NEAR-MISS CLONE PATTERNS IN WEB APPLICATIONS: AN EMPIRICAL STUDY WITH INDUSTRIAL SYSTEMS

Tariq Muhammad, Minhaz F. Zibran, Yosuke Yamamoto, Chanchal K. Roy

Department of Computer Science, University of Saskatchawan, Saskatoon, SK, Canada

ABSTRACT

Dynamic web pages composed of inter-woven (tangled) source code written in multiple programming languages (e.g., HTML, PHP, JavaScript, CSS) makes it difficult to analyze and manage clones in web applications. Despite more than a decade of research on software clones, there are not many studies towards the investigation of code clones in web applications.

In this paper, we present an in-depth study on the patterns (i.e., forking and templating) of exact and near-miss code clones in two industrial dynamic web applications having distinct architecture. The findings of our study confirm the believed patterns for cloning and suggest that specialized techniques and tool support are necessary for effectively managing clones in the tangled source code of dynamic web applications.

Index Terms— Code clone, analysis, empirical study

1. INTRODUCTION

Code clone (duplicate code) is a well-known code smell that has significant impact on the development and maintenance of software systems. Code snippets that have identical source text except for the comments and layout are called the *Type-1* clones. Syntactically similar code snippets, where there may be variations in the names of identifiers/variables are known as *Type-2* clones. Code fragments that exhibit similarity as of *Type-2* clones and also allow further differences such as additions, deletions or modifications of statements are *Type-3* clones [16]. As reported by the earlier studies, a typical software system may have 9%-17% [18] code clones, while the proportion may go as high as even 68% [3] for certain software components.

The creation of code clones by copy-pasting source code is a common practice that the developers adopt to minimize efforts and speedup development process. On the other hand, duplicated code can unnecessarily inflate the program size, which is often proportional to the maintenance effort. Moreover, code clones can also cause fault propagation [21], and may increase future maintenance effort for consistent change and evolution of the code clones [19].

Due to the dual (both positive and negative) role of code clones in the development and maintenance of software systems, code clones need to be detected and managed effectively to minimize the negative impacts of clones, while maximizing the benefits out of code cloning [20, 22]. The ongoing research towards clone management strives in devising effective clone management tools and techniques from the analysis of clones and their impacts in evolving software systems. However, most of the earlier studies on the analysis of code clones focused on traditional applications written in a particular programming language such as Java, C, C++, and C#. While such studies provide useful insights into clones in those kind of software systems, relatively fewer research aimed to investigate clones in web applications, even though web applications are becoming more ubiquitous these days than in the past.

Unlike the traditional software applications written in a particular language, web applications these days typically have multilingual implementations, where a single source file may contain tangled snippets of source code written in different programming languages. Programmers typically use three or four languages while writing a dynamic web page: HTML, Style code, JavaScript and some kind of server side scripts such as PHP, Active server pages (ASP), or Java server pages (JSP). Programmers often put the scripting code inside HTML tags. Such tangling of Cascaded Style Sheet (CSS), scripting and HTML code in dynamic applications are not very uncommon, specially in legacy systems.

The scattered clones in the tangled multilingual source code of the web applications appear to be more difficult to analyze and manage than those in traditional software systems. Those few studies [11, 12, 13] on the analysis of code clones in web applications mostly analyzed only *Type-1* and *Type-2* clones or the set of clones based on the authors' distinct definitions of code similarity. Moreover, those studies were limited in the investigation of clones only in *static* HTML pages, or in the reference graph derived from the links among different pages and resources.

In this paper, we present an exploratory study on the patterns of both exact (*Type-1*) and near-miss (*Type-2* and *Type-3*) code clones in two industrial web applications, which underwent two different development styles. One was developed using the traditional style where HTML mark-up and PHP code were put together on dynamic web pages. The other was developed following a more sophisticated approach using the MVC (Model-View-Controller) pattern that resulted in a relatively more modularized implementation.

