Finding Function Clones in Web Applications

Filippo Lanubile
Dipartimento di Informatica
University of Bari
Bari, Italy
lanubile@di.uniba.it

Teresa Mallardo
RCOST – Research Center on Software Technology
University of Sannio
Benevento, Italy
mallardo@unisannio.it

Abstract

Many web applications use a mixture of HTML and scripting language code as the front-end to business services. Analogously to traditional applications, redundant code is introduced by copy-and-paste practices. Code duplication is a pathological form of software reuse because of its effects on the maintenance of large software systems. This paper describes how a simple semi-automated approach can be used to identity cloned functions within scripting code of web applications. The results obtained from applying our approach to three web applications show that the approach is useful for a fast selection of script function clones, and can be applied to prevent clone spreading or to remove redundant scripting code.

1. Introduction

Code duplication occurs frequently during the development and evolution of large software systems. This ad-hoc form of reuse consists in copying, and eventually modifying, a block of existing code that implements a piece of required functionality. Duplicated blocks are called *clones* and the act of copying, including slight modifications, is said *cloning*. When entire functions are copied rather than fragments, duplicated functions are called *function clones*.

Cloning mainly occurs because programmers find cheaper and quicker using the copy-and-paste feature of their editors than writing instructions from scratch or applying correct reuse mechanisms, based on invocation or inclusion. Furthermore, using code size to measure programmers' productivity, and then to assess their performance, can encourage cloning.

Cloning is considered a pathological form of code reuse because of its negative effects on software maintenance and evolution. When a failure is discovered during testing or normal operations, the underlying fault might be duplicated together with clones, thus multiplying the cost of rework. In general, when clones are affected by a modification, applying a change is more expensive and error-prone because of the larger impact and possible side effects. Another undesirable effect is that redundancy increases artificially the size of a software system, thus complicating program comprehension.

Researchers have extensively studied cloning detection for being applied to procedural programs [1, 2, 5, 8, 12, 14] as well as object-oriented programs [3, 4, 8, 9, 17]. Identifying software clones for the purpose of prevention or removal can help to resist at "software aging" [16], where even small changes become very difficult to apply.

Recently, clone identification has been proposed for static web documents [7], written in HTML. However, modern web applications are a mixture of HTML and scripting language code, where scripts can run as event handlers on the client-side or perform HTTP processing on the server-side.

Scripts come embedded in *client pages*, i.e., documents available to browsers, as the content of the *script* HTML element, or they are retrieved from include files (containing pure scripting code with no HTML) using the *src* attribute (of the *script* HTML element) in every client page that needs them. The vast majority of client scripts are written with JavaScript. On the server side, the enabling technologies are more varied depending more on the vendors than on standards. A major approach to server-side processing of HTTP requests is using *server pages*, i.e., documents containing HTML annotated with server-side interpreted scripts. Many scripting languages are supported and representative examples are Java Server Pages (JSP), Microsoft's Active Server Pages (ASP), and PHP.

Web applications evolved from web sites by adding business functionality [6]. A web application uses web technologies as the front-end to business services for easy of deployment and minimal client configuration. The phenomenon of duplicated code is even worse for this class of applications because many people have erroneously thought that software engineering principles and methods do not apply to web development [18].

This paper presents a simple approach to cloning detection for HTML scripting (i.e., HTML annotated with scripting code) that characterizes both client and server web pages. The approach is partly automated by a tool that we developed to identify suspect duplication of scripting code in web applications.

We assessed the potential benefits of our approach with respect to three different web applications. The experimental results show that the approach is useful for a fast selection of function clones in web applications and might be applied for the goal of verification, to prevent clone spreading, or for the goal of refactoring [10], to remove redundancy and shrink application size.

Section 2 presents our approach for function clone identification in scripted web pages. Section 3 introduces the case studies and reports the experimental results. Finally, Section 4 provides conclusions and directions for future work.

