Accepted for 5th Intl. Conference on Web Engineering (ICWE'05)

An Investigation of Cloning in Web Applications

Damith C. Rajapakse and Stan Jarzabek

Department of Computer Science
School of Computing
National University of Singapore
{damithch,stan}@comp.nus.edu.sg

Abstract. Cloning (ad hoc reuse by duplication of design or code) speeds up development, but also hinders future maintenance. Cloning also hints at reuse opportunities that, if exploited systematically, might have positive impact on development and maintenance productivity. Unstable requirements and tight schedules pose unique challenges for Web Application engineering that encourage cloning. We conducted a systematic study of cloning in 17 Web Applications of different sizes, developed using a range of Web technologies, and serving diverse purposes. We found cloning rates 17-63% in both newly developed and already maintained Web Applications. Contribution of this paper is two-fold: (1) our results confirm potential benefits of reuse-based methods in addressing the key challenges of Web engineering, and (2) a framework of metrics and presentation views that we defined and applied in our study may be useful in other similar studies.

1 Introduction

Today, web sites are changing from mere collections of static hypertext documents to full blown software applications, commonly called Web applications (WA). In contrast to static web sites, WAs are bigger, more complex, more business critical, and more close to traditional software applications, requiring bigger initial investments and longer payback periods. WAs also have dramatically short development life-cycles, and fuzzy initial requirements resulting in frequent latent changes. All these add to the challenge of engineering and maintaining WAs.

Cloning has been recognized as a pervasive problem in maintenance of traditional software applications. It has been the focus of research for at least a decade. Cloning increases the tendency for update anomalies (inconsistencies in updating). Cloning also increases the effort required in program comprehension. Both these negatively affect maintenance. Reasons for cloning are manifold; most of them are related to programmer's intent to reuse the implementation of some abstraction [1]. It is a commonplace practice and cloning levels as high as 68% [8] have been reported in traditional software. With the recent proliferation of WAs, cloning in web domain is becoming an issue worthy of attention. As one benefit of cloning is the reduction of initial development time, shorter time-to-market requirement of WAs makes them ideal breeding grounds for clones. Also, the lack of suitable reuse and delegation mechanisms in HTML makes WAs a good candidates for clone proliferation [5]. On the positive side, the same similarity patterns that make cloning possible also signify valuable reuse opportunities. By exploiting such reuse opportunities systematically, we may cut development effort and ease future maintenance of WAs. Technologies for realizing this potential exist (server side scripting, template engines, meta-level techniques), but it is not known how well they fare in current state of the practice. As per our knowledge, no systematic study of cloning in the web domain has been done so far. In research on cloning in Web domain [3],[4],[5],[6],[11],[12], we did not find a published work giving concrete evidence of the extent of cloning in Web domain. Particularly, it is not known how the cloning problem in Web domain compares to that in the traditional software domain.

The above observations encouraged us to conduct a study of cloning in the WA domain, as described in this paper. The contribution of the paper is two-fold. (1) We conducted a comprehensive study of cloning in many types of WAs. We used WAs of different sizes, developed using a range of technologies, to cater for different application domains, by teams of different structures, using different development/business models, in different development environments. From our study, we were able to confirm potential benefits of reuse-based methods in addressing the key challenges of Web engineering. (2) We defined similarity metrics and clone analysis presentation views to be used in evaluating the extent of cloning in WAs. We adopted a general-purpose clone detector CCFinder [9] for analysis of the many types of sources that form WAs. We used, and validated, our clone evaluation framework in the study and we believe it will provide useful guidelines for future similar studies done by others, not only in Web domain, but in other domains as well.

Our study indicates that the extent of cloning in WAs is indeed substantial, exceeding cloning rates that we find in traditional applications. This shows the importance of investigating engineering techniques capable of defining generic solutions to avoid counter-productive cloning. Current technologies make a step in the right direction, but our analysis shows that there is room for improvement.

