Varis: IDE Support for Embedded Client Code in PHP Web Applications

Hung Viet Nguyen
ECpE Department
Iowa State University, USA

Christian Kästner
School of Computer Science
Carnegie Mellon University, USA

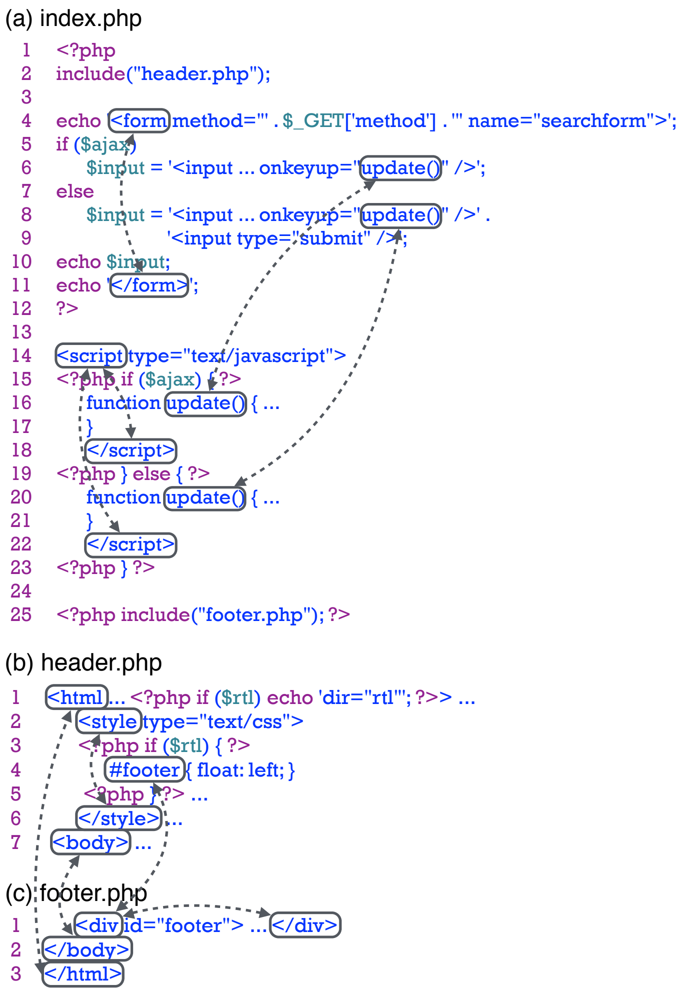
Tien N. Nguyen
ECpE Department
Iowa State University, USA

***Abstract*—In software development, IDE services such as syntax highlighting, code completion, and "jump to declaration" are used to assist developers in programming tasks. In dynamic web applications, however, since the client-side code is dynamically generated from the server-side code and is *embedded* in the server-side program as string literals, providing IDE services for such embedded code is challenging. In this work, we introduce Varis, a tool that provides editor services on the client-side code of a PHP-based web application, while it is still embedded within server-side code. Technically, we first perform symbolic execution on a PHP program to approximate all possible variations of the generated client-side code and subsequently parse this client code into a *VarDOM* that compactly represents all its variations. Finally, using the VarDOM, we implement various types of IDE services for embedded client code including syntax highlighting, code completion, and "jump to declaration". The video demonstration for Varis is available at http://www.youtube.com/watch?v=w1TECeRXGrg.**

I. Introduction

Web applications are among the fastest growing software systems. To support developers in writing their software, modern integrated development environments (IDEs) typically provide editor services such as syntax highlighting, code completion, refactoring, and other types of code analysis. While those IDEs have provided support for traditional software applications, supporting dynamic web applications is challenging due to the dynamic generation of code. In a dynamic web application, web developers write server-side programs to generate client-side programs, which will then be executed by a web browser to display web pages. A server-side program is written in server-side languages such as PHP, ASP, and JSP, whereas the client-side code is written in client-side languages such as HTML, JavaScript (JS), and CSS. Since the server program is executed to generate the client code, it also contains client code appearing as string literals or inline client code. Existing tools currently support either the server-side code or the generated client-side code (e.g., providing syntax highlighting for PHP or HTML), but do not provide support for such *embedded client code*. By means of an example, let us illustrate the challenges of analyzing embedded code.

