IDENTIFYING CLONES IN DYNAMIC WEB SITES USING SIMILARITY THRESHOLDS

Andrea De Lucia, Giuseppe Scanniello, and Genoveffa Tortora
Dipartimento di Matematica ed Informatica - Università di Salerno - Via S. Allende - 84081 Baronissi (SA)

Keywords: Dynamic Web site maintenance, Web engineering, Web site analysis, source code clone analysis

Abstract: *We propose an approach to automatically detect duplicated pages in dynamic Web sites and on the analysis of both the page structure, implemented by specific sequences of HTML tags, and the displayed content. In addition, for each pair of dynamic pages we also consider the similarity degree of their scripting code. The similarity degree of two pages is computed using different similarity metrics for the different parts of a web page based on the Levenshtein string edit distance. We have implemented a prototype to automate the clone detection process on web applications developed using JSP technology and used it to validate our approach in a case study.*

1 INTRODUCTION

In the recent years the industry and academic researchers have been showing great interests in the engineering, maintenance, reverse engineering, restructuring, and reuse of Web sites and applications (Bieber & Isakowitz, 1995; Conallen, 2000; Ginige & Murugesan, 2001; Ricca & Tonella, 2001; Aversano *et al.*, 2001). Similarly to legacy systems, Web applications are subject to continuous evolution (Boldyreff *et al.*, 1999): internal and external factors generate new or modified system requirements, so they inevitably changes. Moreover, Web applications change much more quickly than traditional software systems; therefore, their maintenance absorbs considerable efforts if developers do not use methodologies that anticipate changes and evolutions (Bieber & Isakowitz, 1995; Ginige & Murugesan, 2001; Conallen, 2000).

Unfortunately, the current state of practice of Web application development is far from using consolidated methodologies; Web applications are typically obtained by reusing the fragments of existing pages and without explicit documentation. This approach augments the code complexity and the effort to test, maintain and evolve these applications. Although in some case this is done in a disciplined way (Aversano *et al.*, 2001), the main problem is the proliferation of duplicated Web pages.

Code cloning is one of the factors that make software maintenance more difficult (Baker, 1995; Balazinska *et al.*, 1999; Baxter *et al.*, 1998; Kamiya *et al.*, 2002). A code clone is a code portion in source files that is identical or similar to another. It is common opinion that code clones make the source files very hard to modify consistently. Clones are introduced for various reasons such as lack of a good design, fuzzy requirements, undisciplined maintenance and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste. Thus, code clone detection can effectively support the improvement of the quality of a software system during software maintenance and evolution.

Recently, researchers have extensively studied clone detection for static Web pages, written in HTML, or dynamic ones, written in ASP (Di Lucca *et al.*, 2002; Lanubile & Mallardo, 2003). Lanubile and Mallardo (2003) extend the metric-based approach and the classification schema proposed by Balazinska *et al.* (1999). Their approach exploits a pattern matching algorithm to compare scripting code fragments and is based on two steps: automatic selection of potential function clones based on homonym functions and size measures and visual inspection of selected script functions. This approach does not consider the problem of identifying cloned pages, as it does not take into account neither the structure of web pages nor their content.

Different authors have tackled the problem of identifying cloned web pages. Di Lucca *et al.* (2002) encode the sequences of tags of HTML and ASP pages into strings and identify pairs of cloned pages

De Lucia A., Scanniello G. and Tortora G. (2004).
IDENTIFYING CLONES IN DYNAMIC WEB SITES USING SIMILARITY THRESHOLDS.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 391-396
DOI: 10.5220/0002597303910396

at structural level by computing the Levenshtein string edit distance (Levenshtein, 1966) between the corresponding strings. Pages are considered clones if their Levenshtein distance is zero. They also use metrics, such as lines of code and complexity metrics, to identify clones at the scripting code level. Ricca and Tonella (2003) enhance the approach based on Levenshtein distance with hierarchical clustering techniques to identify clusters of duplicated or similar pages to be generalized into a dynamic page. Unlike the approach proposed by Di Lucca *et al.* (2002), the distance of cloned pages belonging to the same cluster is not zero. Each page is initially inserted into a different cluster and at each step clusters with minimal distance are merged thus producing a hierarchy of clusters. The clusters of cloned pages are selected by the software engineer, by choosing a cut level in the hierarchy.

