Detection of Embedded Code Smells in Dynamic Web Applications

Hung Viet Nguyen
hungnv@iastate.edu

Hoan Anh Nguyen
hoan@iastate.edu

Tung Thanh Nguyen
tung@iastate.edu

Anh Tuan Nguyen
anhnt@iastate.edu

Tien N. Nguyen
tien@iastate.edu

Electrical and Computer Engineering Department
Iowa State University
Ames, IA 50011, USA

ABSTRACT

In dynamic Web applications, there often exists a type of code smells, called *embedded code smells*, that violate important principles in software development such as software modularity and separation of concerns, resulting in much maintenance effort. Detecting and fixing those code smells is crucial yet challenging since the code with smells is embedded and generated from the server-side code.

We introduce WebScent, a tool to detect such embedded code smells. WebScent first detects the smells in the generated code, and then locates them in the server-side code using the mapping between client-side code fragments and their embedding locations in the server program, which is captured during the generation of those fragments. Our empirical evaluation on real-world Web applications shows that 34%-81% of the tested server files contain embedded code smells. We also found that the source files with more embedded code smells are likely to have more defects and scattered changes, thus potentially require more maintenance effort.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Algorithms, Languages, Management, Reliability

Keywords

Code Smells, Dynamic Web Applications, Embedded Code

1. INTRODUCTION

Software maintenance is a crucial phase during software development and can cost up to 75% of developers' time and effort [2]. To reduce the cost in software maintenance, several design/development principles have been proposed and widely adopted. Among others, the important principles are the *separation of concerns*, *software modularity*, and *compliance with coding standards* [2].

In Web applications, however, there exist several types of programming practices that violate those principles. For example, an HTML page often contains multiple code fragments realizing different concerns/functions. The HTML code for *presentation content* (e.g. page structure and user interface elements) is intermixed with the JavaScript (JS) code for *control logic* (e.g. input data validation and event handling), which violates the separation of concerns principle. Moreover, for some Web pages having similar structures and control logic, developers may choose to duplicate the code instead of creating a separate module containing the shared code that can be reused. This "copy-and-paste" practice violates the principle of software modularity. Another type of violation is non-conformance to coding standards. Specifically, many Web pages are not syntactically correct with respect to the grammars of Web languages such as HTML. Even though those syntactic errors may be tolerated by modern Web browsers, they make Web pages more difficult to maintain, especially when the maintenance work is performed by different developers. The above violations are referred to as *code smells* and have been shown to reduce maintainability and cause difficulties in evolving and enhancing the software [2].

In a *dynamic* Web application, in addition to maintaining the server-side code, developers must spend their efforts to maintain and evolve the client-side code *embedded* in the server-side program as well (e.g., modifying the control logic in JS or presentation structure in HTML). In such an application, embedded code is written in client-side languages such as HTML, JS, or CSS, and is scattered within server-side code written in a host language such as PHP. For example, JS functions or HTML elements are often contained in PHP string literals and are output via PHP's `echo`/`print` functions. If the generated client-side code contains the aforementioned smells, the corresponding embedded code which is used to generate that client-side code contains the code smells as well. Because those smells exist in embedded code, we call them **embedded code smells**. Since developers must maintain/evolve embedded code, those smells have negative impacts on the productivity. They also affect other maintenance tasks when developers must analyze the corresponding generated client-side code. Examples of those tasks include run-and-testing and debugging on the control code in JS or the presentation code in HTML/CSS.

While fixing embedded code smells is important, detecting them is challenging. Because those smells exist in embedded code within scattered PHP string literals, they are not easily exposed and recognized until the client-side code is generated from the server-side code. Moreover, to locate and fix those embedded code smells, one might need to detect them in the client-side code, and locate and fix them in the corresponding locations in the server-side code. This is not trivial due to the nature of scattered, embedded code.