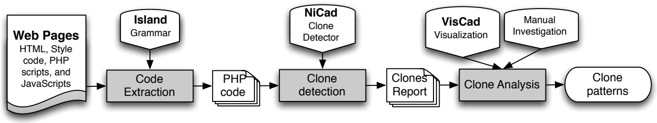


Fig. 1. Procedure of our empirical study

Using island grammar [17], we extracted the PHP code from the tangled source code of the dynamic web pages. Then using the `NiCad` [15] clone detector, we detected clones in the PHP code, and analyzed them with `VisCad` [1], a clone analysis and visualization tool. We investigated the possible instances of different patterns of cloning such as *Forking* and *Templating* [4]. In particular, we aimed to examine whether there is any significant difference in the patterns of cloning in the two web applications that underwent technically two different development approaches. A schematic diagram portraying the procedural steps of our empirical study is presented in Figure 1.

The findings of our study contribute to the ongoing research on clone management. Despite experiencing quite different development styles, the two web applications are almost equally found to have significant number of code clones. However, the code clones in the system developed using the traditional style are more scattered across different locations compared to the dispersion of clones in the other system.

2. BACKGROUND

Code Clones: Two code segments each containing a certain number of contiguous program statements can be regarded as a *clone-pair*, if those two fragments share a certain level of similarity. A set of two or more code fragments are called a *clone-group* if any two members of the set are clone-pairs. The notion of code segment may refer to different levels of granularity, such as functions, blocks, or entire files. In our study, we deal with block clones, where a code fragment refers to a sequence of program statements that span an entire syntactic block (e.g., all PHP statements between a pair of corresponding `<?php` and `?>` tags).

Patterns of Cloning: The *patterns of cloning* characterize the creation and distribution clones in the source code. *Forking* and *Templating* are two prominent patterns of cloning as described by Kapser and Godfrey [4]. In the process of *Forking*, a piece of existing code is cloned to a different location, which is expected to *evolve independently* from its original source. In *Templating*, similar behaviour is implemented by cloning existing code, which are then expected to *evolve together* with any future changes.

3. SUBJECT SYSTEMS

As subjects to our study, we use two industrial web applications: a `Training Registration System (TRS)` and an `Incident Reporting System (IRS)`. The systems are developed at the Information and Communications Technology division of the University of Saskatchewan (UofS), Canada.

Table 1. Subject systems

Subject	Number of		
Systems	.php files	PHP blocks	LOC
`IRS`	75	644	6,848
`TRS`	27	868	5,295

Table 2. `NiCad` settings for clone detection

Parameter	Value
Granularity of Clones	blocks
Minimum Clone Size (LOC)	4, 6, 8, 10, 12, 14, 16, 18, 20
Maximum Clone Size (LOC)	10,000
Filtering of Statements	none
UPI (dissimilarity) Threshold	0%, 10%, 20%, 30%, 40%, 50%
Renaming of Identifiers	blind rename

TRS[1] allows to register in different safety courses offered for the employees and students at UofS. `TRS` is developed using PHP and MySQL. The development of this web application followed the traditional page-based style where HTML mark-ups and PHP code are put together for dynamic web pages. `TRS` also includes JavaScript and CSSs to enhance the client side user experience. The web site has evolved over a period of five years with some recent major changes to add new users and an interface with a desktop client server application. `TRS` represents an incrementally developed dynamic web application.

IRS[2] is a dynamic web application for reporting and managing safety incidents on campus. `IRS` is developed over a period of three years using the PHP Zend framework. The Zend Framework allows development using the MVC pattern. `IRS` has a modular implementation with object-oriented source code well-organized into Controller, Model and View classes.

4. CODE EXTRACTION

As we are interested to investigate the clones in the scripting code written in PHP, we first had to separate the PHP code from the tangled multilingual source code of the dynamic web pages. We used a TXL implementation of island grammar [17] to achieve this. Island grammar can be effectively used to parse source files written in inter-woven multilingual source code [17]. It can separate the target segments (called islands) from the remainder (called water) of a given input file. Since we are interested in the blocks including the entire segment wrapped around by `<?php` and `?>`, which is external to PHP language and part of HTML syntax, we needed to use an island grammar.

5. CLONE DETECTION

For the detection of clones in the extracted PHP code, we used the `NiCad` [15] (version 3.2) clone detector. We wrote several scripts and TXL transformation rules for adapting NiCad to detect clones in PHP code in web applications.