The remainder of this paper is organized as follows. In Section 2 we describe the experiment method, giving details of tools, metrics and graphs used. The results of the study are presented in Section 3, followed by related work is given in Section 4. Conclusions end the paper.

2 Experiment method

In this experiment, we analyzed 17 WAs covering the following.

- **Languages/technologies** - Java, JSP, ASP, ASP.net, C#, PHP, Python, Perl, Web services, proprietary template mechanisms
- **Application domains -** collaboration portals, e-commerce applications, web based DB administration tools, conference management, corporate intranets, bulletin boards, etc.
- **System sizes** - 33 ~1719 files
- **License types** – free, commercial, internal use,

- **Development models** – open source, closed source
- **Life cycle stage** – pre/first/post release, dead
- **Usage types** – off-the-shelf, one-time-use, custom-built, model applications
- **Team structures** – single author, centralized teams, distributed teams
- **Organizations** – software development companies including Microsoft, Sun Microsystems, and Apache Software Foundation, free lance software developers, in-house development teams of non-software companies

In our choice of WAs, we have tried to represent the diversity of WA domain in an unbiased manner. Due to practical limitations, the number of WAs we could include in the study was limited to 17. Although it was possible to increase the sample size by including many readily available open source WAs, we refrained from doing that, in order to keep a balance between open source WAs and (less readily available) closed source WAs. The scope of analysis was clones in *any* text file that is likely to be maintained by hand, including files not normally considered 'source code'. More than 11000 files were analyzed in total.

We used CCFinder [9] as our clone detector. CCFinder can detect exact clones and parameterized clones. Our experiment needed to detect clones in files written in many languages, not necessarily languages supported by CCFinder. Therefore, we instructed CCFinder to assume all input files as 'plain text'. In this mode, only exact clones were detected. We also instructed CCFinder to ignore trivially short clones (i.e. clones shorter than 20 tokens) and clones occurring within the same file, in order to keep the volume of reported clones within manageable limits. We developed a Java program called 'Clone Analyzer' to control the clone detection process and to analyze the clones detected by CCFinder. Fig. 1 shows the steps of clone analysis process. Next, we describe the metrics and visualizations used in the experiment.

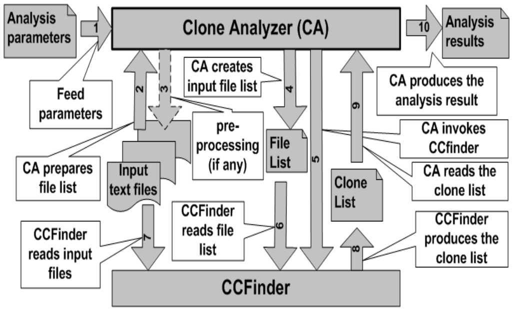


Fig. 1. Clone analysis workflow

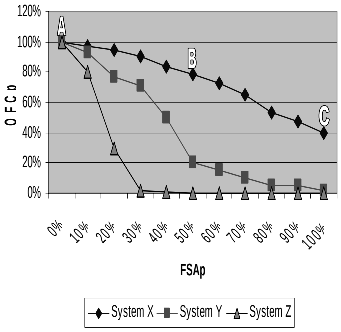


Fig. 2. Sample FSCurves

Total Cloned Tokens (TCT)**:** We defined TCT of a system as the sum of clone related tokens, i.e., tokens that form a part of any of the clones in that system. *TCTp* is *TCT* expressed as a percentage of total number of tokens in the system. When *TCTp* is high, *update anomaly risk* (the risk of inadvertently creating an update anomaly while modifying the system) is also high. If the *TCTp* is greater than 50%, system has more clones than non-clones; every update to the system has a higher chance of involving a clone than not; and hence runs a high risk of creating an update anomaly. We can call such systems *high update anomaly risks*.