In this paper we propose an approach to automatically detect duplicated pages in dynamic and/or static Web sites. Similarly to previous approaches (Di Lucca *et al.*, 2002; Ricca and Tonella, 2003) we also use Levenshtein string edit distance as basic metric to identify clones. Unlike previous approaches, besides the distance at structural level, we also consider the distance of cloned pages at the content and scripting code levels. In particular, starting from the Levenshtein edit distance, we define similarity measures for structural, content, and scripting code of two dynamic pages and use thresholds on the similarity measures to establish whether two pages are clones.

The approach has been tailored for web applications developed using JSP technology, differently from the approaches above. However, it can be also used on web applications developed using other technologies, such as ASP. We have implemented a prototype to automate the clone detection process. The tool compares pairs of HTML and JSP pages and computes the similarity degree of each pair. The prototype enables independent tuning of the thresholds for the similarity metrics of structure, content, and scripting code of pairs of JSP pages; this feature enables to adapt the tool to different contexts and to refine results on particular subsets of a Web application. We have assessed the validity of the approach and the tool by analyzing the official Web site of the 14th International Conference of Software Engineering and Knowledge Engineering (SEKE, 2002).

The paper is organized as follows: in Section 2 we present the clone detection method, while in Section 3 we describe the system prototype and its architecture. The case study and the experimental results are discussed in Section 4. Finally, we give conclusions and future works in Section 5.

2 CLONE DETECTION

We propose three classification levels for clone analysis in dynamic web pages:

Structure - the basic property which is exploited to determine the matching of two pages is the syntactic structure, defined by the HTML tags.

Content – at this level also the text information associated with HTML tags of two web pages is compared.

Scripting code – at this level the server-side and/or the client/side scripting code of two web pages is compared. It is worth noting that this comparison can range from a simple string matching (in this case this is similar to content level clone analysis) to a more complex syntactic or semantic pattern matching.

The first two levels also apply to HTML pages, while the scripting code level is only used for dynamic web pages.

2.1 The Clone Identification Process

Figure 1 shows the underlying process of our clone analysis approach. Although our approach is general, in the following we will refer to dynamic web sites developed using JSP technology. In the first phase of the process the files of a web application are separated in two subsets: JSP and HTML pages. This step is performed because scripting level clone analysis is only performed on the JSP pages.

The HTML parsing produces string representations for the HTML pages corresponding to their structure and content. The strings are produced visiting the abstract syntax tree of the pages. For the JSP pages the process is similar, but also string representations of the java code is produced. There are basically two ways to produce the string representations for structure, content, and scripting code of a web page. The first way, used in the current implementation of the prototype, computes separate strings for the different clone analysis levels. In this way, the comparison of the different strings at the different levels can be performed independently of the other levels.

An alternative to this approach consists of linking each content and java code string to the corresponding HTML/JSP tag. In this case, the content and java source code fragments of corresponding tags in the structure can be compared while performing the structure level analysis. A disadvantage of this approach is that we are not able to identify content and java source code similarity, whenever the similarity degree at the structure level

of the two pages is low. For example, two web pages could have the same java code distributed in different ways across the page structure.

The produced strings are used to compute the similarity degree of two web pages at structure, content and scripting code level. The similarity degree is based on the computation of the Levenshtein string edit distance (Levenshtein, 1966) and its comparison with a threshold. The thresholds are dynamically computed based on the size of the two strings and can be separately tuned for the different clone analysis levels.