[1] http://www.usask.ca/dhse/trainingcourses

[2] http://lamp.usask.ca/dhse/ZF1.0.1

Table 3. Number of clones and cloned files in two systems with UPI threshold 30% and min. clone size set to 4 LOC

Subject Systems	# of clones	# of cloned files	% of cloned files
IRS	240	30	40%
TRS	212	19	70%

For our study, we carefully set NiCad's parameters as shown in Table 2. With this setting, NiCad detects *Type-1, Type-2,* and *Type-3* block clones (at the granularity of syntactic blocks). As shown in the Table 2, we varied the minimum clone size from 04 through 20 LOC (Lines of Code) at each separate run. Similarly, we also varied the UPI threshold at different values as shown in the table. The UPI threshold (Table 2) is a size-sensitive dissimilarity threshold that guides NiCad in the detection of *Type-3* clones. For example, with the UPI threshold set to 30%, NiCad detects two code fragments as clones if at least 70% of their pretty-printed text lines are the same. The "blind rename" option set for our study, instructs NiCad to to ignore the differences in the names of identifiers/variables during comparison of code segments and thus enables the detection of *Type-2* clones.

6. ANALYSIS AND FINDINGS

From the detection of clones in each of the two subject systems with varying settings of NiCad, we obtained a total of 108 sets (for 2 systems ×9 varieties of minimum clone size ×6 varieties of UPI threshold) of reported clones over the two systems. To analyze these large number of clones, we used VisCad [1], a tool for clone analysis and visualization. We carefully examined the patterns of cloning in both the web applications subjects to our study. In particular, we studied the two categories (i.e., Forking and Templating) of cloning patterns.

In Figure 2, we present the percentages (with respect to the sizes of the systems in terms of LOC) of clone-pairs and clone-groups detected in both IRS and TRS at different UPI (dissimilarity) thresholds and different minimum clone sizes. As we see, for both the systems, the percentages of clone-pairs increases, while the percentages of clone-groups in some cases decreases with the increase in UPI (dissimilarity) threshold. In those cases, with higher dissimilarity threshold, higher number of clone fragments were clustered in larger clone-groups, which resulted in the decrease in the number of clone-groups. Comparing the results of clone detection in the two systems (Figure 2), one can infer that the proportion of clones in TRS is about 10 times lower than that in IRS. Thus, the object-oriented system IRS exhibits a higher clone density, which is also consistent with earlier studies [14, 18] that also reported the object-oriented systems to have higher clone density compared to the systems developed using procedural programming languages.

For both the subject systems, the total number detected clones and cloned files (source files containing one or more cloned fragments) are presented in Table 3. As we see from the table, IRS is found to have slightly higher number of clones than that of TRS. This is in accordance with our expectations, as IRS is object-oriented and larger in size than TRS. Perhaps, due to the sophisticated development approach and modularized architecture, the clones in IRS exist in only 40% of the source files. On the contrary, in TRS, which underwent very traditional development approach, the clone are found scattered over 70% of the source files.

6.1. Observed Patterns

In both the systems, the developers' motivation for cloning appeared to be quite different. In TRS, the programmers reused the code as bunch of reusable program statements. However, most of the clones in IRS appeared due to the used templates of underlying framework. Thus, in most cases, the clones in IRS system are found to be the result of copying of entire classes or functions, as compared to those in TRS, where the programmers copied the reusable code fragments from one web page to another.

The clones in TRS are found to be difficult to manage (e.g., refactoring, merging), as those are unpredictably scattered all around in different web pages located in different directories. However, all the clones detected in IRS are in accordance with predefined templates and directory structures, with a consistent naming convention forced by the underlying framework. Thus, the clones in IRS appears to be relatively more manageable compared to those in TRS.