Our research aims to detect such embedded code smells within server-side code and to enable the study of their impacts on software maintenance and software quality. We introduce WebScent, a tool to detect embedded code smells within PHP server code. In WebScent, we have identified the following types of code smells and detected them:

1. Mixing code for presentation content (HTML), presentation styles (CSS), and presentation/business logic (JS),
2. Duplication of JS code in the server-side code,
3. Scattered locations of closely-related client-side elements in the server-side code, and
4. HTML syntactic errors.

To detect and locate the embedded code smells in a PHP-based Web application, WebScent works as follows. First, it uses a dynamic mapping method to execute the server code via an instrumented PHP interpreter, and to record the mapping for all fragments in the output client-side code to their original locations in the server code where they are often embedded in PHP string literals. Second, it then detects code smells in the generated client-side code by analyzing the client-side code and the corresponding locations of its elements in the server code. Finally, based on the client-side code smells detected in step 2, it uses the client-to-server mapping built in step 1 to locate the embedded code smells in the corresponding locations in the server code.

Our empirical evaluation on six real-world Web systems shows that 34%-81% of the tested server files in those systems contain embedded code smells. The generated client pages from them on average contain from 12 up to 151 smells of various types depending on the size of the system. We also found that 88%-100% of the detected smells in the client code actually have corresponding embedded code in PHP string literals on the server code (while the remaining ones correspond to non-literal PHP expressions). More importantly, using WebScent, we performed an empirical experiment and found that *the source files with more code smells are likely to have more scattered changes as well as more defects.* The key contributions of our paper include:

1. The definitions and detection techniques realized in WebScent for six types of embedded code smells,
2. An empirical evaluation in which WebScent is able to detect embedded code smells in six real-world Web projects,
3. Empirical evidence on real-world Web applications suggesting that the source code having embedded code smells has lower quality and requires more maintenance effort.

2. EMBEDDED CODE SMELLS

Let us present a set of *embedded code smells* (Table 1) that violate three important principles in software design: *separation of concerns*, *software modularity*, and *coding standards.*

Table 1: Embedded Code Smells

Smell Type	Smell Name	Violations of Principles
Type 1	JS in HTML	Separation of Concerns
Type 2	CSS in JS	Separation of Concerns
Type 3	CSS in HTML	Separation of Concerns
Type 4	Scattered Sources	Software Modularity
Type 5	Duplicate JS	Software Modularity
Type 6	HTML Syntax Error	Coding Standards

2.1 Bad Smells in Mixed Code

The embedded code smells in this category violate the principle of separation of concerns. We define the following types:

Type 1 (JS in HTML): A portion of JS code is embedded inside HTML code. This smell shows the mixture of the code for presentation logic (JS) and that for presentation content (HTML). The JS code might occur in three kinds of places in the HTML code as follows.

Type 1.1 (Inline JS): A portion of JS code is inlined as the text content of a script tag inside HTML code.

The JS code for control logic should be extracted into a file separated from HTML code for presentation if possible, and included via HTML src attribute.

Type 1.2 (JS in HTML links): A portion of JS code is inlined as the value of the href attribute of an HTML anchor tag such as <a href='javascript:history.back()'>.

The anchor should be free of JS code as in <a id = 'backlink' href='#'> Click Here </a>. The JS code will be as follows.

```
...
var backlink = document.getElementById('backlink');
backlink.addEventListener('onclick',...);
```

Type 1.3 (JS in HTML events): A portion of JS code is inlined as the value of an event attribute of an HTML element. The code within the quotes is the JS code defining the handler for the form submission event. To separate this JS code, one should define the event handler within JS code and associate it with the form as in the case of Type 1.2.

Type 2 (CSS in JS): An assignment statement in JS code sets some properties of the style attribute of an HTML element. This smell shows the mixing of the code for presentation style (CSS) and that for control logic (JS). Different styles should be defined in CSS stylesheets and are associated with presentation elements.

Type 3 (CSS in HTML): A style property of an HTML element is set inside HTML code, e.g. <div align='center' style='color:red;'>. This smell shows the mixing of the code for presentation style and that for presentation content. In this case, the CSS and HTML code should be separated.