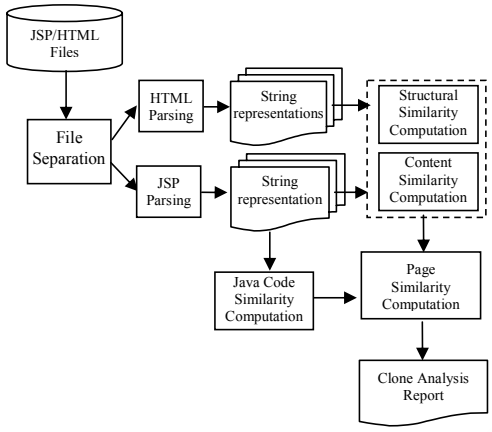


Figure 1: The clone identification process

2.2 Levenshtein Distance Model and Similarity Relation

The similarity degree of the strings encoding structure, content and java code of a web page is based on the *Levenshtein edit distance model* (Levenshtein, 1966), which is one of most important models for string matching. The Levenshtein distance is defined as the minimum cost required to align two strings: given two strings x and y the Levenshtein distance is defined as the minimum number of insert, delete, and replace operations required to transform x into y. Generally, the approach assumes that the insert and delete operations have cost 1, while the replacement has cost 2 (it is equivalent to a sequence of delete and insert operations). We use these costs as default values, although our approach can be parameterized with respect to the costs of the operations.

Whenever the Levenshtein distance is 0 the pairs of strings constitute perfect or identical clones (Di Lucca *et al.*, 2002). However, perfect clones in web applications are rare; more interesting is the case of nearly-identical or similar clones (Ricca and Tonella, 2003). In our approach two web pages can be considered clones if the distance of the corresponding strings is lower than a given threshold t. This threshold can be dynamically computed from the minimum percentage of similarity p required to consider the two strings as clones and the maximum of the length of the two strings. Therefore, given two strings x and y, let $D(x, y)$ be their distance and $S(x, y)$ be their similarity degree. The two strings are clones if $S(x, y) \geq p$, or equivalently $D(x, y) \leq t$, where $t = f(p, maxlength(x, y))$.

In the current implementation, the threshold is computed in the following way:

$$t = 2\, maxlength(x, y)\,(1 - p)$$

It is worth noting that whenever p is 1 (100% minimum similarity required) the clone analysis will only identify perfect clones.

3 PROTOTYPE

The effectiveness of a clone detection process may significantly depend on the tools used to support it. The proposed prototype has been implemented in Java, and integrates several software components needed during the clone analysis of a web application. The system architecture is shown in Figure 2, its main modules are the Graphical User Interface (GUI) and the Engine.

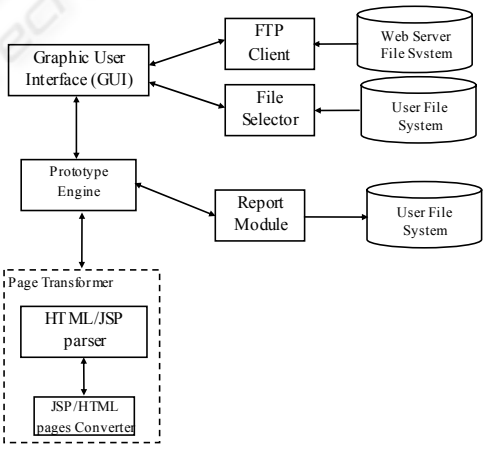


Figure 2: Prototype architecture.

The GUI enables the user to select the files of the Web site and display the results of the clone analysis. Web site files can be locally or remotely stored. When the files are remotely stored on the Web Server, the prototype establishes a FTP connection with the remote file system, through the FTP client module.

The Engine implements the core clone analysis functionality and includes the computation of the

Levenshtein distance. It partitions the files into two sets of HTML and JSP pages to be analyzed by the Page Transformer to produce the strings encoding the page structure, content, and the scripting code. The Engine can be configured to enable the comparison of HTML and JSP pages together, by excluding the analysis of the java source code.

The parser has been implemented by modifying the open source HTML parser written in Java (HTMLParser ver. 1.2), available under GPL license from http://sourceforge.net/projects/htmlparser. This software component has a flexible architecture, which has allowed us to easily enhance its functionality and configure it. The parser is a very important component of the prototype; it has a double function: analyzing the page correctness and retrieving the information useful for the approach. For optimization reasons, the page structure is encoded into a string constructed on a different alphabet than the HTML/JSP tags. Indeed, the computation of the Levenshtein algorithm on strings of HTML/JSP tags would have been very expensive and in some cases impracticable. Thus we have defined and used a different alphabet, where each alphabet symbol codifies an HTML/JSP tag.