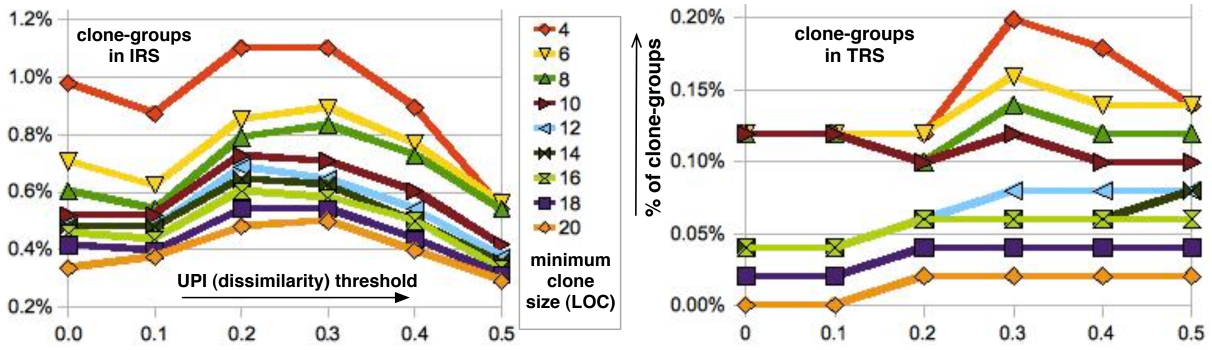
Listing 1. A code fragment in TRS having 68 clones

```
if ($date2 != ‘0000-00-00’) {
  $date2 = getdate(strtotime($date2));
  if ($date1[‘month’] == $date2[‘month’]) {
    $result = $result.‘/’ . $date2[‘mday’];
  } else {
    $result = $result.‘/’ . $date2[‘month’]
                  .‘ ’. $date2[‘mday’];
  }
}
```

6.1.1. Coning Patterns in TRS

As mentioned before, TRS underwent a traditional "page-based" incremental development process, where new web pages were developed to meet the new needs. The source code in each of these web pages includes independent program text often having duplicated code. The relationships among these pages can only be traced by the web site menus, embedded URLs, and use cases. Though the source code is organized into directories pertaining to concerned menus and use cases, we found that many of the clones in TRS were created as a result of shared resources such as menu system, page layouts, authentication code, and GUI forms.

A typical web application development team avoids the duplication of code by configuring the application into functional components, and using shared CSS as well as separate script-files for dynamic code generation to fulfill re-occurring



(a) clone-groups in IRS

(b) clone-groups in TRS

Fig. 2. Percentage of PHP clone-groups in IRS and TRS

requirements of rendering. In TRS, we found signatures of exploiting some of these best practices during the page-based development. Still, NiCad detected 14 clone-groups. In Listing 1, we present an example of code fragment, which is found to have as many as 68 near-miss clones in TRS. Many of these clones are the left-overs from the quick prototypes. At the end of iterations, these prototypes were staged to production with minor changes without considering the possibilities of refactoring and thus caused the existing clones remain intact.

It was interesting to find that the web pages handling higher number of use cases have higher number of clones. This observation is consistent with the programmers' behaviour of reusing the existing code to implement new functionalities. The main page of the web application as well as the web pages inside the Admin module dealing with large number of use cases and functionalities such as the management of class time, students, and courses are found to have relatively higher number of clones compared to other web pages.

In TRS, (that experienced page-based development process), most of the source code is written for HTML rendering and the majority of such code consists of print statements. These print statements often include very complex concatenation of strings encapsulating HTML and scripting code. Due to the lack of support from text editors and IDEs to efficiently write such code, programmers usually rely on the plain text editors, which makes it very tedious to write blocks of such code. Thus, the programmers easily become tempted to reuse such code by copy-pasting and making minor changes. For example, during our manual investigation, we found instances of copy-pasting the header part of an HTML table along with the following loop construct to put data in the table.

In the page-based traditional development of web applications, whenever programmers start working on the new page, their first target often remains to meet the template requirements of the code, session management, and connections with database if required. These are often easily done by copying an entire existing page from the same directory or under the same menu hierarchy. The programmers then start trimming down the code to remove unnecessary code. Sometimes more than the basic skeleton code is kept for some anticipated future use. Signs of such *Templating* practices are commonly found in TRS, which often lead to many clones and some of which are even dead code.