2. Our approach to detect cloned script functions

In Mayrand et al. [14] the strategy for identifying function clones is based on four points of comparison: function name, layout metrics, expression metrics, and control flow metrics. Any pair of functions is then compared with respect to these characteristics to define an ordinal scale of cloning, based on degree of similarity between function clone pairs. Based on an empirical validation of the classification, they conclude that the first two classification levels are reliable while the others result in a high number of false positives. The first class of this taxonomy, *ExactCopy*, requires that the function names are identical as well as metric values. The second class, *DistinctName*, has the same requirements of ExactCopy except for the names of functions which are different; it assumes that function cloning occurs by renaming the function to avoid name conflicts in a module.

Starting from the previous classification, Balazinska et al. [4] develop a different classification to adapt the cloning cases to object-oriented programs written in Java. The eighteen categories are based on the degree of effort required to remove the duplication. The metrics-based approach is limited to a pre-processing stage for reducing the search space, and then a pattern matching algorithm is applied to compare code fragments.

Analogously to [4], we present a specific classification of function cloning in scripting code, based on the examination of existing web applications. We noticed that duplications do not affect function names, and most of the times script functions are copied without any change (we remind that in our context, cloning is motivated by reuse rather than plagiarism). When script functions are not identical, most differences consist in changes to indentation, comments, blank lines, and variable names. Sometimes these differences are negligible with respect to duplication removal, other times parameters should be added to functions in order to account for name variability. In the remaining cases, even if function names are equal, source code is so different that refactoring is not worth the effort.

These observations are reflected in a classification schema for pair comparison of cloned script functions, shown in Table 1.

Table 1. Classification for cloned script functions

Level	Name	Description
1	Identical	An exactly identical copy.
2	Nearly-identical	An almost identical copy that might be discarded. The original function does not need any change to be invoked in place of the copy.
3	Similar	A functionally equivalent copy that differs for variable names or constants. The copies, including the original one, might be replaced by a new function with the same variable names or the addition of parameters.
4	Distinct	A false positive or a copy with different expressions and different control flow.

Our approach to detect cloned script functions is based on two main steps:

1. Automatic selection of potential function clones
2. Visual inspection of selected script functions

The first step is performed by a tool, called eMetrics, which we developed at University of Bari. The tool analyzes web applications that adopt HTML and Microsoft's ASP technology.

Besides measuring the size of a web application to different granularity levels (including script functions), the tool selects homonym programmer-defined functions (written in JavaScript or VBScript) as potential cloned functions. Homonymy is defined by the equality of identifiers that are used as function names in the declaration of script functions. The comparison of function names is case insensitive (JavaScript is a case-sensitive language but VBScript is not) and does not take into account the list of parameters.

For each homonym function, which has been selected as potential cloned function, the extended file name (i.e., including complete path) is reported together with the following three measures of code length:

- number of lines of code (LOC)
- number of effective lines of code (ELOC), excluding comments and blank lines
- number of comment lines of code (CLOC)

The second step of our approach uses as input the report of potential cloned functions to visually inspect code. Rather than using measures to automatically classify clones, such as in [14], we use size measures to provide clues and set priorities for the visual inspection.

Visual inspection is aimed to classify every pair of homonym selected functions in exactly one level of the classification in Table 1. Checking proceeds from level 1 to level 4 and stops when a level can be recognized.

Selected script functions are grouped according to the refactoring opportunity. That is, if a set of homonym functions shares the same level-*n* cloning relation, then the set is classified as a group at level-*n*. For example, if there are 4 homonym script functions and all 6 pairs can be classified as *Identical* (level 1), then we have one group of 4 script functions at level 1 for doing refactoring. However, if one of the script functions is *Similar* (rather than *Identical or Nearly-identical*) with respect to the other 3 homonyms, then there is a group of 4 script functions at level 3, because the opportunity of refactoring is unique. On the contrary, if one of the script functions is *Distinct* (rather than *Identical, Nearly-identical*, or *Similar*), then there is one group of just 3 script functions at level 1 and the different (albeit homonym) function is considered out of the refactoring scope.

3. Case studies

The goal of our three case studies is to assess whether our approach can be easily used by maintainers to identify cloned script functions, and what type of cloning is found with respect to our classification schema.

We used two web applications, respectively from the public and commercial domain, for which we did not have expectations about how much duplication exists, and one web application from the research domain for which it was known that there were many duplicated script functions.