File Similarity (FSA): While *TCTp* is a good indication of the overall cloning level of a system, it can be further complemented by a measure of file similarity. For example, consider two systems X and Y of similar size, both having the same *TCTp*. In X, clones are scattered across the system in such a way that no two files are substantially similar. But In Y, clones are well concentrated into a certain set of files. From a clone treatment perspective, system Y is more interesting than X because the clones in Y are more easily treatable than that of X. To identify the similarity of a file *f* to other files, we calculated the metric *FSA(f)*. We defined *FSA(f)* as follows (This is analogous to *RSA(f)* defined in [13]).

$$FSA(f) = \frac{1}{Tn(f)} \sum_{c \in CF(f)} Tn(c)$$

Here, *Tn(f)* is the number of tokens in file *f*, *CF(f)* is a set of code fragments which are included in file *f* and have a clone relation with some code fragments in other files, and *c* is an element of *CF(f)*. In this summation, overlapped code portions are counted only once. *FSA(f)* is a direct measure of the similarity (resulting from cloning) of file *f* to other files in the system. For example, *FSA=0.6* for a given file *f* means 60% of *f* has been cloned from other files of the system. For convenience, we defined the metric *FSAp* as FSA given as a percentage (i.e., *FSAp(f) = FSA(f) * 100%*)

Qualifying File Count (QFC): We define Qualifying File Count for *FSAp* value *v*, *QFC(v)*, as the number of files for which *FSAp* is not less than *v*. For example, *QFC(30%)* gives the number of files in the system having a *FSAp* value not less than 30%. *QFCp* is *QFC* expressed as a percentage of the total number of files in the system. For example, *QFCp(60%)* = 43% means, in 43% of files in the system, 60% or more have been cloned.

File Similarity Curve (FSCurve): To observe the overall file similarity characteristics across an entire system, we used File Similarity Curve (FSCurve). An FSCurve is created by plotting *QFCp* against *FSAp*. In the example FSCurve shown in Fig. 2, we have marked points A, B and C to illustrate how to interpret FSCurves. Point A indicates the invariant property that in 100% of files at least 0% has been cloned. At the other extreme, point C indicates that 40% of the files in System X have been completely (100%) cloned. Similarly, point B denotes that for System X, *QFC(50%) ≈ 80%*. i.e. in about 80% of the files in System X, at least 50% of the contents have come from other files. From FSCurves we can also get an idea about relative file similarity characteristics of different systems. For example, from the three FSCurves in Fig. 2, we can clearly see that file similarity in system Y is generally less than that of X but more than that of Z. i.e. Higher the position of the curve, higher the file similarity.

3 Experiment Results

3.1 Overall Cloning Level

The initial phase of our investigation was focused on the overall cloning level in WAs. Given in Fig. 3 is the *TCTp* of each WA we studied. Only one WA has a *TCTp* below 20%. The average *TCTp* is 41% (with a standard deviation of 15%). Five WAs are high update anomaly risks (*TCTp*>50%) while three more are close behind.

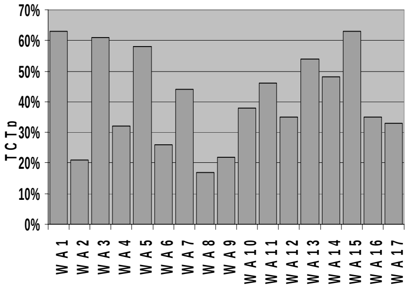
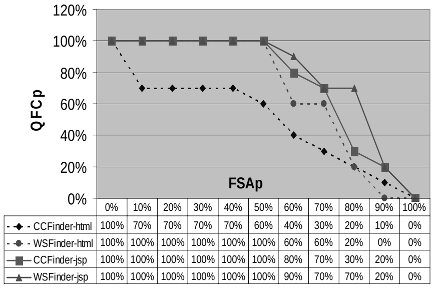