6.1.2. Cloning Pattern in IRS

Recall that, IRS experienced a sophisticated development process and have a modular architecture with object-oriented code. We found many clones in IRS that we believe to have been created due to the necessary library calls to conform the MVC pattern. Many clones are found to have constituted functions implementing of methods of super class, or the functions for initialization of certain program objects and indices in the Controller classes.

A second pattern of cloning in IRS is found in the exception handling code. The code snippet of Listing 2 shows a clone fragment resulting from the programming practice of using simpler code to handle all kinds of program exceptions. Instead of handling context specific exception, a more generic exception is caught. In IRS, we discovered as many as 42 of such clone fragments. A third pattern of cloning in IRS appeared due to the use of very similar code for inserting and updating data in the underlying database. Such code blocks are found as clone-pairs in each of the Controller classes.

Listing 2. Generic exception handling code in IRS having 42 clones

```
catch (Exception $e){
  $db->rollback();
  $this->view-> messages = 'Record not Saved !!';
  echo $e->getMessage();
}
```

With respect to the modules of MVC pattern, the number of clones is found to be much lower in the Model module compared to those in the Controller. In the Controller module, we found 201 clone fragments distributed over 16 source files, whereas in the Model module, only 39 clone segments were found dispersed over 14 source files. The source code inside the Model module are often mapped to database schema. The source code in the Model module requires only localized changes (i.e., changes that do not affect the rest of the system) only when there is a relatively rare change in the database schema. This might be a reason why the Model module exhibited lower number of clones compared to the Controller module.

In addition, the frequent user-centric changes in the View module (i.e., at the interface of the system) result in corresponding changes in the Controller module, which often caused changes in the existing Controller code, the creation of new Controller code, or the creation of new action handlers in the least. During our manual investigation, we found evidences that in such situations, the developers copied the existing code and reused it with necessary customizations, which resulted in many code clones in the Controller module. Similar to TRS, in IRS as well, we found that the Controllers classes handling higher number of use cases also had higher number of clones.

In general, the systems that undergo more usage are more likely to experience changes during the maintenance life cycle. Frequent interaction of a certain user for his or her daily operations, or a large number of users' interaction with a particular portion of the system may result in high usage. These often lead to a large number of feedbacks and enhancement requested posted by the user community. When the developers accommodate those feedbacks and feature requests, they make modifications in the existing code base. These changes often cause the creation of clones by immense *forking* and customization of the existing code snippets. In IRS, a subset of Controller classes such as the Reporter, Short, and Admin classes are found to have experienced extensive interactions from the user community. These Controller classes also underwent frequent changes, and became sources of clones through *forking*. For example, the source code for the Short Controller is a trimmed down clone of the pre-existing Controller class.

From the development history of IRS, we found that the first Controller class developed for the system was the Index class. The rest of the Controller classes were developed by *templating* from the Index class followed by minor customizations. The real center of the universe is the Reporter class. The very first version of Reporter class of the Controller module was handling very detailed input forms. As the department received complaints about those lengthy forms, a change request was initiated to remove some of the fields. This resulted in splitting the Reporter class and *forking* a new Controller class named Short Controller. Later, *templating* from the Reporter class, several new Controller classes were introduced, which were customized for their respective contexts.

Recall that, IRS is developed following the object-oriented paradigm. We found evidences implying that the development process of IRS employs strict rules for the implantation of separate classes for Controller, Model and View modules. It was interesting not to find any code clones across the triplet of Model, View, and Controller modules. However, duplicated blocks of code are found within the source code of the individual modules. Although IRS has a modular architecture built through a framework-based sophisticated development process, the system still has a fair number of code clones, in fact, a higher density of clones compared to TRS.

7. VALIDITY

The *internal validity* of the findings of our study depends on the accuracy of clone detection. However, the NiCad clone detector used in our work is reported to be effective in detecting exact (*Type- 1*) and near-miss (*Type-2* and *Type-3*) clones with high precision and recall [15, 16]. Nevertheless, we manually verified most of the detected clones, as we performed extensive subjective analysis with the help of VisCad.