Figure 1 shows an example PHP web application adapted from AddressBook-6.2.12. The main program (Figure 1a) generates different HTML input fields in an HTML form and different definitions of two JS functions with the same name update, depending on whether the AJAX option is enabled. The top and bottom parts of the page are generated by the files



```
(a) index.php
1   <?php
2   include("header.php");
3
4   echo '<form method="' . $_GET['method'] . '" name="searchform">';
5   if ($ajax)
6       $input = '<input ... onkeyup="update()" />';
7   else
8       $input = '<input ... onkeyup="update()" />' .
9             '<input type="submit" />';
10  echo $input;
11  echo '</form>';
12  ?>
13
14  <script type="text/javascript">
15  <?php if ($ajax) { ?>
16      function update() { ...
17      }
18      </script>
19  <?php } else { ?>
20      function update() { ...
21      }
22      </script>
23  <?php } ?>
24
25  <?php include("footer.php"); ?>
```

```
(b) header.php
1   <html ... <?php if ($rtl) echo 'dir="rtl"'; ?>> ...
2     <style type="text/css">
3     <?php if ($rtl) { ?>
4       #footer { float: left; }
5     <?php } ?> ...
6     </style> ...
7   <body> ...
```

```
(c) footer.php
1     <div id="footer"> ... </div>
2   </body>
3   </html>
```

Fig. 1. An example PHP program with call-graph edges for embedded client code (string literals and inline client code are highlighted in blue)

in Figures 1b and 1c, respectively. Note that the client code (written in HTML, JS, CSS) often appears in the server code as inline code (which is separated from PHP code by the directive <?php...?> and is sent verbatim to the client side when the PHP program is executed) or *string literals*. Analyzing such embedded client code is challenging due to several reasons:

1) The client code is *embedded* in server-side strings. For example, the PHP string on line 6 of Figure 1a contains

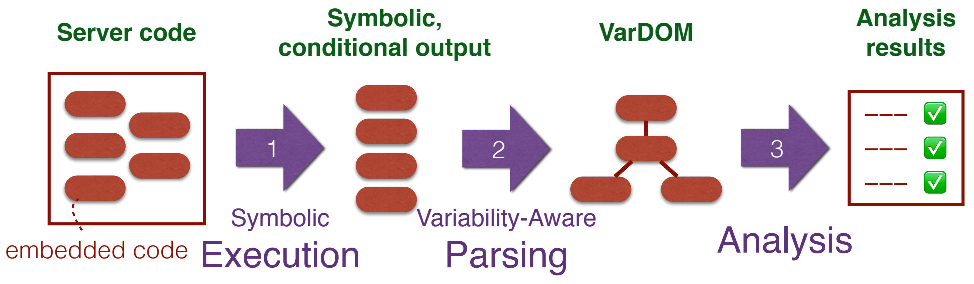


Fig. 2. Approach overview

an HTML input with a JS function call update handling its onkeyup event. An analysis would need to understand the semantics of such string literals to analyze the embedded client code.

2) The client code is often *computed from various sources*. In our example, the two string literals representing two HTML input fields are concatenated, and the combined string is then assigned to a PHP variable and later output at a different statement (lines 8–10 in Figure 1a). Therefore, without seeing the concatenated result, the meaning of an embedded code fragment and how it relates to other fragments are not straightforward.
3) Embedded client code is often *scattered* across different locations in the server-side source code, and spans across multiple files. Also, client-side code could be *fragmented* and may not form a meaningful syntactical unit of the corresponding language (e.g., the first string literal on line 4 of Figure 1a contains an incomplete HTML opening tag). Therefore, parsing these scattered embedded code fragments is non-trivial.
4) The server-side program can generate different *variations of the client-side code* depending on specific values at run time. In our example, the two function calls update have two different definitions depending on whether the AJAX option is enabled or not (whether the PHP variable $ajax evaluates to true). Similarly, the HTML opening <script> tag at line 14 has two alternative closing tags in the respective cases of the AJAX option at line 18 or 22, and the footer <div> element has different CSS styles depending on the server-side execution. During analysis, these strings need to be taken into consideration to ensure feasible relations between embedded code elements.

II. Approach

To address those challenges, we propose Varis, a novel tool to support embedded code analysis for PHP-based web applications. We combine symbolic execution, variability-aware parsing, and client code analyses. The approach proceeds in three steps as shown in Figure 2.