All the three web applications use Microsoft's ASP technology to implement web server pages. Table 2 shows a characterization of the selected case studies, including size-related statistics. The three case studies range in size from tens to hundreds of files.

3.1. BugTrack

BugTrack is a basic web-based bug tracking system which is provided as a demo application of a code generation tool. BugTrack has been built been using CodeCharge Studio [19], which is a rapid application development (RAD) environment that generates dynamic web pages around a database. The code generation engine separates presentation logic (implemented by static HTML pages) from application logic (implemented by dynamic ASP pages).

The tool did not select any server-side script function as suspect clone, and reported as potential clones all the three client-side functions (see Table 3).

Visually inspecting the scripting code of the *delconf* functions, we noticed that they are event handlers which ask for confirmation when the user clicks a Delete button inside a form. The three copies only differ for the name of the document form that the functions check for its value. In order to invoke the same copy of the script function, it would be enough to use a generic form name both in the HTML files and inside the script function. The script function should be placed in a client-side include file (with a 'js' filename extension) and retrieved using the *src* attribute of the *script* tag. Then we classify the selected script functions as clones of a same group at level 3 (*Similar*).

Table 2. Characterization of case studies

Web application	BugTrack	Conf. Mgmt. Service	IBIS
Client-side scripting language	JavaScript	JavaScript	JavaScript
Server-side scripting language	VBScript	VBScript	JavaScript
No. of files (total)	37	46	230
No. of ASP files	15	9	110
No. of HTML files	13	15	0
No. of client-side include files	0	3	8
No. of server-side include files	1	6	0
No. of client-side script functions	3	79	138
No. of server-side script functions	21	52	282

3.2. Conference Management Service

Microsoft's Exchange 2000 Conferencing Server [15] is a platform to support data, audio, and video conferencing. It is built on top Exchange 2000 Server and integrates conferencing features of Microsoft's NetMeeting with calendaring features of Microsoft's Outlook.

We looked at one of the subsystems, called Conference Management Service, which keeps track of scheduled conferences and provides administrators with control of attendee access to conferences. The Conference Management Service is a web application which uses Microsoft's ASP technology for scripted pages and external binary components.

Our tool did not select any server-side script function but reported as potential clones (as shown in Table 4), 25 client-side script functions over a total of 79 (32%).

We visually inspected the code of the selected script functions. The first two suspect clones, named *refreshConfPanel*, are located in the same ASP file, the former in the *head* element and the latter in the *html* element. They differ because the former uses a constant to represent the frequency of screen-refresh while the latter uses the same value but in a numerical form. Then the second function might be removed, and the two functions can be classified as clones of a same group at level 2 (*Nearly-identical*).

The other two couples of suspect clones, *OnResolveClickContinue* and *OnResolveClickCancel*, both differ for a string assigned to the *action* field of the form. To invoke the same function exemplar, a parameter should be added to the script function and callers should pass a string as argument. Both script functions should be placed in a client-side include file (with a 'js' filename extension) and retrieved using the *src* attribute of the *script* tag. Then we classify the two couples of selected script functions as clones of two groups at level 3 (*Similar*).

The other script functions that were selected as potential clones are named *OnLoadBody*. Looking to source code, we noticed that those script functions with equal metric values differ from each other because of a numeric value passed as argument to the same called function. These copies might be replaced by a new script function with a parameter added its signature, and then they can be classified as a group of 15 script functions at level 3 (*Similar*). The remaining four script functions, also named *OnLoadBody*, differ from each other and from previous functions, with respect to the accomplished functionality. These functions are homonyms just because they are triggered by the same *onLoad* event in the *body* element. Then all their cloning relations are classified at level 4 (Distinct). Being false positives, they are excluded from any refactoring.