Fig. 3. Cloning level in each WA



	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
CCFinder-html	100%	70%	70%	70%	70%	60%	40%	30%	20%	10%	0%
WSFinder-html	100%	100%	100%	100%	100%	100%	60%	60%	20%	0%	0%
CCFinder-jsp	100%	100%	100%	100%	100%	100%	80%	70%	30%	20%	0%
WSFinder-jsp	100%	100%	100%	100%	100%	100%	90%	70%	70%	20%	0%

Fig. 4. CCFinder Vs WSFinder

From these data alone, the level of cloning in WAs seems substantial. Still, these data do not include clones with parametric variations (parameterized clones) and non-parametric variations (gapped clones). As a result, the actual cloning level in WAs could be even higher than the levels indicated by these data. We tested this hypothesis by comparing cloning level reported by CCFinder and a web-specific clone detector described in [3]. This clone detector (for convenience, we refer to it as *WSFinder*) detects the similarity among web-specific files. We did not use it as our main clone detector because it currently supports HTML and JSP files only. WSFinder reports three different values of file similarity based on 1. HTML tags, 2. Text included inside HTML tags, and 3. Scripts included in the file (only applicable to JSP pages). For a small set of HTML and JSP pages, we applied both CCFinder and WSFinder to compare results. To make the comparison least biased towards the hypothesis, we compared the minimum of the three values reported by WSFinder against CCFinder results. As shown in Fig. 4, CCFinder almost always reported a cloning level less than or equal to that reported by WSFinder. This supports our hypothesis that actual cloning level in WAs could be even higher than what is reported in this paper.

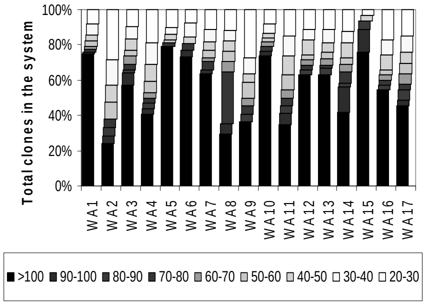


Fig. 5. Distribution of clone size

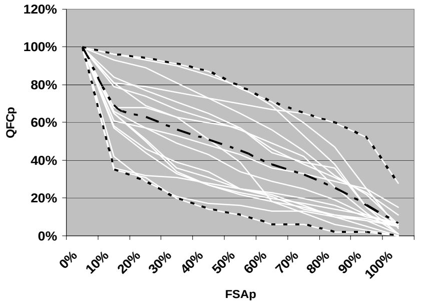


Fig. 6. FSCurves for all WAs

A high level of cloning does not necessarily mean a high reuse potential. The clones detected could be too small, too dispersed, or false positives. Since our minimum clone length was 20 tokens, these results could include clones as short as 20 tokens. (We did not use a higher minimum clone length, in the hope of capturing some of the parameterized clones or gapped clones; a parameterized/gapped clone contains a number of smaller exact clones). This could prompt one to argue that clones detected are trivially short ones, not worthy of elimination. To address this concern, we used the breakdown of the clones by length, in each system (as shown in Fig. 5). Clone size increases from 20 to 100+ as we go from top to bottom of each bar. Increasingly larger clones are shown in increasingly darker colors. As an average LOC is accounted by 6-8 tokens, a 100 token clone is roughly 15 LOC long. Therefore, this graph shows that most clones we detected are longer than 15 LOC.

To address the issue of clones dispersed across the system too thinly, we generated FSCurves for each system. To save space, we show all the FSCurves together in Fig. 6, with the average, the minimum, and the maximum curves marked with dashed lines. According to the average curve, close to 50% of the files have at least 50% of their content cloned. Fig. 7 represents two cross sections of Fig. 6, namely, at *FSAp*=50% and *FSAp*=90%. We use this graph to give a bit more detailed view of the clone concentration in each WA. It shows the percentage of files in each system that we can consider 'cloned' (*FSAp*≥50%) and 'highly cloned' (*FSAp*≥90%). In eleven of the WAs, we find more than 10% of the files have been highly cloned. In five, we find more than 20% of the files have been highly cloned. Aggregating all the WAs, the percentages of cloned and highly cloned files are 48% and 17% respectively. These data suggest that there is good clone concentration in files.