In the first step, we approximate the output of a PHP program using a symbolic execution engine we developed in prior work [7], [5]. The symbolic executer explores *all feasible paths* in the program to obtain all possible outputs (with

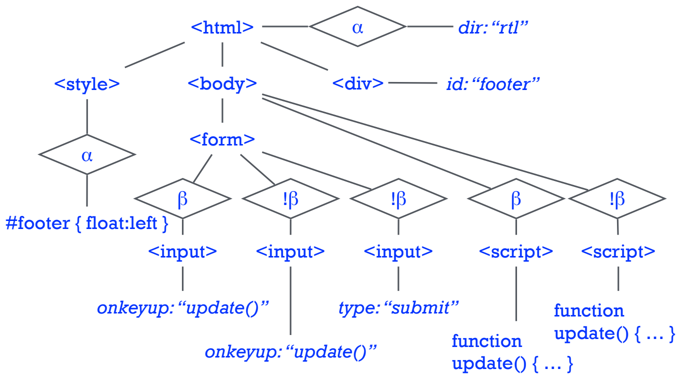
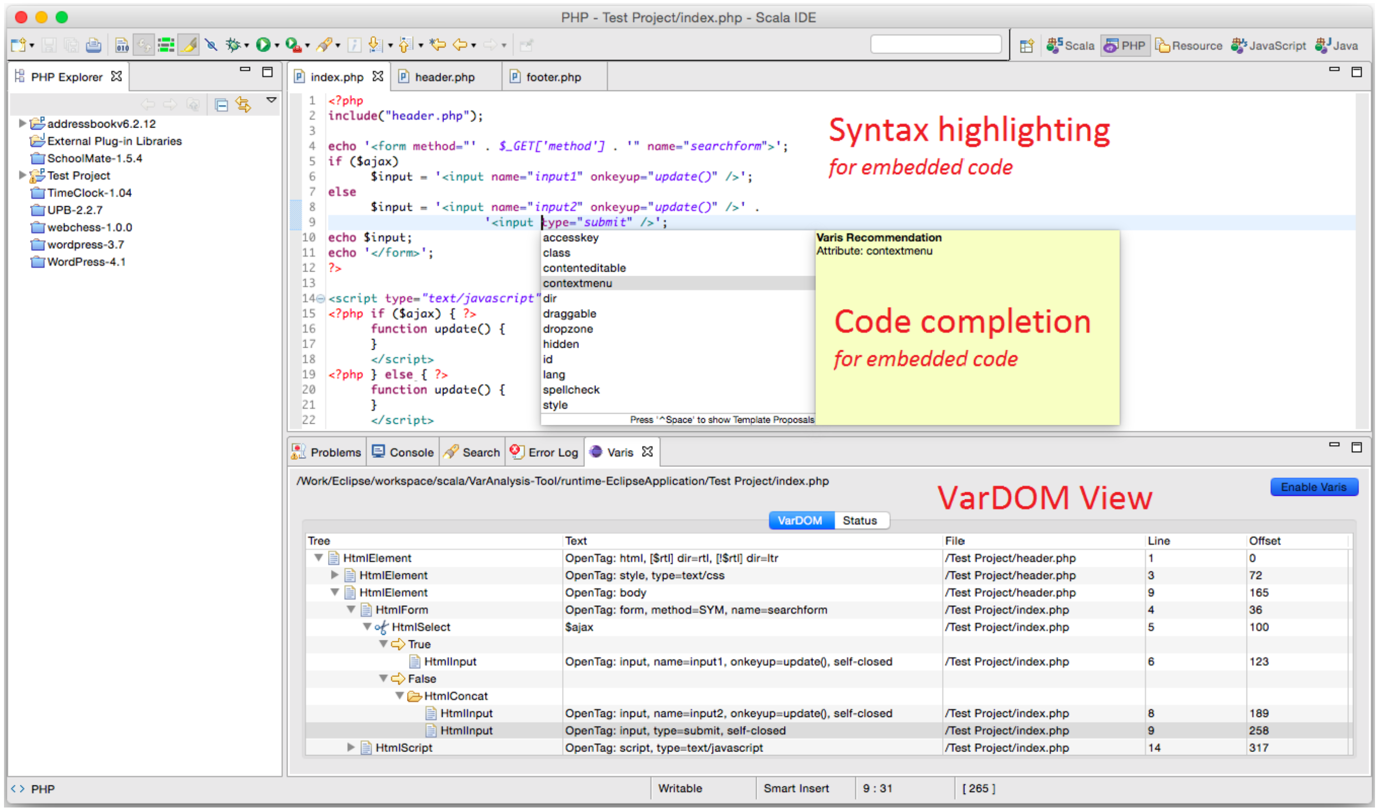


Fig. 3. The VarDOM for the PHP program in Figure 1 with condition nodes in diamond shapes and Greek letters representing symbolic values

conservative approximation). When encountering unknown data such as user inputs or data from databases, we represent them with *symbolic values*. The result of symbolic execution is the computed client code in which certain values could be symbolic or generated under some path conditions.