The Engine identifies the clones by computing the Levenshtein edit distance of each pairs of strings encoding the structure, the content, and the java code, and using different dynamic thresholds for different clone analysis levels, as defined in the previous section. The thresholds are computed according to the minimum similarity percentage required, chosen by the user through the GUI.

The Report Module enables to assess and analyze the results of the clone analysis. The information concerning the analyzed pages and the identified clones are stored into files; these results can be also visualized through the GUI.

4 CASE STUDY

The method and tool presented in the previous sections have been validated in a case study. We have used the official site of SEKE 2002, the fourteenth International Conference on Software Engineering and Knowledge Engineering (http://www.scienzemfn.unisa.it/seke/index.html), a web application developed in JSP and used to support the organizers and the academic community for paper submission, refereeing, and conference registration. Two joint workshops on Web Engineering and Software Engineering Decision Support, respectively, also used this application. As our research group periodically organizes scientific conferences, we are going to reuse, and enhance the SEKE Web application. To this aim, it has been necessary to analyze and restructure the system.

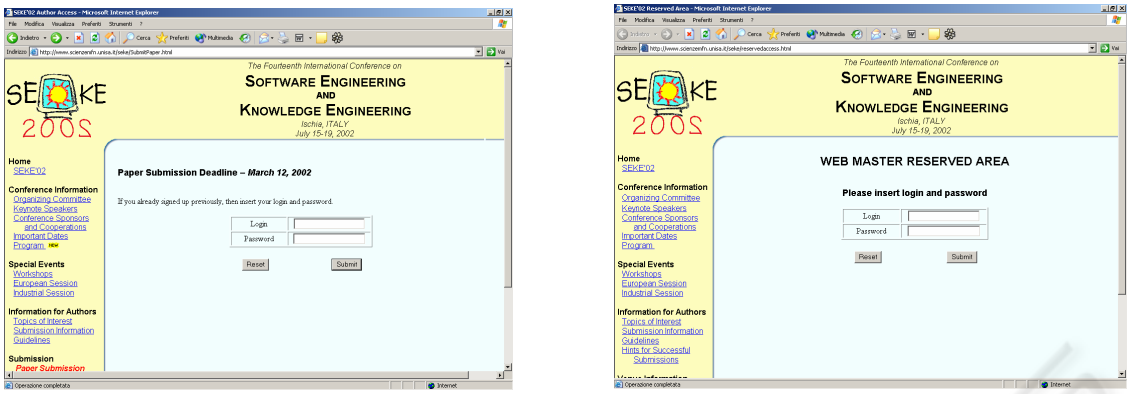
The case study consists of 157 files distributed in one directory. The static pages are implemented by 52 files with html extension, while 107 files with extension jsp implement the dynamic pages. The site contained some other files, such as images, a Java applet, java classes, logos, etc., which were used in the analyzed pages. The files that implement the static and dynamic behavior of the SEKE site were stored in a single folder, so the pages have been grouped without any meaningful classification.

The Web application was grown in an uncontrolled way. The main reason, for this was the lack of software requirements and development methodology. In fact, the requirements were incomplete and fuzzy. The development process was not documented, so the design rationales were not evidenced. Moreover, no configuration and versioning management was used during the development of the Web application. These factors have produced dead code, i.e. some unreachable pages that were sometimes also grouped. As a result, the general navigational schema of the SEKE Web application appears as set of disconnected graphs. Typically, each sub-graph represents an event of the conference, for example paper submission deadline, acceptance notification, or camera-ready paper due.

The first step of the case study consisted of tuning the similarity thresholds. We have tried different thresholds for the HTML and JSP pages and evaluated the obtained results using two well known metrics of the information retrieval and reverse engineering field, namely precision and recall. In our case, the recall is the ratio between the number of actual pairs of cloned pages identified by the tool over the total number of actual pairs of cloned pages in the web application; the precision is the number of actual pairs of cloned pages identified by the tool over the total number of identified pairs. Some results of this tuning step are shown in Table 1 and Table 2 for HTML and JSP pages, respectively, and discussed in the following subsections.