With regards to the issue of false positives, it is not practical to manually weed out the false positives in a study of this scale. However, since we detected only exact clones, we believe the false positives are at a minimum.

3.2 Cloning Level in WAs Vs Cloning Level in Traditional Applications

Since cloning in Traditional Applications (TAs) has been widely accepted as a problem, we wanted to compare cloning levels of WAs to that of TAs. We started by separating the files in WAs into two categories:

- WA-specific files – files that use WA-specific technologies, e.g., style sheets, HTML files, ASP/JSP/PHP files
- General files – files equally likely to occur in WAs and TAs. e.g., program files written in Java/C/C#, build scripts

We found 13 of the WAs had both type of files, while some smaller WAs had only Web-specific files. For WAs with both type of files, we calculated *TCTp_W* (*TCTp* for WA-specific files) and *TCTp_G* (*TCTp* for general files) as given in Fig. 8. The last two columns show that overall *TCTp_W* was 43% and overall *TCTp_G* was 35%. The *TCTp* comparison of individual WAs shows that in 6 WAs *TCTp_W* is significantly higher (*TCTp_W* > *TCTp_G* by more than 10%), in 3 WAs levels are similar (|*TCTp_W-TCTp_G*|≤10%), and only in 4 WAs *TCTp_G* was significantly higher (*TCTp_G* > *TCTp_W* by more than 10%). These figures suggest that WA-specific files have more cloning than general files. But we can reasonably assume that cloning in full fledged TAs is not worse than cloning in these general files. This infers that cloning in WAs is worse than cloning in TAs.

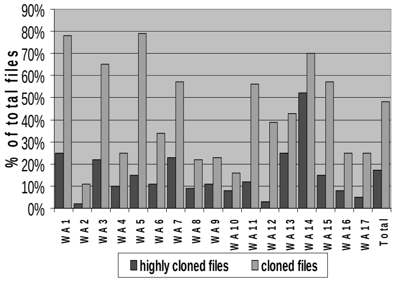


Fig. 7. Percentage of cloned files

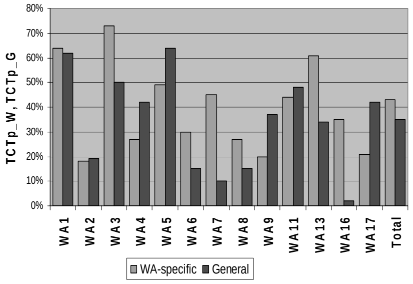


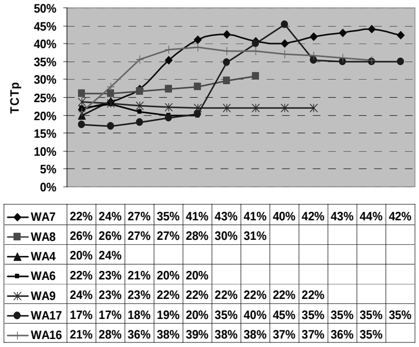
Fig. 8. WA-specific files Vs General files

3.3 Factors Affecting the Cloning Level

Our investigation also included collecting quantitative data on different factors that might affect cloning in WAs. We started by investigating whether system size has any effect on the cloning level. However, a comparison of average cloning level in small, medium, and large WAs (Table 1) showed that cloning level does not significantly depend on the system size.