2.2 Bad Smells in Scattered Code

Type 4 (Scattered Sources): Multiple *scattered, embedded sources* in the server code are used to generate two *closely-related* portions of client code. We define three cases where portions of client code are considered *closely-related*:

1. An HTML opening tag and its respective closing tag,
2. The attributes of an HTML tag (including the attributes' names and values), and
3. All statements inside a JS function or all parts of a single statement declared outside of any JS function.

On the server side, we consider two portions of code as *scattered* if they are in different functions. The rationale is that each function is designed to fulfill a well-defined task. An example of this type of smells is when an HTML opening tag and its closing tag are generated from two different functions in the server code. In that case, it might be difficult for maintenance, e.g. checking if an opening tag is properly closed. Note that this type of smells requires the conditions on both client-side and server-side code (see Type 4).

For this type of smells (scattered sources), after generating the client page and recording the client-server mapping, WebScent analyzes the generated client code in HTML/JS and the corresponding server-side source locations in PHP according to the above definition to detect the smells.

2.3 Bad Smells in Duplicated Code

Type 5 (Duplicate JS): Two portions of embedded JS code f and g are considered as duplicate if (1) their similarity, measured by function $sim(.)$, exceeds a threshold σ, i.e. $sim(f, g) \geq \sigma$, and (2) they do not overlap each other.

For this type, WebScent takes different inputs to generate the client pages of interest. It then uses the clone detection technique in our prior work [1] to detect this type of smells. The technique first traverses the abstract syntax trees (AST) of the JS code in all client pages to build the set of all code fragments and the corresponding vectors representing their structures. Then, it compares the vectors to measure the similarity between the corresponding fragments.

Note that JS fragments that are duplicate on the client side may be generated from the same or overlapping sources in server code. Moreover, either of them may be produced by multiple sources. In those cases, those embedded code fragments on the server side are not considered as duplicate.

2.4 Bad Smells in Non-Standard Code

Type 6 (HTML Syntax Error): A portion of HTML code does not conform to the specification of the HTML language (e.g. missing closing tags). WebScent uses Tidy [3], a tool for detecting HTML validation errors to identify this kind of smells in the generated client page, and locates them in the server code using the client-server mapping.

3. EMPIRICAL STUDY AND EVALUATION

For evaluation, we sought to answer the following:

RQ1. Via WebScent, how many embedded code smells can be detected in the server-side code? How many smells of each type are there in client pages?

RQ2. Software quality and maintenance effort due to embedded code smells. How do embedded code smells affect the quality and maintainability of a Web application?

To answer those questions, we conducted our experiments on six PHP-based Web applications (Table 2). For each system, we set up sample data and navigated through its Web browser interface to create test cases for many server pages. Then, we ran WebScent on them to detect code smells.

3.1 Embedded Smells in Server Code

To detect the embedded code smells in those systems, WebScent first detected different types of code smells in the client code and then used output mappings to locate them back to PHP code. During the process, we also recorded the numbers of client-side code smells of different types detected on the test client pages and the number of smells that can be located back to PHP via our mapping technique (Table 3).

Table 2: Subject Systems

System	Ver	Files	KLoc	Avg Loc	Down-Loads
AddressBook (AB)	6.2.12	103	19	184	54,783
DotProject (DP)	2.1.5	769	201	261	1,278,713
Mambo (MB)	4.6.5	995	176	177	878,428
PhpFusion (PF)	7.2.3	795	91	114	1,403,419
WebCollab (WC)	2.71	276	48	174	75,135
ZenCart (ZC)	1.3.9	1118	156	140	2,481,766

For each type, column Sum shows the number of smell instances detected in all the test pages, whereas column Avg. gives the average number per test page. The smell instances of a given type are further classified as Lit if they are located in PHP string literals (embedded code), and NLit otherwise. For example, WebScent detected in Mambo (MB) a total of 1,881 smell instances of type JS in HTML over 58 test pages, or 32 smell instances per test page on average. 100% of those smells are mapped to PHP string literals where developers can directly make changes to fix the smells. The aggregate numbers of smells for all smell types are given in column All Smell Types. Duplicate JS smells will be reported in Table 4 describing the smells in the server code since two duplicate code fragments on the client side may be generated from the same source on the server side. To show the density of the smells on a client page, we counted the average ratio of the number of detected smells over the number of HTML tags in a client page (last column, Table 3).

