for similarity of their results. To achieve this we will compare the resulted web-pages' source of both original and transformed programs with diff[23], a file comparison utility that outputs the differences between two files. The output of the comparison will be the similar and the different lines of the transformed file comparing to the original one.
4. Finally, we will discuss the results of this comparison.

6.3 Semantics-preserving transformations, the results

Table 6.1 below shows the similarity of the original and the transformed programs:

22 http://sourceforge.net/projects/schoolmate
23 http://en.wikipedia.org/wiki/Diff

Original program	*Template*	*Total lines*	*Differences*	*Similarity*
AddUser.php	temp0.tpl	68	1	98.50%
ViewGrades.php	temp5.tpl	38	0	100.00%
ViewAnnouncements.php	temp9.tpl	33	0	100.00%
ManageAnnouncements.php	temp10.tpl	99	0	100.00%
ManageUsers.php	temp11.tpl	100	0	100.00%
Footer.php	temp12.tpl	20	2	85.00%
Main.php	temp17.tpl	54	29	54.00%
ManageTeachers.php	temp21.tpl	101	0	100.00%
AddAnnouncements.php	temp29.tpl	40	3	92.50%
ParentViewStudents.php	temp33.tpl	29	2	93.00%
AddAttendance.php	temp36.tpl	46	5	89.00%
MakeTop.php	temp39.tpl	26	1	96.00%
ViewStudents.php	temp40.tpl	34	3	91.00%
ManageParents.php	temp43.tpl	104	3	97.00%
ManageTerms.php	temp44.tpl	103	3	97.00%
VisualizeClasses.php	temp50.tpl	47	3	93.50%
ManageSemesters.php	temp53.tpl	106	3	97.00%
AddAssignment.php	temp59.tpl	45	4	91.00%
AddTerm.php	temp62.tpl	40	3	92.50%

Table 6.1: Similarity of original and transformed programs

In the above table we can see that the results show similarity mainly up to 90.00%. The differences were exclusively spaces or empty lines. We can see that there is a big difference in the files Main.php and temp17.tpl. The reason is that the PHP file has a function include() in it that our tool cannot transform. However, we manually created a new file where we copied the included file and then transformed it with our tool. This time there were only 3 differences (2 differences because of a white-space and 1 because of an empty line) and the similarity between the two programs was 94.00% (51 lines of code). Few other templates also have similar problems so we make the comparison only in code that both programs (original and transformed) contain. Another notice is that there might be approximately 2-3 less differences in some of these programs. The reason is that few original programs had some problems with the database so they didn't show small blocks of code. However they were containing code that in previous comparisons displayed differences, so we added them manually in the present cases. **Table 6.2** shows the programs that we couldn't do the comparison:

Original program	*Template*	*Problem*
ParentMain.php	temp1.tpl	Both programs couldn't show up in the browser
GradeReport.php	temp2.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
AddParent.php	temp3.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
EditSemester.php	temp4.tpl	Both programs couldn't show up in the browser
ManageGrades.php	temp6.tpl	The content of the template wasn't correct – Our tool

		didn't do the transformation properly
Login.php	temp7.tpl	Both programs couldn't show up in the browser
AddTeacher.php	temp8.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
AddStudent.php	temp13.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
ViewAssignments.php	temp.14.tpl	The original program couldn't show up in the browser
Registration.php	temp15.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
ManageAssignments.php	temp16.tpl	The original program couldn't show up in the browser
StudentViewCourses.php	temp18.tpl	Both programs couldn't show up in the browser
PointsReport.php	temp19.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
EditAssignment.php	temp20.tpl	Both programs couldn't show up in the browser
EditTerm.php	temp22.tpl	Both programs couldn't show up in the browser
Registration.php	temp15.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
ManageAssignments.php	temp16.tpl	The original program couldn't show up in the browser
StudentViewCourses.php	temp18.tpl	Both programs couldn't show up in the browser
PointsReport.php	temp19.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
EditAssignment.php	temp20.tpl	Both programs couldn't show up in the browser
EditTerm.php	temp22.tpl	Both programs couldn't show up in the browser
EditUser.php	temp23.tpl	Both programs couldn't show up in the browser
DBFunctions.php	temp24.tpl	Both programs couldn't show up in the browser
EditTeacher.php	temp26.tpl	Both programs couldn't show up in the browser
Header.php	temp27.tpl	The original program couldn't show up in the browser
ViewCourses.php	temp28.tpl	Both programs couldn't show up in the browser
Index.php	temp30.tpl	Both programs couldn't show up in the browser
EditStudent.php	temp31.tpl	Both programs couldn't show up in the browser
DeleteFunctions.php	temp32.tpl	Both programs couldn't show up in the browser
ReportCards.php	temp34.tpl	Both programs couldn't show up in the browser
Connect.php	temp35.tpl	Both programs couldn't show up in the browser
ParentViewCourses.php	temp37.tpl	Both programs couldn't show up in the browser
AdminMain.php	temp38.tpl	The original program couldn't show up in the browser
ManageStudents.php	temp41.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
DeficiencyReport.php	temp42.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
EditAnnouncements.php	temp45.tpl	Both programs couldn't show up in the browser

