Clone Detection Using DIFF Algorithm For Aspect Mining

Rowyda Mohammed Abd El-Aziz
Department of Computer Science
Faculty of Computers and Information
Helwan University
Cairo, Egypt

Amal Elsayed Aboutabl
Department of Computer Science
Faculty of Computers and Information
Helwan University
Cairo, Egypt

Mostafa-Sami Mostafa
Department of Computer Science
Faculty of Computers and Information
Helwan University
Cairo, Egypt

Abstract— **Aspect mining is a reverse engineering process that aims at mining legacy systems to discover crosscutting concerns to be refactored into aspects. This process improves system reusability and maintainability. But, locating crosscutting concerns in legacy systems manually is very difficult and causes many errors. So, there is a need for automated techniques that can discover crosscutting concerns in source code. Aspect mining approaches are automated techniques that vary according to the type of crosscutting concerns symptoms they search for. Code duplication is one of such symptoms which risks software maintenance and evolution. So, many code clone detection techniques have been proposed to find this duplicated code in legacy systems. In this paper, we present a clone detection technique to extract exact clones from object-oriented source code using Differential File Comparison Algorithm (DIFF) to improve system reusability and maintainability which is a major objective of aspect mining.**

Keywords- aspect mining; reverse engineering; clone detection; DIFF algorithm.

I. INTRODUCTION

In software engineering, it is essential to manage the complexity and evolution of software systems. Hence, decomposing large software systems into smaller units is required. The result of this decomposition is separation of concerns that leads to facilitating parallel work, team specialization, quality assurance and work planning [1].

However, there are some functionalities that cannot be assigned to a single unit because the code implementing them is scattered over many units and tangled with other units. Such functionalities are called *crosscutting concerns* [2]. The existence of these crosscutting concerns leads to reducing maintainability, evolution and reliability of software systems.

Aspect Oriented Software Development (AOSD) is a new programming paradigm that solves the problem of crosscutting concerns existence in legacy systems. Aspect oriented programming modularizes such crosscutting concerns in new units called *aspects* and introduces ways for weaving aspect code with the system code at the appropriate places [3]. The success of aspect oriented programming directs software engineers to a new research area called *aspect mining*. Aspect mining is a specialized reverse engineering process which aims at discovering crosscutting concerns automatically in existing systems. This process improves system maintainability and evolution and reduces system complexity. It also enables migration from object-oriented to aspect-oriented systems in an efficient way [4][5][6]. Aspect mining approaches vary according to the type of crosscutting concerns symptoms they search for. Code duplication is one of the main symptoms of crosscutting concerns. It is considered a major problem for large industrial software systems because it increases their complexity and maintenance cost. So, many clone detection techniques are used to find this duplicated code in legacy systems and will be discussed in details in section 2. In this paper, we present a clone detection technique to extract exact clones from object-oriented source code using Differential File Comparison Algorithm (DIFF).

The basic idea is to find different lines of code between two source code files using Diff Algorithm. As a consequence, the remaining lines of code in both files are identical and considered clones. Clones can then be extracted from files. Finding clones in source code as a symptom of crosscutting concerns helps in improving system reusability and maintainability which is the aim of aspect mining. In section 2, previous work on clone detection techniques is presented. In section 3, we describe the basic idea of the used technique to detect clones in source code. In section 4, experimental work and results are discussed. Finally, conclusion and future work are presented in section 5.

II. PREVIOUS WORK

Previous studies report that about 5% to 20% of software systems contain code duplication which is a consequence of copying existing code fragments and then reusing them by pasting with or without minor modifications instead of rewriting similar code from scratch [7]. Therefore, it is considered a common activity in software development. Developers perform this activity to reduce programming time and effort. However, this activity results into software systems which are difficult to maintain. The reason is that if a bug is detected in a code fragment, other similar code fragments have to be checked for the same bug. Consequently, there is a need for automated techniques that can find duplicated code fragments in source code such as clone detection techniques.

A. Clone Detection Techniques

Clone detection techniques can be categorized into the following [8]:

- String-based techniques (also called text-based techniques): at the beginning, little or no transformation in raw source code is performed; for example, white spaces and comments are ignored. Then, the source code is divided into a number of strings (lines). These strings are compared according to the used algorithm to find duplicated ones [9].
- Token-based techniques: use lexical analysis for tokenizing source code into a stream of tokens used as a basis for clone detection.
- AST-based techniques: use parsing to represent source code as an abstract syntax tree (AST) [10]. Then, clone detection algorithm compares similar sub-trees in this tree.
- PDG-based techniques: use Program Dependence Graphs (PDGs) to represent source code [11]. PDGs describe the semantic nature of source code in high abstraction such as control and data flow of the program.
- Metrics-based techniques: hashing algorithms are used in such techniques [12]. A number of metrics are calculated for each code fragment in source code. Then, code fragments are compared to find similar ones.