4.1 HTML pages

For the HTML pages the best results have been achieved using 90% as threshold for structural similarity. The prototype detected 229 pairs of structural clones. The number of pairs of clones is so high because the developers have used the same graphical layout for each page. We have applied different thresholds for content similarity to the results achieved with the structural similarity threshold above. This has been used to identify



(a) (b)

Figure 3: A pair of HTML cloned pages

Table 1: Precision and recall for HTML pages.

Structure	*Content*	*Recall-Precision for Structure*	*Recall-Precision for Content*
75%	70%	100% - 25%	85% - 35%
90%	60%	95% - 75%	95% - 25%
90%	80%	95% - 75%	60% - 70%
95%	70%	90% - 80%	65% - 65%

Table 2: Precision and recall values on JSP pages

Structure	*Content*	*Code*	*Recall-Precision for Structure*	*Recall-Precision for Contents*	*Recall-Precision for Java Code*
90%	80%	80%	90% - 43%	90% - 60%	65% - 82%
90%	90%	80%	90% - 43%	80% - 65%	65% - 82%
95%	80%	80%	60% - 80%	70% - 70%	60% - 85%
95%	90%	95%	60% - 80%	65% - 85%	50% - 95%

groups of pages with different content similarity. The minimum content similarity threshold that maintains the same 229 pairs recovered with the structural similarity analysis is 50%. This means that all recovered pages have at least 50% content similarity. In our case this was due to the fact that all pages have a column including the same navigational menu on the left side and the conference name and logo on the top (see the two sample pages in Figure 3). It is worth noting that the precision values for the content similarity are not very high. This is normal as in HTML pages we do not expect to find many pages with the same content. However, using higher thresholds for the content similarity, we were able to prune pairs of pages with more differences in the text content in the middle area of the page. Pairs of pages recovered with higher thresholds have very little text in the middle or are very similar (nearly perfect clones). For example, the two pages in Figure 3 are still recovered with a 97% structural similarity threshold and content similarity 95% threshold. This means that pages with high content similarity are also likely to have high structural similarity and therefore content similarity can help to improve the results achieved with structural similarity towards the identification of nearly perfect clones.

4.2 JSP Pages

On the other hand, the results for structural similarity for the JSP pages are not so good as for the HTML pages. The best results are achieved between 90% and 95% thresholds. However, in this case we are not able to achieve high values of recall and precision with the same threshold. This is because the web application includes pages with a very structured layout and pages with very poor layout, developed to enable the organization to achieve statistics about the conference. In the case of pages with very structured layout the number of tags of the middle area is marginal with respect to the number of tags of the layout. For these pages both precision and recall are very high even with high structural similarity thresholds. On the other hand, small differences in the tags of two pages with poor layout results in lower structural similarity values. This means that higher recall values are achieved with lower thresholds, but this also results in lower precision values.

In the case of JSP pages most of the content is dynamically generated and very few is included in the pages. This justifies the better results of precision and recall for content similarity for the JSP pages with respect to the HTML pages. The results for the clone analysis at the code level are also very interesting. Very good precision values are achieved using high code similarity thresholds, still maintaining good recall values. Better results might be achieved using a different approach to code clone analysis than the Levenshtein string edit distance.

The clone analysis results have demonstrated that there are some cases of nearly identical pages. Therefore, we used higher thresholds to identify the nearly identical pages, in particular 100% for structural and content similarity and 99% for scripting code similarity. The results were 15 pairs of clones corresponding to 22 different files. We

tried to understand the real difference between each pair of cloned pages and we have observed that this difference only consisted on their SQL code. Moreover, the references to the database tables often were the only difference in the SQL queries. It is worth noting that the file names of these pages are also very similar, meaning that there are groups of different pages that actually implement the same or very similar functionality for the different events (main conference and workshops). Indeed, these files might be clustered into groups of clones in a similar way to the approach by Di Lucca *et al.* (2002). In general, clustering is possible if the similarity threshold required is very high, otherwise our experience has demonstrated that clustering does not produce meaningful results.