ManageClasses.php	temp46.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
AddClass.php	temp47.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
TeacherMain.php	temp48.tpl	Both programs couldn't show up in the browser
ClassSettings.php	temp49.tpl	The original program couldn't show up in the browser
AddSemester.php	temp51.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
ViewClassSettings.php	temp52.tpl	The original program couldn't show up in the browser
ManageAttendance.php	temp54.tpl	The content of the template wasn't correct – Our tool didn't do the transformation properly
StudentMain.php	temp55.tpl	Both programs couldn't show up in the browser
ValidateLogin.php	temp56.tpl	Both programs couldn't show up in the browser
ManageSchoolInfo.php	temp57.tpl	The original program couldn't show up in the browser
EditClass.php	temp58.tpl	Both programs couldn't show up in the browser
VisualizeRegistration.php	temp60.tpl	The original program couldn't show up in the browser
EditGrade.php	temp61.tpl	Both programs couldn't show up in the browser

Table 6.2: Programs with no available comparison

As we can see many programs had problems and we were not able to compare them. In general there are 3 reasons why the comparison is not available.

1. **The content of the template wasn't correct – Our tool didn't do the transformation properly:** In this case, inside the original PHP programs there was code that our tool couldn't handle (we have discussed about such cases in **Chapter 5**). Future work is needed to solve these problems.

2. **The original program couldn't show up in the browser:** In this case the PHP programs couldn't show up in the browser due to problems with the database. If this problem had been fixed we could compare 9 more cases.

3. **Both programs couldn't show up in the browser:** In this case both the PHP programs and the templates had the above problems so we couldn't compare them.

In general, although we didn't have the opportunity to compare the majority of the programs, for those that we made it, the results were satisfactory. The small differences don't affect the operation, especially since these differences are only white-spaces and empty lines, something that can be changed easily by the designer by hand. A serious problem would be if for example the assignment of the variables was erroneous. Since this is not happening and all the important information appear correctly in the templates, our tool proves to be useful, thus the first question of the evaluation is answered successfully.

6.4 Separation of concerns, the method

In the previous chapters of this thesis we stressed how important is the separation of the business logic from the presentation in web applications. That was a major reason why we created our prototype and now it is time to argue if this separation of concerns really helps designers and developers when maintaining or updating code. The method that we will follow for this task is the following:

1. We will pick a simple program from the application that we tested in this chapter and we will compare it with the programs that were the results of its transformation procedure.

2. Thereafter we will create a simple maintenance scenario about the same programs and we will give our opinion about which of these two cases was easier for us to maintain.

6.5 Separation of concerns, the results

The program that we will test is the ParentViewStudents.php. The code is below:

```
<?php
$query = mysql_query("SELECT parentid, fname, lname FROM parents WHERE userid =
$_SESSION[userid]");
$parent = mysql_fetch_row($query);
$parentid = $parent[0];

print("
 <h1>Students of $parent[1] $parent[2]</h1>
 <br>
 <table align='center' width='300' cellspacing='0' cellpadding='0' border='0'>
 <tr>
 <td>
 <form name='classes' action='./index.php' method='POST'>");

print("
  <br>
  <table cellspacing='0' width='300' align='center' class='dynamiclist'>
  <tr class='header'>
   <th>Student Name</th>
  </tr>");
print("  </table>
  <br />
  <input type='hidden' name='student' value='' />
  <input type='hidden' name='page2' value='$page2' />
  <input type='hidden' name='logout' />
  <input type='hidden' name='page' value='$page' />
 </form>
 </td>
 </tr>
 </table>
 <table width='520' border=0 cellspacing=0 cellpadding=0 height=1>
  <tr>
   <td valign='top'>
   <empty>
   </td>
  </tr>
 </table>
 ");
?>
```