Table 1. Average cloning for WAs of different size

Size (in # of files)	Avg *TCTp*	Std. Deviation
Small (size < 100)	40%	21%
Medium (100 ≤ size < 1000)	42%	14%
Large (size ≥ 1000)	40%	16%
All	41%	15%



◆ WA7	22%	24%	27%	35%	41%	43%	41%	40%	42%	43%	44%	42%
■ WA8	26%	26%	27%	27%	28%	30%	31%					
▲ WA4	20%	24%										
▪ WA6	22%	23%	21%	20%	20%							
✳ WA9	24%	23%	23%	22%	22%	22%	22%	22%	22%			
● WA17	17%	17%	18%	19%	20%	35%	40%	45%	35%	35%	35%	35%
+ WA16	21%	28%	36%	38%	39%	38%	38%	37%	37%	36%	35%	

Fig. 9. Movement of cloning level over time

Continuing, we also investigated the progression of cloning level over time. For this, we used seven of the WAs for which at least four past releases were readily available. All seven suitable WAs were open source, and of medium or large size. In the Fig. 9, we show the moving average (calculated by averaging three neighboring values) of *TCTp* over past versions, up to the current version. According to this graph, all WAs show an initial upward trend in the cloning level. Some WAs have managed to bring down the *TCTp* during the latter stages, even though current levels still remain higher than the initial levels. This indicates that the cloning level is likely to get worse over time. WA9, and to a smaller extent WA6, are the only exceptions, but this may be due to non-availability of the versions corresponding to the initial stage.

3.4 Identifying the Source of Clones

Finally, we attempted to obtain some quantitative data that could be useful for devising a solution to the cloning problem. We were interested to find which of the following file categories contributed most clones

i. **Static files (STA)** – files that needs to be delivered 'as is' to the browser. Includes markup files, style sheets and client side scripts (e.g., HTML, CSS, XSL, JavaScripts).

ii. **Server pages (SPG)** – files containing embedded server side scripting. These generate dynamic content at runtime (e.g., JSP, PHP, ASP, ASP.NET).

iii. **Templates (TPL)** – files related to additional templating mechanisms used.

iv. **Program files (PRG)** – files containing code written in a full fledged programming language (e.g., Java, Perl, C#, Python)

v. **Administrative files (ADM)** – build scripts, database scripts, configuration files

vi. **Other files (OTH)** – files that do not belong to other five types.

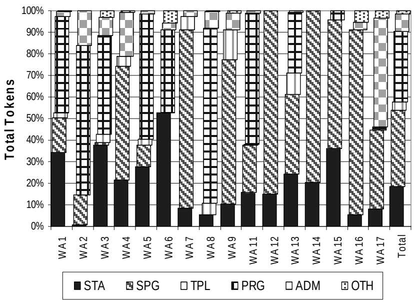


Fig. 10. Contribution of different file types to system size

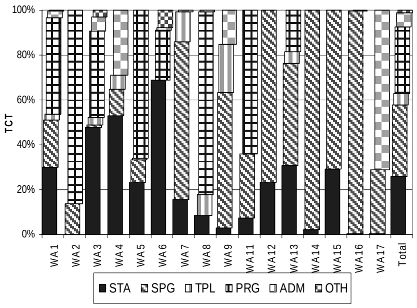


Fig. 11. Contribution of different file types to cloning

Fig. 10 gives the contribution of each file type towards system size while Fig. 11 gives the contribution of each file type towards cloning. The rightmost column of each graph shows the overall situation (aggregation all the WAs). The salient feature of these graphs is that there is no single file type that clearly dominates the composition of the system, or the composition of the clones. At least three types (STA, SPG, and PRG) shows dominant influence, while the influences of TPL and ADM are smaller, but not negligible. This shows that a successful solution to the cloning problem has to be applicable equally to the entire range of file types. Moreover, the high influence of WA-specific types (STA, SPG and to a lesser extent, TPL) suggests that a solution rooted in TAs might not be successful in solving the cloning problem in WAs.

4 Related Work

Clones have been defined in slightly different ways by different researches. These differences are usually related to different detection strategies used and the different domains focused on by each researcher. Accordingly, there are clone definitions specific to web domain; [5] defines HTML clones as pages that include the same set of tags while [4] defines clones as pages that have the same, or a very similar structure. In [3], web page clones are classified into three types based on similarity in page structure, content, and scripts. Since our study considers all text files for clone detection, not just HTML pages or script files, we use a simpler definition of clones, i.e., clones are similar text fragments.