B. Clone Terminology

When two code fragments are identical or similar, they are called *clones*. There are four types of clones: Type I, Type II, Type III and Type IV. Each of these four types of clones belongs to one of two classes according to the type of similarity it represents: textual similarity or functional similarity. In this context, clones of Type I, Type II and Type III are categorized under textual similarity and Type IV is categorized under functional similarity [13].

- Type I: is called exact clones where a copied code fragment is identical to the original code fragment except for some possible variations in whitespaces and comments.
- Type II: a copied code fragment is identical to the original code fragment except for some possible variations about user-defined identifiers (name of variables, constants, methods, classes and so on), types, layout and comments.
- Type III: a copied code fragment is modified by changing the structure of the original code fragment, e.g. adding or removing some statements.
- Type IV: in this type, clones have semantic similarity between code fragments. Clones, according to this type, are not necessarily copied from the original code because sometimes, they have the same logic and are similar in their functionalities but developed by different developers.

III. Proposed Technique

In this paper, a clone detection technique is presented using Differential File Comparison Algorithm (DIFF) [14] to detect exact clones in source code files. Our clone detection technique passes through three stages:

- Source code normalization: this stage acts as a preprocessing stage. Our clone detection technique is text-based and, therefore, a little transformation of the source code is needed. White spaces and comments are removed at this stage.
- Differential File Comparison: This is the main stage of the proposed technique. The Differential File Comparison algorithm (DIFF) [14] determines differences of lines between two files. It solves the problem of 'longest common subsequence' by finding the lines that are not changed between files. So, its goal is to maximize the number of lines left unchanged. An advantage of the DIFF algorithm is that it makes efficient use of time and space. So, this idea is used to find differences in source code lines between two files.
- Extracting exact clones: After finding differences in source code lines between the two given source code files using the DIFF Algorithm, the remaining lines of code in both files are identical and considered clones. The complement of the difference between 2 files is determined which results in extracting exact clones from two given source code files.

The main steps of DIFF algorithm are summarized as follows [14]:

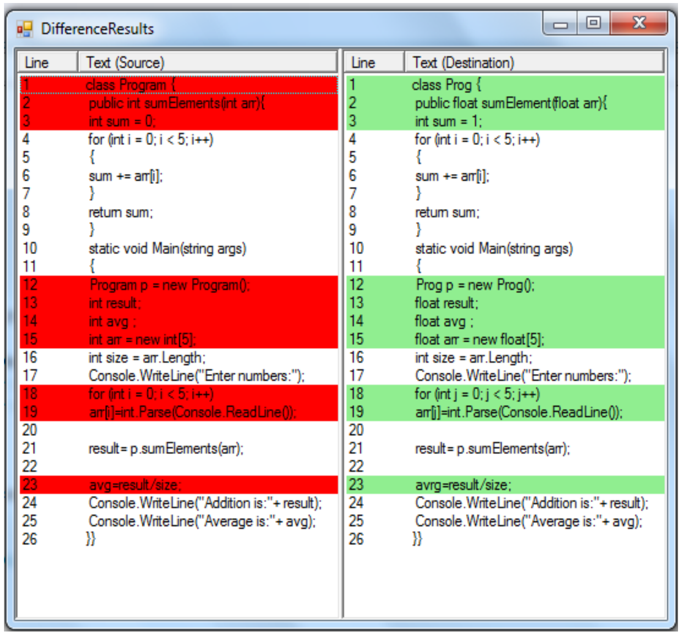
1. Determine equivalence classes in file 2 and associate them with lines in file 1. Hashing is used to get better optimization when comparing large files (thousands of lines).
2. Find the longest common subsequence of lines.
3. Get a more convenient representation for the longest common subsequence.
4. Weed out spurious sequences called jackpots.

IV. Experimental Work And Results

Our experiment was conducted on a simple case study consisting of two source code files implemented in the C# programming language. These files have some differences and similarities in their lines of code as shown in figure 1. At the beginning, the two files are normalized by removing white spaces and comments. Then, they are compared using DIFF algorithm and the differences in source code lines between both files are highlighted as shown in figure 2.