Listing 6.1: ParentViewStudents.php

In **Listing 6.2** and **6.3** we can see the result of the transformation:

```php
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';

$query= mysql_query('SELECT parentid, fname, lname FROM parents
WHERE userid= '.$_SESSION['userid']);$smarty->assign('val1',$query);
$parent= mysql_fetch_row($query);$smarty->assign('val2',$parent);
$parentid= $parent[0];$smarty->assign('val3',$parentid);
$smarty->display('temp33.tpl');
?>
```

Listing 6.2: temp33.php

```
{* Smarty *}

 <h1>Students of  {$val2.1}   {$val2.2} </h1>
 <br>
 <table align='center' width='300' cellspacing='0' cellpadding='0' border='0'>
 <tr>
 <td>
 <form name='classes' action='./index.php' method='POST'>
  <br>
  <table cellspacing='0' width='300' align='center' class='dynamiclist'>
  <tr class='header'>
   <th>Student Name</th>
  </tr>   </table>
  <br />
  <input type='hidden' name='student' value='' />
  <input type='hidden' name='page2' value=' ' />
  <input type='hidden' name='logout' />
  <input type='hidden' name='page' value=' ' />
 </form>
 </td>
 </tr>
 </table>
 <table width='520' border=0 cellspacing=0 cellpadding=0 height=1>
  <tr>
   <td valign='top'>
   <empty>
   </td>
  </tr>
 </table>
```

Listing 6.3: temp33.tpl

First of all some parts of the code of the ParentViewStudents.php couldn't be transformed by our tool, so we removed them. Another small note about the code that is displayed in the temp33.tpl, is that there are 2 differences from the code of ParentViewStudents.php: `<input type='hidden' name='page2' value='$page2' />` is transformed to `<input type='hidden' name='page2' value=' ' />` and `<input type='hidden' name='page2' value='$page' />` to `<input type='hidden' name='page2' value=' ' />`. As we can see the variables $page and $page2 are missing from the template. The reason is that these two variables are not defined anywhere and there is not any value assigned

to them (unlike the variable $parent), so they are not passed into the template.

After this short explanation we can pass to the main topic. In the first program (**Listing 6.1**) we can see that the variable assignment and the “print” commands that generate the HTML are mixed together. The application logic programmer and the layout designer work in the same file and they can easily interfere in each other's work causing trouble, especially if the program is more complex and there are loops and functions inside its body. In the second case (**Listings 6.2** and **6.3**) the variables are assigned in the temp33.php program, which in turn calls the temp33.tpl, that is responsible for the display. We can see for example that the variable $parent passes as 'val2' into the template. The designer who is responsible for the layout is not able to change this variable as it is predefined by the programmer in the PHP file. However, he can put this variable wherever he wants inside the template and for multiple times and in the same time he can deal with the HTML that is responsible for the layout. Respectively the programmer can only work with the temp33.php file and he is not interfering with the template.

A big advantage of the second case (template system) is that it gives the possibility to different people to work separately on design and code at the same time. In our case, the programmer for example can add variables to the transformed PHP file, while at the same time the designer can add a menu at the template file. In general this separation of concerns helps both the programmer and the designer not to move beyond their limits (and their knowledge). Let us assume now that our original program is the one of the **Listing 6.4** and that we want to change some parts of the HTML:

```
<?php
$query = mysql_query("SELECT parentid, fname, lname FROM parents WHERE userid =
$_SESSION[userid]");
$parent = mysql_fetch_row($query);
$parentid = $parent[0];

print("
 <h1>Students of $parent[1] $parent[2]</h1>
 <br>
 <table align='center' width='300' cellspacing='0' cellpadding='0' border='0'>
 <tr>
 <td>
 <form name='classes' action='./index.php' method='POST'>");

$newQuery = mysql_query("SELECT studentid, fname, lname FROM students WHERE
userid = $_SESSION[userid]");

print("
  <br>
  <table cellspacing='0' width='300' align='center' class='dynamiclist'>
  <tr class='header'>
   <th>Student Name</th>
  </tr>");
$student = mysql_fetch_row($newQuery);
 print("  </table>
  <br />
  <input type='hidden' name='student' value='' />
  <input type='hidden' name='page2' value='$page2' />
  <input type='hidden' name='logout' />
  <input type='hidden' name='page' value='$page' />
 </form>
 </td>
 </tr>
 </table>
");
$studentid = $student[0];
print(" <table width='520' border=0 cellspacing=0 cellpadding=0 height=1>
  <tr>
   <td valign='top'>
   <empty>
   </td>
  </tr>
 </table>
 ");
?>
```