Existing research in clone detection (CD) is based on two major approaches [7]: (1) Using structural information about the code. E.g., metrics, AST, control/data flow, slices, structure of the code/expressions, PDG, etc. (2) Using string-based matches. Our main detection tool CCFinder [9] falls into the 2nd category. It was developed at Osaka University, Japan. Since then, it has been used [10] and evaluated [2] by independent researchers. The string based approach of CCFinder is well suited for studies like ours involving text written in many languages including natural languages. Clone detection (CD) have many applications, like detecting plagiarism, program comprehension, reengineering, quality evaluation etc. and is an area extensively studied in the context of traditional software domain. Syntax errors and routine use of multiple programming languages make clone detection in the web domain harder [12]. There are CD tools/techniques specific to web being proposed [4],[5],[6],[12]. Technique in [5] detects clones in static pages only, but in [4] it was extended to include ASP pages. A (semi) automatic process aimed at identifying static web pages that can be transformed into dynamic ones, using clustering to recognize a common structure of web pages, is described in [11]. Technique in [3] identifies HTML and JSP pages that are more similar to each other than a specified threshold.

Gemini [13] is a maintenance support environment that augments CCFinder. Gemini provides the user with functions to analyze the code clones detected by CCFinder. It primarily provides two visualizations: scatter plot and metrics graph. The scatter plot graphically shows the locations of code clones among source codes. The metrics graph shows different metric values for clones. We regularly use Gemini in our research and have found it to be a very useful tool. However, there were three main limitations of Gemini that led us to write our own Clone Analyzer. First, when using Gemini, it is difficult to grasp the total cloning activity in the system [10]. Size of the clone metric graph in particular grows out of control when a large number of files are involved. Second, Gemini does not provide an API to access the analysis data. When using Gemini, it is not possible to analyze the clones beyond the visual data provided by the tool itself. And third, due to its visual nature, Gemini is more resource intensive compared to CCFinder; the number of clones it could handle is limited. Our Clone Analyzer overcomes these issues and was very useful in cutting down the time required to analyze the large amount of clone data generated in this study.

5 Conclusions and Future Work

We conducted a study of cloning in 17 Web Applications of different sizes, developed using a range of Web technologies, and serving diverse purposes. We found cloning rates 17-63% in both newly developed and already maintained Web Applications. To emphasize the reuse potential signified by these clones, we showed that most of the clones are substantially long, well concentrated and unlikely to be false positives. With the aid of a Web-specific clone detector, we substantiated our hypothesis that actual cloning level could be even higher than the levels reported here. We also showed that cloning equally affect small, medium or large WAs, and cloning gets worse over time. More importantly, we showed that cloning in WAs could be even worse than that of traditional applications. Firstly, our findings provide the concrete evidence of cloning in WAs we set out to produce at the start of this study. In doing so, it confirms the potential benefits of reuse-based methods in addressing the key challenges of Web engineering. Secondly, our study defines and validates a framework of tools, metrics, and presentation views that may be useful in other similar studies.

One explanation of substantial cloning we found is extensive similarity within WA modules, across modules and across WAs. The existence of those similarities, exceeding the rates of similarity we are likely to find in traditional software, underlines the importance of investigating techniques to improve the effectiveness of reuse mechanisms in Web Engineering. The study itself revealed an important consideration when devising such mechanisms. That is, it shows that a number of files categories – some of which use Web-specific technologies – contribute towards cloning in WAs. This suggests that any successful solution need to be uniformly applicable to all text sources, not just code written in a particular language.

In our future work, we hope to complement this quantitative analysis with more qualitative analysis. We hope such work will result in further insights into the nature of problem of cloning in WAs. We also plan to address design-level similarities, so-called structural clones. Structural clones usually represent larger parts of programs than the 'simple' clones detected by current clone detectors like CCFinder; therefore their treatment could be even more beneficial.