5 CONCLUSION

In this paper an approach to clone analysis for Web applications has been proposed together with a prototype implementation for JSP web pages. Our approach analyzes the page structure, implemented by specific sequences of HTML tags, and the content displayed for both dynamic and static pages. Moreover, for a pair of JSP pages we also consider the similarity degree of their java source. The similarity degree can be adapted and tuned in a simple way for different web applications.

We have reported the results of applying our approach and tool in a case study. The results have confirmed that the lack of analysis and design of the Web application has effect on the duplication of the pages. In particular, these results allowed us to identify some common features for the SEKE conference and the collocated workshops that could be integrated, by deleting the duplications. Moreover, the clone analysis of the JSP pages enabled to acquire information to improve the general quality and conceptual/design of the database of the web application. Indeed, we plan to exploit the results of the clone analysis method to support web application reengineering activities (Antoniol *et al.*, 2000).

REFERENCES

Antoniol, G., Canfora, G., Casazza, G., and De Lucia, A., 2000. Web Site Reengineering using RMM. *Proc. of International Workshop on Web Site Evolution*, Zurich, Switzerland, pp. 9-16.

Aversano, L., Canfora, G., De Lucia, A., and Gallucci, P., 2001. Web Site Reuse: Cloning and Adapting. *Proc. of 3rd International Workshop on Web Site Evolution*, Florence, Italy, IEEE CS Press, pp. 107-111.

Baker, B. S., 1995. On finding duplication and near duplication in large software systems. *Proc. of 2nd Working Conference on Reverse Engineering*, Toronto, Canada, IEEE CS Press, pp 86-95.

Balazinska, M., Merlo, E., Dangenais, M., Lague, B. and Kontogiannis, K., 1999. Measuring Clone Based Reengineering Opportunities. *Proc. of 6th International Symposium on Software Metrics*, Boca Raton, Florida, IEEE CS Press, pp. 292-303.

Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M., and Bier, L., 1998. Clone Detection Using Abstract Syntax Trees, *Proc. of International Conference on Software Maintenance*, IEEE CS Press, pp. 368-377.

Bieber, M. and Isakowitz, T., 1995. Special issue on Designing Hypermedia Applications. *Communications of the ACM*, 38(8).

Boldyreff, C., Munro, M., and Warren, P., 1999. The evolution of websites. *Proc. of 7th International Workshop on Program Comprehension*, Pittsburgh, Pennsylvania, IEEE CS Press, pp. 178-185.

Conallen, J., 2000. *Building Web application with UML*, Addison Wesley.

Di Lucca, G. A., Di Penta, M., and Fasolino, A. R., 2002. An Approach to Identify Duplicated Web Pages. *Proc. of 26th Annual International Computer Software and Application Conference (COMPSAC'02)*, Oxford, UK, IEEE CS Press, pp. 481-486.

Ginige, A. and Murugesan, S., (eds.) 2001. Special issue on Web Engineering. *IEEE Multimedia*, 8(1-2).

Kamiya, T., Kusumoto, S., and Inoue, K., 2002. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. *IEEE Transactions on Software Engineering*, 28(7), pp. 654-670.

Lanubile, F. and Mallardo, T., 2003. Finding Function Clones in Web Application. *In Proc. of 7thEuropean Conference on Software Maintenance and Reengineering*, Benevento, Italy, IEEE CS Press, pp. 379-386.

Levenshtein, V. L., 1966. Binary codes capable of correcting deletions, insertions, and reversals, *Cybernetics and Control Theory*, 10, 707-710.

Ricca, F. and Tonella, P., 2001. Understanding and Restructuring Web Sites with ReWeb. *IEEE Multimedia*, 8(2), 40-51.

Ricca, F. and Tonella, P., 2003. Using Clustering to Support the Migration from Static to Dynamic Web Pages. *Proc. of 11th International Workshop on Program Comprehension*, Portland, Oregon, IEEE CS Press, pp. 207-216.