Listing 6.4: ParentViewStudents(2).php

```php
<?php
define('SMARTY_DIR', 'C:/Smarty/libs/Smarty-3.1.13/libs/');

    require_once(SMARTY_DIR . 'Smarty.class.php');

    $smarty = new Smarty();

    $smarty->template_dir = 'C:/xampp/htdocs/smarty/templates';
    $smarty->compile_dir  = 'C:/xampp/htdocs/smarty/templates_c';
    $smarty->config_dir   = 'C:/xampp/htdocs/smarty/configs/';
    $smarty->cache_dir    = 'C:/xampp/htdocs/smarty/cache/';


$query = mysql_query('SELECT parentid, fname, lname FROM parents WHERE userid =
'.$_SESSION['userid']);$smarty->assign('val1',$query);

$parent = mysql_fetch_row($query);$smarty->assign('val2',$parent);

$parentid = $parent[0];$smarty->assign('val3',$parentid);

$newQuery = mysql_query('SELECT studentid, fname, lname FROM students WHERE
userid = '.$_SESSION['userid']);$smarty->assign('val4',$newQuery);

$student = mysql_fetch_row($query);$smarty->assign('val5',$student);

$studentid = $student[0];$smarty->assign('val6',$studentid);

$smarty->display('showTemplate.tpl');

?>
```

Listing 6.5: temp33(2).php

```
{* Smarty *}

 <h1>Students of  {$val2.1}   {$val2.2} </h1>
 <br>
 <table align='center' width='300' cellspacing='0' cellpadding='0' border='0'>
 <tr>
 <td>
 <form name='classes' action='./index.php' method='POST'>
  <br>
  <table cellspacing='0' width='300' align='center' class='dynamiclist'>
  <tr class='header'>
   <th>Student Name</th>
  </tr>   </table>
  <br />
  <input type='hidden' name='student' value='' />
  <input type='hidden' name='page2' value=' ' />
  <input type='hidden' name='logout' />
  <input type='hidden' name='page' value=' ' />
 </form>
 </td>
 </tr>
 </table>
  <table width='520' border=0 cellspacing=0 cellpadding=0 height=1>
  <tr>
   <td valign='top'>
   <empty>
   </td>
  </tr>
 </table>
```

Listing 6.6: temp33(2).tpl

It is obvious that it is way easier for a designer to make changes in the template shown in **Listing 6.6** rather than in the original program (**Listing 6.4**). The person can act independently without thinking about the variables and the problems that might rise in case of an incorrect deletion. He is also not loosing valuable time by constantly discussing with the programmer about the possible changes in the code, since they work in different files.

At the same time the programmer can add or delete variables without further thought about where exactly to put them (**Listing 6.5**) inside the program. Both developers can work in parallel, in different files so there are no delays and no inconveniences between them. The only contact they need to have is when the programmer tells the designer the name that he has given to a new variable. Thereafter the designer can use it wherever and whenever he wants, without the possibility to change its value.

6.6 Conclusion and Future Work

After the previous examples it is clear that template systems can significantly help when dealing with web applications' maintenance. Consequently our prototype is a useful tool with lot of potential. Hopefully after some future work it will be able to deal with a greater variety of cases. Back in **Chapter 5** we discussed about the potential and the weaknesses of our prototype, while in **Table 5.1: List of PHP features which are handled, semi-handled or unhandled by our tool**, we gave a summary of important PHP features and the way they are handled by our tool. This table is a valuable advisor for the future of our prototype, since Semi-handled and Unhandled features are the ones that we have to take care of.