In the second step, we parse this output with symbolic values and conditional characters. This task resembles parsing C code with pre-processing directives (e.g., #ifdefs), which have been addressed recently [8], [3], [2]. In this work, we use our TypeChef variability-aware parser framework [3]. A parser built on top of the TypeChef framework is able to take any program with conditional characters and parse it into a *conditional* AST. A conditional AST contains regular AST nodes for the given language and *condition nodes* to indicate that the subtree under the condition node is generated only under the corresponding condition. We extended TypeChef to create a variability-aware parser for HTML. This parser parses the output from the previous step into a conditional DOM (similar to the DOM for HTML but having condition nodes; see Figure 3). We then extract CSS and JS code fragments from this conditional DOM (which in turn can also contain conditional characters) and parse them with two further variability-aware parsers for CSS and JS into conditional ASTs for CSS and JS. In the final step, we assemble all these conditional ASTs for HTML, CSS, and JS into a single VarDOM representation which compactly represents all possible variations of the generated client code. More technical details can be found in [5].



```
 1 <?php
 2 include("header.php");
 3
 4 echo '<form method="' . $_GET['method'] . '" name="searchform">';
 5 if ($ajax)
 6       $input = '<input name="input1" onkeyup="update()" />';
 7 else
 8       $input = '<input name="input2" onkeyup="update()" />' .
 9                      '<input type="submit" />';
10 echo $input;
11 echo '</form>';
12 ?>
13
14 <script type="text/javascript">
15 <?php if ($ajax) { ?>
16       function update() {
17       }
18       </script>
19 <?php } else { ?>
20       function update() {
21       }
22       </script>
```

```
accesskey
class
contenteditable
contextmenu
dir
draggable
dropzone
hidden
id
lang
spellcheck
style
Press '^Space' to show Template Proposals
```

/Work/Eclipse/workspace/scala/VarAnalysis-Tool/runtime-EclipseApplication/Test Project/index.php

Tree	Text	File	Line	Offset
HtmlElement	OpenTag: html, [$rtl] dir=rtl, [!$rtl] dir=ltr	/Test Project/header.php	1	0
HtmlElement	OpenTag: style, type=text/css	/Test Project/header.php	3	72
HtmlElement	OpenTag: body	/Test Project/header.php	9	165
HtmlForm	OpenTag: form, method=SYM, name=searchform	/Test Project/index.php	4	36
HtmlSelect	$ajax	/Test Project/index.php	5	100
True				
HtmlInput	OpenTag: input, name=input1, onkeyup=update(), self-closed	/Test Project/index.php	6	123
False				
HtmlConcat				
HtmlInput	OpenTag: input, name=input2, onkeyup=update(), self-closed	/Test Project/index.php	8	189
HtmlInput	OpenTag: input, type=submit, self-closed	/Test Project/index.php	9	258
HtmlScript	OpenTag: script, type=text/javascript	/Test Project/index.php	14	317

Fig. 4. The VarDOM view, syntax highlighting, and code completion support in Varis

The VarDOM provides the foundation for different types of analysis. These include building call graphs for embedded JS code, syntax highlighting and code completion on HTML elements and CSS rules, "jump to declaration" (navigating from JS function calls to their declarations, from opening to closing HTML tags, and from CSS rules to selected HTML elements), and potentially many others.

