A Framework for Method Level Clone Breakthrough On Web Pages Using Metrics

K.Nirmalrajan[1], V.UdhayaKumar [2]
[1] M.Tech I Yr., Dept. of IT, [2] Asst., Prof., Dept. of CSE, Prist University
Pondicherry – 605 007, India.
nirmalrajank@gmail.com[1], Udhaya_kurinji@yahoo.com [2]

Abstract— **As the use of the web pages are increasing nowadays to a huge amount, some of the data that are present in the web pages can be characterized so that the work of the programmer is reduced. Many web applications use a mixture of HTML and scripting language code as the front-end to business services, where scripts can run on both the client and server side. Thus code duplication occurs frequently during the development and evolution of web applications. This ad-hoc but pathological form of reuse consists in copying, and eventually modifying, a block of existing code that implements a piece of required functionality. Duplicated blocks are named clones and the act of copying, including slight modifications, is called cloning. When entire functions are copied rather than fragments, duplicated functions are called function clones. In this paper we propose a model that identifies all the function clones that are present in the web pages.**

I. 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 named clones and the act of copying, including slight modifications, is called cloning. When entire functions are copied rather than fragments, duplicated functions are called function clones.A code fragment that has identical or similar code fragment(s) to it in the source code,in general, terms as code clone. A copied fragment can be used with or without minor modifications in a system by the developer. If there is no modifications or the modifications are within a certain level in the copied fragment then the original and copied fragments are called code clones and they form a clone pair as shown in fig1.1

Metrics-based techniques are related to hashing algorithms For each fragment of a program the values of a number of metrics is calculated, which are subsequently used to find similar fragments.

- Clone Pair:

A pair of code portions/fragments is called a clone pair if there exists a clone-relation between them, i.e., a clone pair is a pair of code portions/fragments which are identical or similar to each other.

- Clone Class:

A clone class is the maximal set of code portions/fragments in which any two of the code portions/fragments hold a clone-relation, i.e., form a clone pair.

- Clone Class Family:

The group of all clone classes that have the same domain is called a clone class family . Such a clone class family is also termed super clone.

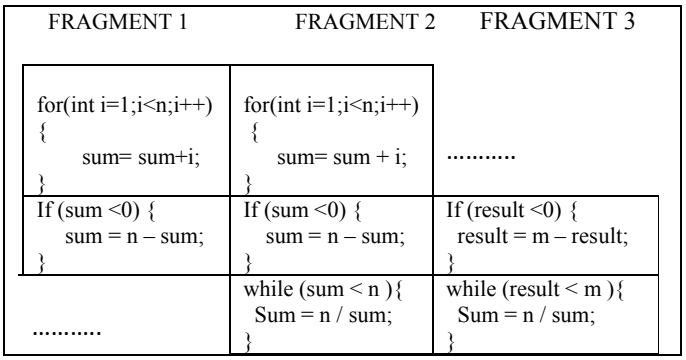

Fig: 1.1 Clone Pair and Clone Class

A. Different Types Of Clones

The cloning is performed based on to main constraints which is the textual similarity and the functional similarity.

- *Type I Clones:*

In *Type I* clones, a copied code fragment is the same as the original. However, there might be some variations in whitespace (blanks, new line(s), tabs etc.), comments and/or layouts. *Type I* is widely known as *Exact clones*.

- *Type II Clones:*

A *Type II* clone is a code fragment that is the same as the original except for some possible variations about the corresponding names of user-defined identifiers (name of variables, constants, class, methods and so on), types, layout and comments. The reserved words and the sentence structures are essentially the same as the original one. We see that the two code segments change a lot in their shape, variable names and value assignments. However, the syntactic structure is still similar in both segments.

- *Type III Clones:*

In *Type III* clones, the copied fragment is further modified with statement(s) changed, added and/or deleted. This copied fragments with one statement inserted is called *Type III* code clone of the original with a gap of one statement inserted.

- *Type IV Clones:*

Type IV clones are the results of semantic similarity between two or more code fragments. In this type of clones, the cloned fragment is not necessarily copied from the original. Two code fragments may be developed by two different programmers to implement the same kind of logic making the code fragments similar in their functionality. Functional similarity reflects the degree to which the components act alike, i.e., captures similar functional properties and similarity assessment methods rely on matching of pre/post-conditions.

