An Investigation of Cloning in Web Applications

Damith C. Rajapakse
Department of Computer Science
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
+65 6874 2834
damithch@comp.nus.edu.sg

Stan Jarzabek
Department of Computer Science
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
+65 6874 2863
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 are conducting a systematic study of cloning in Web Applications of different sizes, developed using a range of Web technologies, and serving diverse purposes. Our initial results show cloning rates up to 63% in both newly developed and already maintained Web Applications. Expected contribution of this work is two-fold: (1) to confirm potential benefits of reuse-based methods in addressing clone related problems of Web engineering, and (2) to create a framework of metrics and presentation views to be used in other similar studies.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Restructuring, reverse engineering, and reengineering;* D.2.8 [**Software Engineering**]: Metrics – *Product metrics;* D.2.13 [**Software Engineering**]: Reusable Software

General Terms: Measurement, Experimentation.

Keywords: Web Engineering, Web Applications, Clones, Clone metrics, Clone analysis, Software reuse, Software maintenance.

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 leading to 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. Cloning increases the tendency for update anomalies (inconsistencies in updating). Cloning also increases the effort required in program comprehension. Both these negatively affect maintenance. Cloning is a commonplace practice and cloning levels as high as 68% [1] have been reported in traditional software. With the recent proliferation of WAs, cloning in web domain is becoming an issue worthy of attention. 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.

The above observations encouraged us to conduct a study of cloning in the WA domain. The expected contribution of this work is two-fold. (1) A comprehensive study of cloning in many types of WAs. (2) To define similarity metrics and clone analysis presentation views, to be used in assessment of cloning in WAs. Initial results of our study indicate substantial levels of cloning in WAs, confirm potential benefits of reuse-based methods in addressing counter-productive cloning in Web Engineering. Current technologies make a step in the right direction, but our initial results suggest that there is room for improvement.

2. EXPERIMENT METHOD
In this experiment, we analyzed 17 WAs covering diverse **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), l**ife 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) and **organizations** (Software development companies including Microsoft, Sun Microsystems, and Apache Software Foundation, free lance software developers, in-house development teams of non-software companies). 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'. The study included over 11000 files.

We used CCFinder [2] 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,

WWW 2005, May 10-14, 2005, Chiba, Japan.
ACM 1-59593-051-5/05/0005.

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. The Figure 1 shows the steps of clone analysis process.

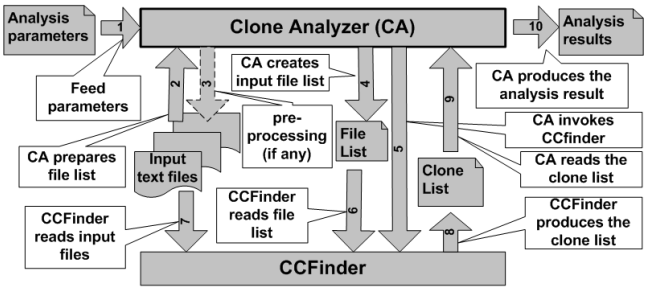

Figure 1. Clone analysis workflow

We used *Total Cloned Tokens (TCT)* – the sum of tokens that form a part of any of the clones in that system – as our primary clone metric. *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. Currently, we are in the process of defining more metrics and visualizations in the areas of file similarity, clone concentration and progression of cloning over time.

3. PRELIMINARY RESULTS

Given in Figure 2 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 have TCTp higher than 50% while three more are close behind.

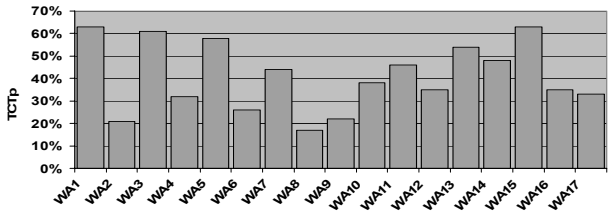

Figure 2. Cloning level in each WA

The length of the clones is an important factor when determining clones worth eliminating. The Figure 3 shows the breakdown of the clones by length, in each system. Minimum 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. This graph shows that most WAs have significant amount of clones that are longer than 100 tokens. These values clearly suggest that the cloning level in WAs is indeed substantial enough to warrant further investigation.

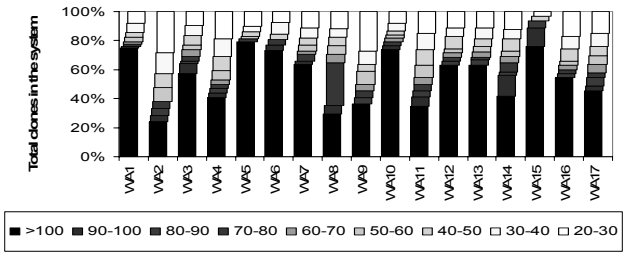

Figure 3. Distribution of clone size

4. FUTURE WORK

In our future work, we hope to produce more useful data on the cloning in WAs, by continuing with this quantitative analysis using more metrics and visualizations. Such metrics and visualization, together with the tools we use, could be the basis of a framework for other similar investigations. Then, we shall complement the quantitative analysis with more qualitative analysis seeking further insights into the nature of problem of cloning in WAs. This work will address so-called structural clones – patterns of repetitions emerging from analysis and design levels. 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.

We also hope to address issues related to validity and applicability of the results of this study, particularly, in the following three areas: (1) selecting a sample of WAs representative of the WA domain, (2) the accuracy of clone detection, and (3) the relationship of analysis results to potential reuse benefits.

This study currently includes only one-off WAs. 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.

5. REFERENCES

[1] 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.

[2] 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.