All the WAs included in this study are one-off products. Going further, a promising area we hope to work on is cloning in WA product lines. In the absence of effective reuse mechanisms, whole WAs could be cloned to create members of the product line, resulting in much worse levels of cloning than reported here. Server side scripting, current technology of choice for adding run time variability to WAs, may fall well short of the construction time variability a product line situation demands. We plan to explore this area to find synergies between run time and construction time technologies in solving the cloning problem of WA domain in general, and WA product lines in particular.

6 Acknowledgements

Authors thank following persons for their contributions during the study: Andrea De Lucia and Giuseppe Scanniello (Università di Salerno, Italy), Katsuro Inoue, Shinji Kusumoto, and Higo Yoshiki (Osaka Uni. Japan), Toshihiro Kamiya (PRESTO, Japan), Sidath Dissanayake (SriLogic Pvt Ltd, Sri Lanka), Ulf Pettersson (SES Systems Pte Ltd., Singapore), Yeo Ann Kian, Lai Zit Seng, and Chan Chee Heng (National University of Singapore).

References

1. Baxter, I., Yahin, A., Moura, L., and Anna, M. S., "Clone detection using abstract syntax trees," *Proc. Intl. Conference on Software Maintenance* (ICSM '98), pp. 368-377.
2. Burd, E., and Bailey, J., "Evaluating Clone Detection Tools for Use during Preventative Maintenance," *Second IEEE Intl. Workshop on Source Code Analysis and Manipulation (SCAM'02)* pp. 36-43.

3. De Lucia, A., Scanniello, G., and Tortora, G., "Identifying Clones in Dynamic Web Sites Using Similarity Thresholds," Proc. Intl. Conf. on Enterprise Information Systems (ICEIS'04), pp.391-396.
4. Di Lucca, G.A., Di Penta, M., Fasolino, A.R., "An approach to identify duplicated web pages," *Proc. 26th Annual Intl. Computer Software and Applications Conference* (COMPSAC 2002), pp. 481 – 486.
5. Di Lucca, G. A., Di Penta, M., Fasilio, A. R., and Granato, P., "Clone analysis in the web era: An approach to identify cloned web pages," *Seventh IEEE Workshop on Empirical Studies of Software Maintenance* (WESS 2001), pp. 107–113.
6. Lanubile, F., Mallardo, T., "Finding function clones in Web applications," Proc. Seventh European Conference on Software Maintenance and Reengineering, (CSMR' 2003). pp.379 – 386.
7. Marcus, A., and Maletic, J. I., "Identification of High-Level Concept Clones in Source Code," Proc. Automated Software Engineering, 2001, pp. 107-114.
8. Jarzabek, S. and Shubiao, L., "Eliminating Redundancies with a "Composition with Adaptation" Meta-programming Technique," Proc. European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE'03), pp. 237-246.
9. Kamiya, T., Kusumoto, S, and Inoue, K., "CCFinder: A multi-linguistic token-based code clone detection system for large scale source code," IEEE Trans. Software Engineering, vol. 28 no. 7, July 2002, pp. 654 – 670.
10. Kapser, C., and Godfrey, M. W., "Toward a taxonomy of clones in source code: A case study," *In Evolution of Large Scale Industrial Software Architectures*, 2003.
11. Ricca, F., Tonella, P., "Using clustering to support the migration from static to dynamic web pages," *Proc. 11th IEEE International Workshop on Program Comprehension,* (IWPC' 2003), pp. 207 – 216.
12. Synytskyy, N. Cordy, J. R., Dean, T., "Resolution of static clones in dynamic Web pages," *Proc. Fifth IEEE Intl. Workshop on Web Site Evolution,* (IWSE'2003), pp. 49 – 56.
13. Ueda, Y., Kamiya, T., Kusumoto, S., and Inoue, K., "Gemini: Maintenance Support Environment Based on Code Clone Analysis," *Proc. Eighth IEEE Symposium on Software Metrics*, pp. 67-76, 2002.