Table 3: Report of potential client-side function clones for BugTrack

Function	*Expanded filename*	*LOC*	*ELOC*	*CLOC*
delconf	bugtrack\BugRecord.html	4	4	0
delconf	bugtrack\EmployeeMaint.html	4	4	0
delconf	bugtrack\ProjectMaint.html	4	4	0

Table 4: Report of potential client-side function clones for Conference Management Service

Function	*Expanded filename*	*LOC*	*ELOC*	*CLOC*
refreshConfPanel	Exchange2000server\confpanel.asp	4	4	0
refreshConfPanel	Exchange2000server\confpanel.asp	4	4	0
OnResolve ClickContinue	Exchange2000server\schedule.asp	12	12	0
OnResolve ClickContinue	Exchange2000server\send.asp	12	12	0
OnResolve ClickCancel	Exchange2000server\schedule.asp	6	6	0
OnResolve ClickCancel	Exchange2000server\send.asp	6	6	0
OnLoadBody	help\ConfCancel.htm	10	10	0
OnLoadBody	help\ConfJoin.htm	10	10	0
OnLoadBody	help\ConfSchedule.htm	10	10	0
OnLoadBody	help\ConfTypes.htm	10	10	0
OnLoadBody	help\ConfTypes_Data.htm	10	10	0
OnLoadBody	help\ConfTypes_Video.htm	10	10	0
OnLoadBody	help\Experience.htm	10	10	0
OnLoadBody	help\Experience_Org.htm	10	10	0
OnLoadBody	help\Experience_Part.htm	10	10	0
OnLoadBody	help\Faq.htm	10	10	0
OnLoadBody	help\Help.htm	10	10	0
OnLoadBody	help\HowToSchedO2K.htm	10	10	0
OnLoadBody	help\HowToSchedOWA.htm	10	10	0
OnLoadBody	help\HowToSchedWS.htm	10	10	0
OnLoadBody	help\Welcome.htm	10	10	0
OnLoadBody	Exchange2000server\leftpane.asp	4	4	0
OnLoadBody	Exchange2000server\list.asp	63	54	1
OnLoadBody	Exchange2000server\schedule.asp	26	24	1
OnLoadBody	Exchange2000server\send.asp	6	6	0

3.3. IBIS

IBIS [13] is a web application that provides groupware support for distributed software inspections. All groupware functionalities were implemented by final-year undergraduate students at University of Bari using Microsoft's ASP technology for scripted pages and external components for basic services.

The eMetrics tool selected as potential clones 56 client-side script functions over 138 (40%), and 168 server-side script functions over 282 (59%). For the sake of space, we do not include the report in a tabular format, as we did for the previous case studies. Figures 1 and 2 summarize the selected potential clones, respectively for client-side and server-side script functions. Function names are shown on the x-axis, and their frequency on the y-axis.

We analyzed source code in order to classify the suspected clones and identify refactoring opportunities. Among potential clones, there were 2 client-side script functions and 29 server-side script functions, which were classified as *Distinct* (level 4). In other terms, false positives were 1% and 10%, respectively for client and server-side functions, of the total of automatically selected potential clones.

Results for refactoring opportunities are reported for client-side and server-side script functions, respectively in Table 5 and Table 6.

For each level, from level 1 to level 3, the following values are reported:

- the number of groups for which there is the same level of clone relationship;
- the number of functions involved in cloning;
- the above number expressed as percent of the total number of functions in the application;
- the amount of code involved in the groups, measured as effective lines of code;
- the above number expressed as percent of the total amount of scripting function code in the application.

The total raw in the two tables contains the cumulative frequencies and cumulative percentages of the three categories (*Identical*, *Nearly-identical*, *Similar*). The totals represent the number and the impact of refactoring opportunities that are worth taking advantage of. Each refactoring opportunity is a simple but useful change that does not alter the external behavior of the application.