As seen, WebScent has detected up to 151 smells per client page (with density up to 69%). The majority of the smells (88%-100%) can be mapped back to PHP string literals. Thus, WebScent is very useful in helping developers in locating the PHP code to fix those smells. Most importantly, this very high percentage shows that *the smells in the embedded code (i.e. embedded code smells) on the server side actually account for 88%-100% of the smells in the client-side code.* Thus, our embedded code smell detection approach via detecting client-side code smells is very effective. For those client code smells that do not come from the embedded code in PHP strings, the corresponding client code is constructed from data from databases or PHP string built-in functions.

RQ1. How many embedded code smells are there in the tested server pages? To answer that, after locating the client-side code smells back to server code in the previous study, we then counted the distinct *embedded code locations* in the server files that contain the smells (since two client-side code smells may be generated from the same source in the server code). In Table 4, the number of embedded code smells are given in the rows Sum for all smell types from 1-6. Column Tested shows the number of involved server files. The rows %Files display the percentage of those server files containing at least one smell. For instance, in ZenCart (ZC), WebScent detected 90 smells of type 5, and 15% of the tested server files contain this smell. Overall, 34%-81% of the tested server files contain at least an embedded code smell, and some systems have over 1,000 smells (column All).

3.2 Code Smells and Maintenance Effort

To investigate the relation of embedded code smells and the system maintainability, we first hypothesized that:

H1. Source files having embedded code smells generally have lower quality than source files without code smells.

Table 3: Results on Detecting Embedded Code Smells

Sys	test		JS in HTML			CSS in JS			CSS in HTML			Scattered Src			HTML Syn Err			All Smell Types			Smells / Tag
			Sum	Avg	%	Sum	Avg	%	Sum	Avg	%	Sum	Avg	%	Sum	Avg	%	Sum	Avg	%	
		Lit	14	1	100%	4	0	100%	18	1	100%	60	4	100%	87	6	99%	183	12	99%	
AB	15	NLit	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	1%	1	0	1%	
		Sum	14	1		4	0		18	1		60	4		88	6		184	12		16%
		Lit	1465	28	100%	13	0	100%	5252	99	100%	290	5	100%	955	18	97%	7975	150	100%	
DP	53	NLit	0	0	0%	0	0	0%	0	0	0%	0	0	0%	27	1	3%	27	1	0%	
		Sum	1465	28		13	0		5252	99		290	5		982	19		8002	151		67%
		Lit	1877	32	100%	14	0	100%	4806	83	99%	16	0	94%	1709	29	90%	8422	145	97%	
MB	58	NLit	4	0	0%	0	0	0%	50	1	1%	1	0	6%	197	3	10%	252	4	3%	
		Sum	1881	32		14	0		4856	84		17	0		1906	33		8674	150		69%
		Lit	441	7	100%	0	0		3683	55	96%	134	2	100%	1367	20	70%	5625	84	88%	
PF	67	NLit	0	0	0%	0	0		135	2	4%	0	0	0%	597	9	30%	732	11	12%	
		Sum	441	7		0	0		3818	57		134	2		1964	29		6357	95		35%
		Lit	177	4	100%	0	0		350	8	100%	0	0		134	3	100%	661	15	100%	
WC	45	NLit	0	0	0%	0	0		0	0	0%	0	0		0	0	0%	0	0	0%	
		Sum	177	4		0	0		350	8		0	0		134	3		661	15		12%
		Lit	1014	19	100%	28	1	100%	4706	87	100%	162	3	100%	1970	36	100%	7880	146	100%	
ZC	54	NLit	4	0	0%	0	0	0%	0	0	0%	0	0	0%	7	0	0%	11	0	0%	
		Sum	1018	19		28	1		4706	87		162	3		1977	37		7891	146		35%