II. **RELATED WORK**

In the existing system we use a tool called as the dup tool. The Dup tool[5] uses a line-by-line parameterized match to identify code portions that differ in semantic substitution of variable and constant names. This tool do not easily scale up because it is expensive. A string matching algorithm is also applied by the Duploc tool[4]. It offers a clickable matrix display that allows users to visually inspect the source code that produced the match. Like our semi automated approach, automatic clone detection and visual exploration of code are combined to guide refactoring. Here a viewing tool is presented that identifies exact repetitions of text using fingerprints, which are short strings used in place of larger data objects for more efficient comparisons. Sif is based on the same approach. CCFinder concatenates the tokens of a single file into a single token sequence, skipping whitespaces and applying transformation rules, such as replacement of variables with special tokens.

From all the substrings in the transformed token sequence, equivalent pairs are detected as clone pairs. Since this tool uses line by line parameterized match, this tool does not easily scale up because this is expensive.

III. PROPOSED SYSTEM

Our aim is to create a tool in JAVA which is used to find the potential function clones in HTML pages. The main aim is to identify all the various types of clones. Recently, clone identification has been proposed for static web documents, 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. The main advantages of this system is that it can detect the function clones which is present in the web pages with the help of the java script.

Our approach uses as input the report of potential cloned functions as a guide to the visual inspection of code. The goal of this stage is to check whether homonym script functions can be actually considered clones, and to identify the opportunities of refactoring

The first level, *Identical*, holds when the two functions are exactly equal, because no changes have been applied after the copy. This is the simplest occurrence of function cloning: it does not matter which of the copies will be taken off to eliminate duplication.

The second level, *Nearly-identical*, occurs when the two functions differ for modifications which have no effect on output or application state.

Analogously to the first level, any of the cloned functions can be removed during refactoring, but you need to choose which copy will be discarded. Changes to indentation, comments, and blank lines surely fall in this class.

The third level, *Similar*, takes place when two script functions have common characteristics, such as same code structure and same expressions, but refactoring will require changes to unify the functions. This may happen when the two function clones have different output statements or work on different inputs.

The fourth level, *Distinct*, occurs when two script functions, albeit homonym, differ so much in what they do (and then in how they do it) that any refactoring for eliminating duplication would not make sense.

IV.DESIGN OF ARCHITECTURE

The architecture consists of four modules as shown in fig 1.2 are:

1. Transformation
2. Clone Detection
3. Clone Extraction
4. Clone Documentation

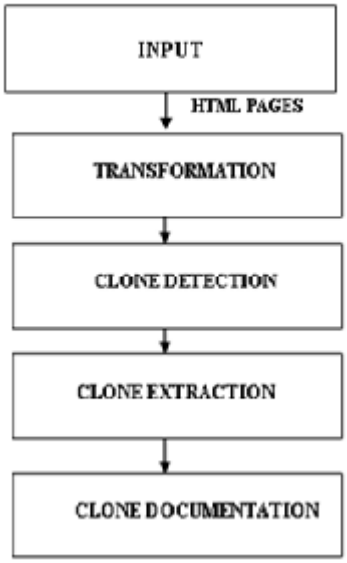

Fig:1.2 PROPOSAL MODULES

1. *Transformation*

The comparison units of the source code are transformed to another intermediate internal representation for ease of comparison or for extracting comparable properties. This transformation is just removing the whitespace and comments to very complex representation and/or extensive source code transformations.

- Pretty printing of source code:

Pretty printing is a simple way of reorganizing the source code to a standard form. By applying pretty printing, source code of different layouts can be transformed to a common standard form. Pretty printing is normally used by the text-based clone detection approaches to avoid the false positives that occur due to the different layouts of the similar code segments. Cordy et al. use an *extractor* to generate separate pretty-printed text file for each of the potential clones obtained using an island grammar.

- Removal of comments:

Most of the approaches ignore/remove comments from the source code before performing the actual comparison. Marcus & Maletic search for similarities of concepts extracted from comments and source code elements. Mayrand et al., on the other hand, use metrics to measure the amount of comments and use that metric as a measuring metrics to find clones.