Our manual efforts and subjective judgment can be subject to human errors. During clone analysis, the tool support from VisCad helped us to keep the probable human errors to the minimum. Moreover, as the first author of this study is also a developer of the systems, we have been able to capture facts (e.g., contextual information about the situations and intentions of clone creation) that were not possible to extract from the analysis of source code only. Indeed, the findings of our study are based on only two industrial systems, and thus can be argued against their *generalizability*.

8. RELATED WORK

Not many works have been done towards the investigation of code clones in web applications. Lucca et al. [6] aimed to detect of similar static web pages by pairwise computation of Levenstein distance between tag-sequences produced from serialization of HTML tags. In a follow up work [7], they attempted to detect clones in ASP pages by serializing references to ASP objects. Lucia et al. [8, 9] proposed a graph-based method to identify clones in the navigational patterns in web applications. An extension of their technique was realized in a tool [10]. However, our study is different from theirs in the fact that we investigate cloning patterns in the PHP scripts, whereas their work was towards the detection and analysis of clones in the navigational patterns. Our work fundamentally differs from those. Instead of detecting similarities in entire web pages or navigational patterns, we investigated the patterns of cloning in the scripting code embedded in dynamic web applications.

Lanubile and Mallardo [5] applied a metric based approach to detect function level clones in the JavaScript and VBScript code in three web based systems. Rajapakse and Jarzabek [12] reported a cloning rate of up to 63% in web applications. In a later study [13], they investigated the effectiveness of clone unification using Server Pages technique to reduce the number of clones. Our work significantly differs from those studies in a number of ways. The metric based clone detection technique used in the study of Lanubile and Mallardo [5] might have reported many false positives and missed many potential clones, as the metric based techniques are known to have low accuracy in clone detection [16, 22]. The studies of Rajapakse and Jarzabek [12, 13] were only on *Type-1* and *Type-2* clones. However, in our study, we use a hybrid clone detector, NiCad [15], which was reported to have high accuracy in detecting both exact (*Type-1*) and near-miss (*Type-2* and *Type-3*) clones [15, 16]. Moreover, the objective of Lanubile and Mallardo [5] was on the detection of clones in JavaScript and VBScript code, whereas Rajapakse and Jarzabek [12, 13] focused on the existence of clones and the effectiveness of Server Pages technique in clone minimization. On the contrary, our work is on the *analysis* of cloning patterns in PHP scripts in web applications that underwent distinct development processes.

Similar to ours, the work of Mao et al. [11] also used TXL [2] implementation of island grammar [17] to extract style information from HTML files. Then they identified the exact (*Type-1*) clones in the style code segments by pairwise comparison. The objective of their work was to convert the table-based layout websites to standards-compliant modern CSS stylesheet-based websites. Our work, is orthogonal to theirs. We use island grammar to PHP code from the dynamic web applications, and then we detect and analyze clones in the scripting code.

9. CONCLUSION

In this paper, we have presented an empirical study on the exact (*Type-1*) and near-miss (*Type-2* and *Type-3*) clones in two industrial dynamic web applications, which have architecturally different implementations developed through two different development processes. Applying island grammar, we extracted the scripting code written in PHP, and detected clones in them with a state-of-the-art clone detection tool. Then we carried out an in-depth analysis on the cloning patterns (i.e., *Forking* and *Templating*) in the systems.

The findings of our study confirm the earlier reported reasons for cloning and inform clone management in the context of web applications. We found that, despite the architectural differences and development styles, both the web applications had significant number of code clones. However, the system developed using the traditional page-based approach have the clones very scattered over different areas of the code base. On the contrary, the modular implementation following the MVC pattern resulted in relatively less scattered clones in the other system. The dispersion of clones in the later system was dictated by the underlying framework, and those clones appeared to be relatively easier to manage compared to those in the other system.

In both the systems, we found many footprints of *Forking* and *Templating* cloning patterns. However, compared to a traditional desktop application written in a single programming language such as Java or C++, the development and maintenance of the web applications need to handle an additional dimension of difficulty, since the web pages are composed of inter-woven source code written in more than one languages (e.g., HTML, PHP, CSS, JavaScript). Due to the lack of sufficient support for writing such tangled code, developers make extensive code reuse by copy-pasting and result in many clones in the code base. Tracing and managing those clones in the web applications demand specialized tool support that should be able to perform language specific separation of code from the tangled source code, and handle them separately while still preserving the inter-language relationships of the code clones.