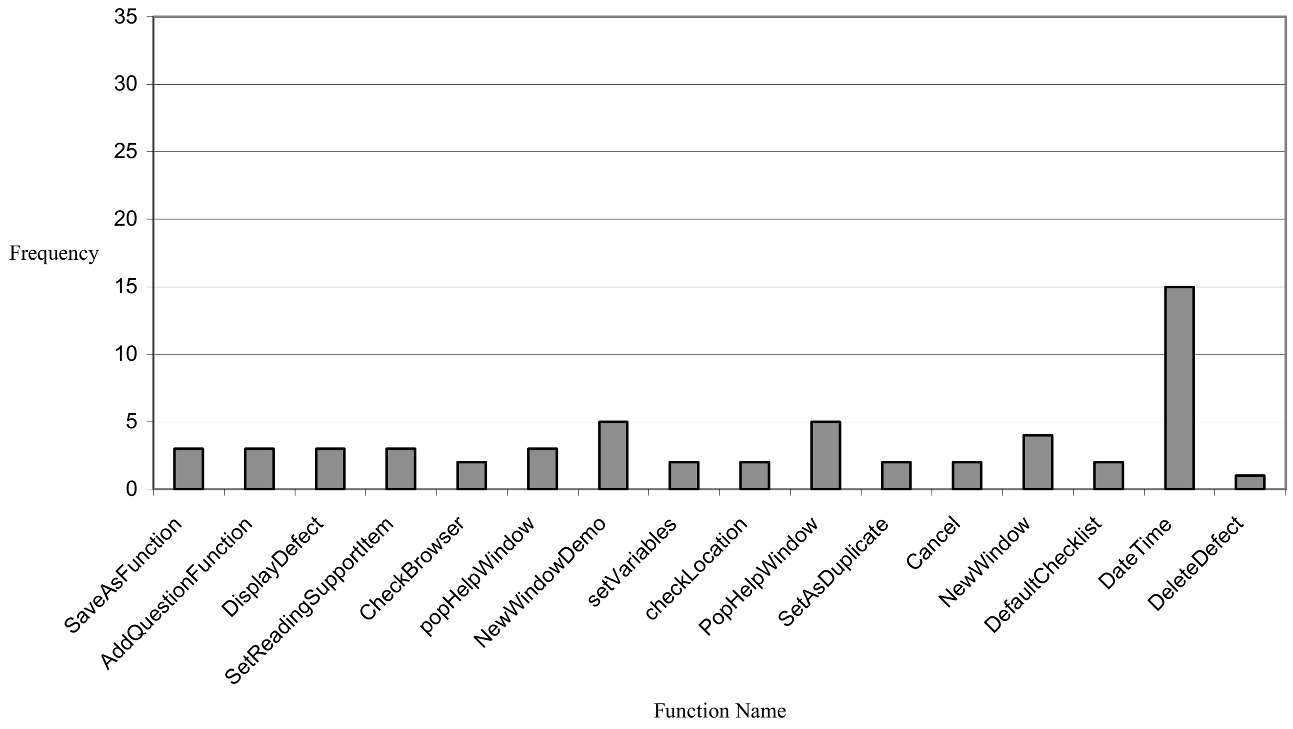

Figure 1. Summary of potential client-side function clones for IBIS

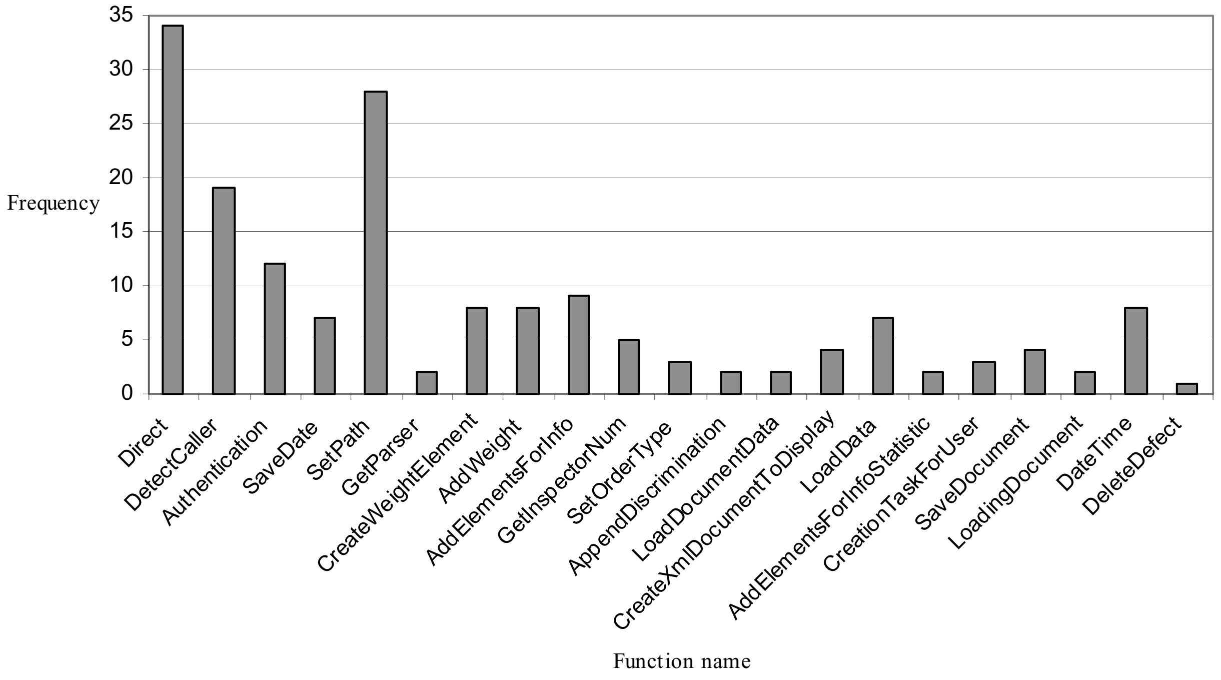

Figure 2. Summary of potential server-side function clones for IBIS

Table 5. Summary of refactoring opportunities for client-side function clones in IBIS

Level	Groups	Function clones	% Function clones	ELOC	% ELOC
1	8	31	22%	602	36%
2	0	0	0%	0	0%
3	5	23	17%	171	10%
Total	**13**	**54**	**39%**	**773**	**46%**

Table 6. Summary of refactoring opportunities for server-side function clones in IBIS

Level	Groups	Function clones	% Function clones	ELOC	% ELOC
1	11	58	21%	841	16%
2	2	42	15%	400	7%
3	2	40	14%	461	9%
Total	**15**	**140**	**50%**	**1702**	**32%**

4. Conclusions

This paper presented a semi-automated approach for identifying function clones embedded in HTML scripting of web applications. A list of potential cloned script functions is automatically produced by a tool, while verification and classification of suspect clones is performed through visual inspection of source code. The contribution of this approach is directed to support both verification tasks, before a new file is put under configuration management, and reengineering activities, after that the lack of method in development has driven a web application towards code decay.

An empirical validation of the approach has been performed on three case studies. From the larger case study, for which we knew that there were many cloned functions, we measured the number of false positives and refactoring opportunities. False positives accounted between 1% and 10% of the reported potential clones. On the other hand, groups of true clones, for which duplicated code can be easily removed, accounted between 39% and 50% of the total