class Program {
 public int sumElements(int[] arr){
 int sum = 0;
 for (int i = 0; i < 5; i++)
 {
 sum += arr[i];
 }
 return sum;
 }
 static void Main(string[] args)
 {
 Program p = new Program();
 int result;
 int avg;
 int arr = new int[5];
 int size = arr.Length;
 Console.WriteLine("Enter numbers:");
 for (int i = 0; i < 5; i++)
 arr[i]= int.Parse(Console.ReadLine());
// sum of array elements
 result = p.sumElements(arr);
 // average of array elements
  avg = result / size;
 Console.WriteLine("Addition is:" + result);
 Console.WriteLine("Average is:" + avg);
}}
	class Prog {
	 public float sumElement(float[] arr) {
	 int sum = 1;
	 for (int i = 0; i < 5; i++)
	 {
	 sum += arr[i];
	 }
	return sum;
	}
	 static void Main(string[] args)
	 {
	 Prog p = new Prog();
	 float result;
	 float avg;
	 float arr = new float[5];
	  int size = arr.Length;
	 Console.WriteLine("Enter numbers:");
	for (int j = 0; j < 5; j++)
	arr[j] = int.Parse(Console.ReadLine());
	// sum of array elements
	  result = p.sumElements(arr);
	 // average of array elements
	  avrg = result / size;
	Console.WriteLine("Addition is:" + result);
	Console.WriteLine("Average is:" + avg);
	 }}

Figure1. Two source code files



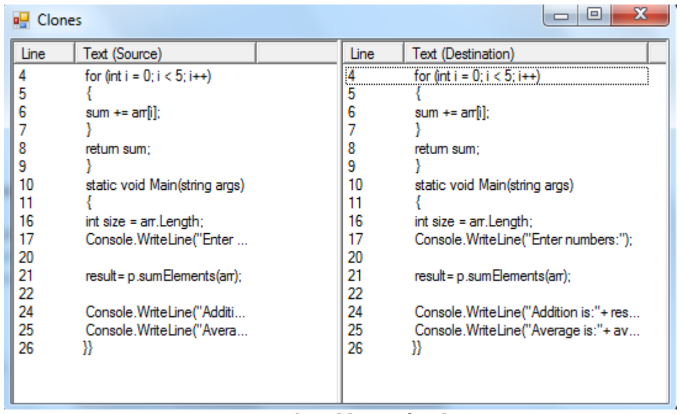
DifferenceResults

Line	Text (Source)	Line	Text (Destination)
1	class Program {	1	class Prog {
2	public int sumElements(int arr){	2	public float sumElement(float arr){
3	int sum = 0;	3	int sum = 1;
4	for (int i = 0; i < 5; i++)	4	for (int i = 0; i < 5; i++)
5	{	5	{
6	sum += arr[i];	6	sum += arr[i];
7	}	7	}
8	return sum;	8	return sum;
9	}	9	}
10	static void Main(string args)	10	static void Main(string args)
11	{	11	{
12	Program p = new Program();	12	Prog p = new Prog();
13	int result;	13	float result;
14	int avg ;	14	float avg ;
15	int arr = new int[5];	15	float arr = new float[5];
16	int size = arr.Length;	16	int size = arr.Length;
17	Console.WriteLine("Enter numbers:");	17	Console.WriteLine("Enter numbers:");
18	for (int i = 0; i < 5; i++)	18	for (int j = 0; j < 5; j++)
19	arr[i]=int.Parse(Console.ReadLine());	19	arr[j]=int.Parse(Console.ReadLine());
20		20	
21	result= p.sumElements(arr);	21	result= p.sumElements(arr);
22		22	
23	avg=result/size;	23	avrg=result/size;
24	Console.WriteLine("Addition is:"+ result);	24	Console.WriteLine("Addition is:"+ result);
25	Console.WriteLine("Average is:"+ avg);	25	Console.WriteLine("Average is:"+ avg);
26	}}	26	}}

Figure2. Difference between lines of code

Finally, exact cloned lines of code are detected in both files after removing those differences from source code lines as shown in figure 3.

Clone Detective tool [15] [16] is a Visual Studio integration that allows analyzing C# projects for source code that is duplicated somewhere else. Clone Detective tool is supposed to detect type I and type II clones but it may miss some clones as explained in [17].



Clones

Line	Text (Source)	Line	Text (Destination)
4	for (int i = 0; i < 5; i++)	4	for (int i = 0; i < 5; i++)
5	{	5	{
6	sum += arr[i];	6	sum += arr[i];
7	}	7	}
8	return sum;	8	return sum;
9	}	9	}
10	static void Main(string args)	10	static void Main(string args)
11	{	11	{
16	int size = arr.Length;	16	int size = arr.Length;
17	Console.WriteLine("Enter ...	17	Console.WriteLine("Enter numbers:");
20		20	
21	result= p.sumElements(arr);	21	result= p.sumElements(arr);
22		22	
24	Console.WriteLine("Additi...	24	Console.WriteLine("Addition is:"+ res...
25	Console.WriteLine("Avera...	25	Console.WriteLine("Average is:"+ av...
26	}}	26	}}

Figure3. Cloned lines of code

By comparing our results with those obtained from the Clone Detective tool for Visual Studio 2008 using the same case study; it is found that the Clone Detective tool cannot detect all the differences in lines of code whereas our proposed technique can do that.