Acknowledgement: This works is supported in part by the Walter C. Sumner Memorial Foundation.

10. REFERENCES

[1] M. Asaduzzaman and C. Roy, “VisCad: flexible code clone analysis support for NiCad,” In *IWSC*, pp. 77–78, 2011.

[2] J. Cordy, “Source transformation, analysis and generation in TXL,” In *PEPM*, pp. 1–11, 2006.

[3] S. Jarzabek and L. Shubiao, “Eliminating Redundancies with a “Composition with Adaptation” Meta-programming Technique,” *SIGSOFT Softw. Eng. Notes*, 28(5):237–246, 2003.

[4] C. Kapser and M. Godfrey, ““Cloning considered harmful” considered harmful: patterns of cloning in software,” *Empirical Softw. Eng.*, 13:645–692, 2008.

[5] F. Lanubile and T. Mallardo, “Finding function clones in web applications,” In *CSMR*, pp. 379–386, 2003.

[6] G. Lucca, M. Di Penta, and A. Fasolino, “Clone analysis in the web era: An approach to identify cloned web,” In *WESS*, pp. 107–113, 2001.

[7] G. Lucca, M. Di Penta, and A. Fasolino, “An Approach to Identify Duplicated Web Pages,” In *COMPSAC*, pp. 481–486, 2002.

[8] A. Lucia, R. Francese, G. Scanniello, and G. Tortora, “Reengineering web applications based on cloned pattern analysis,” In *IWPC*, pp. 132 – 141, 2004.

[9] A. Lucia, G. Scanniello, G. Tortora, “Identifying Clones in Dynamic Web Sites Using Similarity thresholds,” In *ICEIS*, pp. 391–396, 2004.

[10] A. Lucia, R. Francese, G. Scanniello, G. Tortora, “Understanding Cloned Patterns in Web Applications,” In *IWPC*, pp. 333–336, 2005.

[11] A. Mao, J. Cordy, and T. Dean, “Automated conversion of table-based websites to structured stylesheets using table recognition and clone detection,” In *CASCON*, pp. 12–26, 2007.

[12] D. Rajapakse, “An investigation of cloning in web applications,” In *WWW*, pp. 924–925, 2005.

[13] D. Rajapakse and S. Jarzabek, “Using server pp. to unify clones in web applications: A trade-off analysis,” In *ICSE*, pp. 116–126, 2007.

[14] C. Roy and J. Cordy, “Near-miss function clones in open source software: an empirical study,” *J. Softw. Maint. Evol.*, 22(3):165–189, 2010.

[15] C. Roy and J. Cordy, “NiCad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization,” In *ICPC*, pp. 172–181, 2008.

[16] C. Roy, J. Cordy, and R. Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,” *Science of Computer Programming*, 74(7):470–495, 2009.

[17] N. Synytskyy and J. Cordy, “Robust multilingual parsing using island grammars,” In *CASCON*, pp. 266–278, 2003.

[18] M. Zibran, R. Saha, M. Asaduzzaman, and C. Roy, “Analyzing and Forecasting Near-miss Clones in Evolving Software: An Empirical Study,” In *ICECCS*, pp. 295–304, 2011.

[19] M. Zibran and C. Roy, “A Constraint Programming Approach to Conflict-aware Optimal Scheduling of Prioritized Code Clone Refactoring,” In *SCAM*, pp. 105–114, 2011.

[20] M. Zibran and C. Roy, “Towards flexible code clone detection, management, and refactoring in IDE,” In *IWSC*, pp. 75–76, 2011.

[21] M. Zibran and C. Roy, “IDE-based real-time focused search for near-miss clones,” In *SAC*, pp. 1235–1242, 2012.

[22] M. Zibran and C. Roy, “The Road to Software Clone Management: A Survey,” *Tech. Report 2012-03*, Department of Computer Science, University of Saskatchewan, Canada, pp. 1–62, 2012.