number of script functions in the web application, and between 32% and 46% of the total amount of code inside script functions. These measures show the usefulness of our approach with respect to precision (few false positives) and opportunity for design improvement (many duplicates to remove). However, that we found a high number of function clones does not mean that we were able to identify all duplicates. In fact, while we indirectly assessed precision (percentage of true clones with respect to reported clones), we did not assess recall (percentage of true clones with respect to existing clones) because we were not able to check all files for code duplication manually. Thus, the empirical validation might be improved by measuring both precision and recall, on the basis of the experimental approach followed in [11], where programmers of the subject systems where consulted in advance to get a sample of known duplicated code fragments.

Our approach might be refined, analogously to [14], by extending the type and number of metrics so that an exact (or nearly exact) metrics identity is required to classify a pair of functions. The automatic selection of potential clones could be extended to other web enabling technologies, such as Java Server Pages or PHP. Finally, our approach might be used as an empirical instrument to study the evolution of web applications across multiple releases.

Acknowledgments

We would like to thank Vincenzo Fiorentino and Biagio Taccogna for having helped to implement the eMetrics tool.

References

[1] G. Antoniol, G. Casazza, M. Di Penta and E. Merlo, “Modeling Clones Evolution Through Time Series”, *Proc. of the International Conference on Software Maintenance*, Florence, Italy, November 2001, pp. 273-280.