Table 4: Embedded Code Smells in Server Files

System	Tested		Smell Types						
			1	2	3	4	5	6	All
AB	20	Sum	13	2	3	4	18	19	59
		%Files	10%	5%	15%	15%	10%	35%	50%
DP	75	Sum	117	3	628	16	62	175	1001
		%Files	55%	4%	73%	12%	35%	75%	81%
MB	137	Sum	117	2	563	8	19	299	1008
		%Files	31%	1%	45%	4%	7%	46%	54%
PF	205	Sum	64	0	636	1	16	117	834
		%Files	19%	0%	39%	0%	4%	26%	47%
WC	59	Sum	68	0	62	0	0	49	179
		%Files	39%	0%	25%	0%	0%	51%	59%
ZC	270	Sum	86	3	386	47	90	340	952
		%Files	19%	1%	17%	12%	15%	17%	34%

H2. Source files having embedded code smells generally require more maintenance effort than those without smells.

To validate our hypotheses, we use two metrics. We use the *number of bugs* (nBug) that a source file has during its life cycle to indicate its quality. Since development effort is usually estimated via lines of code, we measured maintenance effort for a source file via the *average changed lines of code* (avgCLOC) over all the changes made to that file.

Using the smell detection results in the previous experiment, we divided the source files of each system involved in the generation of the test pages into two groups: S is the group of source files having at least one detected code smell, and $\bar{S}$ is the group of source files without code smells.

Then, we used statistical testing to compare the two metrics, nBug (number of bugs) and avgCLOC (average changed lines of code), for the source files in those two groups to validate *H1* and *H2*. Since we made no assumption about the distribution of those two metrics on S and $\bar{S}$, we used Wilcoxon test, a non-parametric one. Each test for a system produces a *p-value* representing the significance of the test, in which a smaller p-value indicates higher confidence on the tested hypothesis. For example, if p-value $< 5\%$, we could accept the hypothesis with more than 95% confidence.

Table 5: P-values of Wilcoxon tests comparing the quality and maintenance effort of files with and without embedded code smells

Sys	$\|S\|$	$\|\bar{S}\|$	avgCLOC(S)>avgCLOC($\bar{S}$)	nBug(S)>nBug($\bar{S}$)
DP	61	14	0.2567	0.0074
MB	74	63	0.0011	0.0167
PF	96	109	< 2.2e-16	< 2.2e-16
WC	35	24	0.0030	0.0337
ZC	91	179	2.09E-05	0.0331

Table 5 shows the group sizes and the p-values of the statistical tests for five subject systems (we excluded AddressBook since it has just 20 tested pages, thus, the sample size is not sufficient for the test). As shown, the p-values are all significant at 5% level (except the test comparing avgCLOC on DotProject). Thus, we accept both hypotheses *H1* and *H2*, confirming that the source files having embedded code smells have lower quality (having more bugs) and require more maintenance effort than the files without code smells.

4. CONCLUSIONS

We introduce WebScent, a tool for detecting embedded code smells in server code with high accuracy. Our empirical analysis of embedded code smells suggests that source files having smells have lower quality and more scattered changes than the ones without smells.

5. ACKNOWLEDGMENTS

This project is funded in part by US National Science Foundation (NSF) CCF-1018600 grant.

6. REFERENCES

[1] H. A. Nguyen, T. T. Nguyen, N. H. Pham, J. Al-Kofahi, and T. N. Nguyen. Clone management for evolving software. *IEEE TSE*, (PrePrints), 2011.

[2] R. S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw Hill, June 2000.

[3] HTML Tidy Project. http://tidy.sourceforge.net/.