III. The Varis Tool

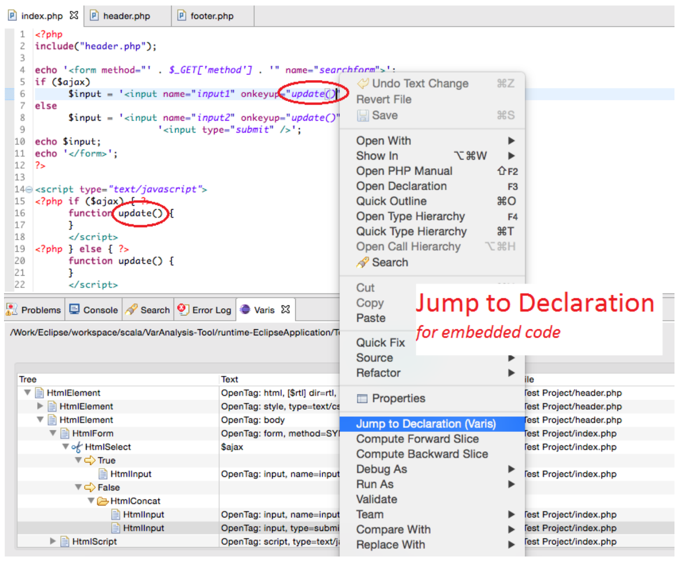
We implemented Varis as a plug-in to the Eclipse IDE. In our current implementation, the key features in Varis include the display of the VarDOM and three IDE editor services for embedded client code: (1) syntax highlighting, (2) code completion, and (3) "jump to declaration".

1) The VarDOM view: When a PHP program is loaded and the Varis tool is enabled, Varis analyzes the PHP program and displays its VarDOM tree in an Eclipse tree view (the lower half in Figure 4). Each HTML element is displayed with its type, textual content, and location in the PHP code. Condition nodes in the VarDOM are annotated with an arrow. When the user selects an HTML element in the VarDOM view, its corresponding text in the source code will be highlighted. For instance, the highlighted HTML element in Figure 4 shows that it is located on line 9 of the current PHP file and the label of its condition node indicates that the element is generated when $ajax evaluates to false.

2) Syntax highlighting: With the type and location information of HTML elements in the VarDOM, Varis is able to highlight syntactic elements in the embedded client code. For instance, the HTML opening tag, the HTML attribute name, and HTML attribute values are colored differently on line 4 of Figure 4. Note that even though the HTML tag at line 4 is split into three fragments in the server code with some unknown data ($_GET['method']), Varis is still able to highlight the code elements correctly.

3) Code completion: When the user points to a position inside a PHP string and invokes code completion support, Varis recognizes the code element of the embedded code at that location and provides a list of code recommendations for the code element as if it was written on static client code (without being embedded in a string). As an illustration, in Figure 4, since the user requests code completion support at a position after the HTML input tag name, Varis recommends attribute names that an HTML input can have. We use the W3C standard on HTML elements and their appropriate attributes for embedded HTML code completion.

4) Jump to declaration: Based on the underlying call graph created from the VarDOM, Varis allows the user to navigate between sources and targets in calling relationships: from JS function calls to their declarations, from opening to closing HTML tags, and from CSS rules to selected HTML elements. In Figure 5, when the user selects the JS function call update on line 6, a context menu appears allowing the user to use



```
index.php   header.php   footer.php
1  <?php
2  include("header.php");
3
4  echo '<form method="' . $_GET['method'] . '" name="searchform">';
5  if ($ajax)
6        $input = '<input name="input1" onkeyup="update()"
7  else
8        $input = '<input name="input2" onkeyup="update()"
9                    '<input type="submit" />';
10 echo $input;
11 echo '</form>';
12 ?>
13
14 <script type="text/javascript">
15 <?php if ($ajax) { ?>
16       function update() {
17       }
18       </script>
19 <?php } else { ?>
20       function update() {
21       }
22       </script>
```

```
/Work/Eclipse/workspace/scala/VarAnalysis-Tool/runtime-EclipseApplication/T

Tree                          Text                              File
▼ HtmlElement                 OpenTag: html, [$rtl] dir=rtl,    Test Project/header.php
  ▶ HtmlElement               OpenTag: style, type=text/cs      Test Project/header.php
  ▼ HtmlElement               OpenTag: body                     Test Project/header.php
    ▼ HtmlForm                OpenTag: form, method=SY          Test Project/index.php
      ▼ HtmlSelect            $ajax                             Test Project/index.php
        ▼ True
            HtmlInput         OpenTag: input, name=input        Test Project/index.php
        ▼ False
          ▼ HtmlConcat
              HtmlInput       OpenTag: input, name=input        Test Project/index.php
              HtmlInput       OpenTag: input, type=submi        Test Project/index.php
    ▶ HtmlScript              OpenTag: script, type=text/j      Test Project/index.php
```

Fig. 5. "Jump to declaration" support in Varis

the "Jump to Declaration" functionality, which would take the user to the function declaration on line 16. If one source has multiple targets, Varis shows the condition of the jump and allows the user to select the conditional navigation. For example, Varis will show the AJAX option and allow the user to choose the navigation from the opening <script> tag at line 14 to its respective closing tag at either line 18 or 22. Details on the call graph construction algorithm and its accuracy are described in [5].