[2] B.S. Baker, “On Finding Duplication and Near-Duplication in Large Software Systems”, *Proc. of the Second Working Conference on Reverse Engineering*, Toronto, Ontario, Canada, July 1995, pp. 86-95.

[3] M. Balazinska, E. Merlo, M. Dagenais, B. Lague and K. Kontogiannis, “Partial Redesign of Java Software Systems Based on Clone Analysis”, *Proc. of the Sixth Working Conference on Reverse Engineering*, Atlanta, Georgia, October 1999, pp.326-336.

[4] M. Balazinska, E. Merlo, M. Dagenais, B. Lague and K. Kontogiannis, “Measuring Clone Based Reengineering Opportunities”, *Proc. of the Sixth IEEE International Symposium on Software Metrics*, Boca Raton, Florida, November 1999, pp. 292-303.

[5] I.D. Baxter, A. Yahin, L. Moura, M. Santa Anna and L. Bier, “Clone Detection Using Abstract Syntax Trees”, *Proc. of the International Conference on Software Maintenance*, Bethesda, Maryland, November 1998, p. 368-377.

[6] J. Conallen, *Building Web Applications with UML*, Addison-Wesley, Reading, MA, 2000.

[7] G.A. Di Lucca, M. Di Penta, A.R. Fasolino and P. Granato, “Clone Analysis in the Web Era: an Approach to Identify Cloned Web Pages”, *Proc. of the Seventh IEEE Workshop on Empirical Studies of Software Maintenance*, Florence, Italy, November 2001, pp.107-113.

[8] S. Ducasse, M. Rieger and S. Demeyer, “A Language Independent Approach for Detecting Duplicated Code”, *Proc. of the International Conference on Software Maintenance*, Oxford, England, UK, August 1999, pp.109-118.

[9] F. Fioravanti, G. Migliarase and P. Nesi, “Reengineering Analysis of Object-Oriented Systems via Duplication Analysis”, *Proc. of the International Conference on Software Engineering*, Toronto, Canada, May 2001, pp.577-590.

IEEE COMPUTER SOCIETY

[10] M. Fowler, *Refactoring: Improving the design of existing code*, Addison-Wesley, Reading, MA, 1999.

[11] K. Kontogiannis, “Evaluation Experiments on the Detection of Programming Patterns Using Software Metrics”, *Proc. of the 4th Working Conference on Reverse Engineering*, Amsterdam, The Netherlands, October 1997, pp. 44-54.

[12] B. Lague, D. Proulx, J. Mayrand, E.M. Merlo and J. Hudepohl, “Assessing the Benefits of Incorporating Function Clone Detection in a Development Process”, *Proc. of the International Conference on Software Maintenance*, Bari, Italy, October 1997, pp. 314-321.

[13] F. Lanubile, and T. Mallardo, “Tool Support for Distributed Inspection”, *Proc. of the 26th Annual International Computer Software & Applications Conference*, Oxford, England, August 2002, pp.1071-1076.

[14] J. Mayrand, C. Leblanc and E.M. Merlo, “Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics”, *Proc. of the International Conference on Software Maintenance*, Monterey, CA, November 1996, pp. 244-254.

[15] Microsoft, Exchange 2000 Conferencing Server, http://www.microsoft.com/exchange/techinfo/conferencing/

[16] D.L. Parnas, “Software Aging”, *Proc. of the 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994, pp. 279-287.

[17] J.F. Patenaude, E.M. Merlo, M. Dagenais, B. Lague, “Extending Software Quality Assessment Techniques to Java Systems”, *Proc. of the Seventh International Workshop on Program Comprehension*, Pittsburgh, Pennsylvania, May 1999, pp. 49-57.

[18] R. S. Pressman, T. Lewis, B. Adida, E. Ullman, T. DeMarco, T. Gilb, B. Gorda, W. Humphrey, R. Johnson, “Can Internet-Based Applications Be Engineered?”, *IEEE Software*, Vol. 15, No. 5, Sept./Oct. 1998, pp. 104-110.

[19] YesSoftware, Inc, CodeCharge Studio, http://codecharge.com/studio