As we stated in **Chapter 4** and **Chapter 5** we created our tool by proceeding from simple cases to more complex ones. The final feature that we managed to implement was the “while loop” (control structures). We didn't move further because of a combination of reasons. These reasons were:

- Complexity of the remaining features.
- Dependencies between the remaining features. For example when we were trying to handle functions, we had to implement different solutions for cases that there was HTML generation inside the body of the function and for the cases that there wasn't. However, what if there was a variable integer inside the body of the function that was increased or decreased (and there wasn't any HTML generated) ? And what if this variable was later generated outside the body of the function? Furthermore, what if this variable was returned with a return statement? We should take care of all these cases.
- Lack of time.

The combination of the previous reasons made us decide not to deal with more PHP features. Furthermore, we believe that the level of our tool is sufficient for a prototype. The next step (since Smarty doesn't support the do-while and the switch structures) is to implement the for loop and a satisfactory solution for functions.

As a conclusion we can say that in this thesis we managed to achieve our primary goal, that is to answer the research question and implement a prototype that demonstrate it practically. During the implementation of the prototype we were occupied with a wide variety of PHP features, some simple and some more complex. In the end we managed to overcome a big part of the complexity of PHP. By being aware what the remaining features are, we can now focus in particular cases (the most complex ones), making our prototype a very useful and handy tool.

References

[1] Martin Fowler , **“Refactoring - Improving the Design of Existing Code, page 9”**, Addison-Wesley Longman Publishing Co, 1999**.**

[2] Terence Parr, **“Enforcing Strict Model-View Separation in Template Engines"**, Proceedings of the 13th International Conference on World Wide Web, ACM, May 2004.

[3] F.J. García, Raúl Izquierdo Castanedo, and Aquilino A. Juan Fuente, **"A Double-Model Approach to Achieve Effective Model-View Separation in Template Based Web Applications, page 442-456"**, ICWE'07 Proceedings of the 7th international conference on Web engineering, 2007.

[4] Paul Klint, Tijs van der Storm, Jurgen Vinju**, "RASCAL: a Domain Specific Language for Source Code Analysis and Manipulation, page 168-177"**, SCAM '09 Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, 2009.

[5] Joel Jones, **“Abstract SyntaxTree Implementation Idioms",** Proceedings of the 10th Conference on Pattern Languages of Programs PLoP2003, 2003.

[6] Jos Warmer, Sylvia Van Egmond, **"The implementation of the Amsterdam SGML Parser, page 65-90"**, Electronic Publishing—Origination, Dissemination, and Design, Volume 2 Issue 2, July 1989.

[7] Adam Kiezun, Philip J. Guo , Karthick Jayaraman, Michael D. Ernst ,**"Automatic Creation of SQL Injection and Cross-Site Scripting Attacks, page 199-209"** , In Proceedings of the 31st International Conference on Software Engineering, 2009.

[8] Hesam Samimi, Max Schafer, Shay Artzi, Todd Millstein, Frank Tip, and Laurie Hendren, **“Automated Repair of HTML Generation Errors in PHP Applications Using String Constraint Solving”**, Proceedings of the 34th International Conference on Software Engineering (ICSE 2012), 2012.

[9] Fabian Bannwart, Peter Muller, **“Changing Programs Correctly: Re-factoring with Specifications, page 492-507”,** Proceedings of the 14th international conference on Formal Methods, 2006.

[10] Tom Mens, Tom Tourwé **“A Survey of Software Refactoring, page 126-139 ”**, IEEE Transactions on Software Engineering, Volume 30 Issue 2, February 2004.

[11] H. V. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen, **“Auto-Locating and Fix-Propagating for HTML Validation Errors to PHP Server-Side Code, page 13–22”,** Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, 2011.

[12] Yu Ping, Kontogiannis. K., Lau, T.C., **"Transforming legacy Web applications to the MVC architecture, page 133-142"**, Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice, 2003.

[13] Hasin Hayder, Lucian Gheorghe, **“Smarty PHP Template Programming and Applications, page 5-12”,** Packt Publishing Ltd, 2006.

[14] S. Artzi, J. Dolby, F. Tip, and M. Pistoia. **“Practical fault localization for dynamic Web applications, page 265-274”,** Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, 2010.