For most subject systems used in our study, the initial creation of the VarDOM and call graph completed within a few seconds [5]. When the source code is changed, Varis re-analyzes relevant files only. Therefore, Varis is suitable as a background service of an IDE. More information about the Varis tool and its source code can be found on our website [1].

IV. Related Work

Contemporary IDEs such as Eclipse, NetBeans, and PhpStorm typically provide a rich set of IDE services ranging from syntax highlighting, syntax validation, to code completion and refactoring. However, these services are not applicable to client-side code that is *embedded* within server-side code (since these code fragments are treated as merely string constants in the server-side program). In our work, we aim to fill in that gap in supporting embedded client code.

There exist several other approaches for analyzing embedded client code. PHPQuickFix and PHPRepair [9] detect errors in PHP programs that result in invalid HTML code. While PHPQuickFix considers only PHP echo/print statements, and PHPRepair uses a dynamic approach with test cases to examine the generated client code, Varis uses our symbolic execution engine PhpSync [7] to track how string literals are computed and approximate all possible outputs. We also used PhpSync in DRC [6], a tool for detecting dangling references in PHP web applications. DRC collects program entities in the symbolic output via heuristics without explicitly parsing the output into a DOM and ASTs as we did with Varis. In [5], we demonstrated building call graphs for embedded client code (without tooling). In this work, we put together the call graph application and a number of other services into a supporting tool for web development.

Minamide proposed a string analyzer [4] that approximates HTML output via context-free grammar. Wang *et al.* [10] reused the string analyzer and detected the constant strings for translation. These works do not analyze JS or CSS and do not aim to provide IDE services for embedded client code.

V. Conclusion and Future Work

In dynamic web applications, client-side code is often embedded in server-side string literals. Due to the generation process from the server code to produce client code, supporting analysis for such embedded client code is challenging. We introduce Varis, a tool that provides IDE services for embedded client-side code. We demonstrated Varis' capabilities with three services: syntax highlighting, code completion, and "jump to declaration". We plan to support more services such as auto-correction, refactoring, and visualization of the VarDOM in our future work.

VI. Acknowledgments

This project is funded in part by National Science Foundation grants: CCF-1318808, CCF-1018600, CNS-1223828, CCF-1349153, CCF-1320578, and CCF-1413927.

References

[1] Building call graphs for embedded client-side code in dynamic web applications. http://home.engineering.iastate.edu/~hungnv/Research/Varis/.

[2] P. Gazzillo and R. Grimm. SuperC: Parsing all of C by taming the preprocessor. In *Proc. Conf. Programming Language Design and Implementation (PLDI)*. ACM Press, 2012.

[3] C. Kästner, P. G. Giarrusso, T. Rendel, S. Erdweg, K. Ostermann, and T. Berger. Variability-aware parsing in the presence of lexical macros and conditional compilation. In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pages 805–824. ACM Press, 2011.

[4] Y. Minamide. Static approximation of dynamically generated web pages. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 432–441. ACM Press, 2005.

[5] H. V. Nguyen, C. Kästner, and T. N. Nguyen. Building call graphs for embedded client-side code in dynamic web applications. In *Proc. Int'l Symposium Foundations of Software Engineering (FSE)*, pages 518–529. ACM Press, 2014.

[6] H. V. Nguyen, H. A. Nguyen, T. T. Nguyen, A. T. Nguyen, and T. N. Nguyen. Dangling references in multi-configuration and dynamic PHP-based Web applications. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*. IEEE Computer Society, 2013.

[7] 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. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*, pages 13–22. IEEE CS, 2011.

[8] Y. Padioleau. Parsing C/C++ code without pre-processing. In *Proc. Int'l Conf. Compiler Construction (CC)*, pages 109–125. Springer-Verlag, 2009.

[9] H. Samimi, M. Schäfer, S. Artzi, T. Millstein, F. Tip, and L. Hendren. Automated repair of HTML generation errors in PHP applications using string constraint solving. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 277–287. IEEE Press, 2012.

[10] X. Wang, L. Zhang, T. Xie, H. Mei, and J. Sun. Locating need-to-translate constant strings in web applications. In *Proc. Int'l Symposium Foundations of Software Engineering (FSE)*, pages 87–96. ACM Press, 2010.