- Removal of whitespace:

Almost all the approaches (except line-based approaches) disregard whitespace. Line-based approaches remove all whitespace except line breaks. Davey et al use the indentation pattern of pretty printed source text as one of the features for their attribute vector. Mayrand et al. use layout metrics like *number of non-blank lines*.

- Transformation of program elements:

In addition to identifier normalizations, several other transformation rules may be applied to the source code elements. In this way, different variants of the same syntactic element may treat as similar to find clones.

2. *Clone Detection*

The transformed code is next input to a suitable comparison algorithm where transformed comparison units are compared to each other to find a match. Using the order of the comparison units, adjacent similar units are summed up to form larger units. For fixed granularity clones, all the comparison units that belong to a source unit are aggregated. For free granularity clones, on the other hand, aggregation is continued as long as the aggregated sum is above a given threshold for the number of aggregated comparison units. This makes sure that

the aggregation is continued until the largest possible group of comparison units is found. The output is a list of matches with respect to the transformed code. These matches are either already in the clone pair candidates or have to aggregate to form clone pair candidates. Each clone pair is normally represented with the location information of the matched fragments in the transformed code.

3. Clone Extraction

In this phase, false positive clones are filtered out with manual analysis and/or a visualization tool.

- Manual Analysis

After extracting the original source code, raw code of the clones of the clone pairs are subject to the manual analysis. In this phase, false positive clones are filtered out.

- Visualization

The obtained clone pair list can be used to visualize the clones with a visualization tool. A visualization tool can speed up the process of manual analysis for removing false positives or other associated analysis.

4. Clone Documentation

After the clones are being detected and are extracted, then finally the clones that are extracted are identified and all the clones are documented according to their respective types for the further purpose.

V. CONCLUSION

The model proposed here outlines an algorithm capable of detecting all four clone types. By this identification the line of code of the program is being reduced which reduces the complexity for the programmer. Here we present 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 classification of suspect clones is performed through visual inspection of source code. We took the chance to remove function clones in the web application for which there were more duplicates. However, that we found a high number of function clones does not mean that we were able to identify most of code duplication. In fact, our approach looks for clones at the function level but it does not detect duplicated code at lower granularity levels. Our approach might be refined, by developing specific redesign patterns for web applications to provide a guide to duplication removal. The automatic selection of potential clones could also be extended to other web enabling technologies, such as Java Server Pages or PHP.

REFERENCES

[1].Yue Jia, David Binkley, Mark Harman, Jens Krinke and Makoto Matsushita. KClone: A Proposed Approach to Fast Precise Code Clone Detection. International Workshop on Software Clones (IWSC) – 2009.

[2].Elmar Juergens, Florian Deissenboeck, Benjamin Hummel. Clone Detective – A Workbench for Clone Detection Research. International Conference on Software Engineering (ICSE) – 2009.

[3].S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo.Comparison and evaluation of clone detection tools. IEEE Transactions on Software Engineering, 33(9):577–591, 2007.

[4].Ducasse, S., Rieger, M. and Demeyer, S. A Language Independent Approach for Detecting Duplicated Code. in International Conference on Software Maintenance, (Oxford, U.K., 1999).109-118.

[5]. Baker, B.S. On Finding Duplication and Near-Duplication in Large Software Systems. in SecondWorking Conference on Reverse Engineering, (Toronto, Canada, 1995). 86-95.

[6].Eytan Adar and Miryung Kim. SoftGUESS: Visualization and Exploration of Code Clones in Context. In the proceedings of the 29th International Conference on Software Engineering (ICSE'07), Tool Demo, pp.762-766, Minneapolis, MN, USA, May 2007 .

[7].Brenda S. Baker. Finding Clones with Dup: Analysis of an Experiment. *IEEE* Transactions on Software Engineering, Vol. 33(9): 608-621, September 2007.

[8].Lerina Aversano, Luigi Cerulo, and Massimiliano Di Penta. How Clones are Maintained: An Empirical Study. In Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR'07), pp. 81-90, Amsterdam, the Netherlands, March 2007.