Table 1 shows the results of comparing the two tools regarding the total number of lines in each file and the total number of cloned lines between two files with setting clone minimum length equals to one. It is noticed that our proposed technique can detect all exact cloned lines which are actually 14 lines but Clone Detective tool detects 24 cloned lines and this is not accurate because only 14 lines are exact clones and other lines are different.

Table1.Comparison of results obtained by the proposed technique and the Clone Detective tool

Comparison		Total number of lines	Total number of cloned lines
Proposed Technique	Source	26	14
	Destination	26	14
Clone Detective	Source	26	24
	Destination	26	24

V. CONCLUSION AND FUTURE WORK

We present a simple clone detector to discover code cloning which is a symptom of crosscutting concerns existence in software systems. Detection of code clones decreases maintenance cost, increases understandability of the system and helps in obtaining better reusability and maintainability which is the aim of aspect mining .The technique is experimented on a simple case study (two source code files) and finally exact clones are extracted from source code.

We consider this tool as a starting point towards a complete clone detection system. In the future, this tool can be extended to detect type II and type III clones and mine source code written in other programming languages, not only C#. It can also be extended to work on more than two source code files.

REFERENCES

[1] Arie van Deursen, Marius Marin and Leon Moonen, “Aspect Mining and Refactoring”, In Proceedings of the First International Workshop on REFactoring: Achievements, Challenges, Effects (REFACE03), 2003.

[2] Bounour Nora and Ghoul Said, “A model-driven Approach to Aspect Mining”, Information Technology Journal ,vol.5, 2006 , pp. 573-576.

[3] M.Marin, A.vanDeursen and L.Moonen ,“Identifying Crosscutting Concerns Using Fan-In Analysis”,ACM Transactions on Software Engineering and Methodology, Vol. 17, December 2007.

[4] Bounour Nora, Ghoul Said and Atil Fadila, “A Comparative Classification of Aspect Mining Approaches”, Journal of Computer Science,vol. 2 , pp. 322-325, 2006.

[5] Chanchal Kumar Roy, Mohammad Gias Uddin, Banani Royand Thomas R. Dean,“Evaluating Aspect Mining Techniques: A Case Study”, 15th IEEE International Conference on Program Comprehension (ICPC'07), 2007.

[6] Andy Kellens, Kim Mens, and Paolo Tonella, “A Survey of Automated Code-Level Aspect Mining Techniques”,In Transactions on Aspect Oriented Software Development, Vol. 4 (LNCS 4640), pp. 145-164, 2007.

[7] Chanchal Kumar Roy and James R. Cordy, “A Survey on Software Clone Detection Research”, Technical Report No.2007-541, School of Computing,Queen's University, KingstonOntario, Canada, September 2007.

[8] Magiel Bruntink, “Aspect Mining using Clone Class Metrics”, In Proceedings of the 1st Workshop on Aspect Reverse Engineering, 2004.

[9] Kunal Pandove,“Three Stage Transformation for Software Clone Detection”, Master Thesis,Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Deemed University,May 2005.

[10] Ira D. Baxter, Andrew Yahin,Leonardo Moura, Marcelo Sant’Anna and Lorraine Bier,“Clone Detection Using Abstract Syntax Trees”, In Proceedings of the 14th International Conference on Software Maintenance (ICSM'98), pp. 368-377, Bethesda, Maryland, November 1998.

[11] Jens Krinke, “Identifying Similar Code with Program Dependence Graphs”, In Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01), pp. 301-309,Stuttgart, Germany, October 2001.

[12] Jean Mayrand, Claude Leblanc and Ettore M. Merlo, “Experiment on the Automatic Detectionof Function Clones in a Software System Using Metrics”, In Proceedings of the International Conference on Software Maintenance (ICSM '96),1996.

[13] Yogita Sharma “Hybrid Technique for Object Oriented Software Clone Detection”, Master Thesis,Computer Science and Engineering Department,Thapar University, June 2011.

[14] J.W.Hunt and M.D.McIlroy, “An Algorithm for Differential File Comparison”, Bell Laboratories, Murray Hill, New Jersey, 1976.

[15] http://clonedetectivevs.codeplex.com, last accessed Augest 2012.

[16] Elmar Juergens, Florian Deissenboeck and Benjamin Hummel, “CloneDetective–A Workbench for Clone Detection Research”, In Proccedings of the 30th International Conference on Software Engineering (ICSE), 2009.

[17] Chanchal K. Roy, James R. Cordy and Rainer Koschke,“Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach”, Science of Computer Programming Journal, February 2009.

AUTHORS PROFILE

Rowyda Mohammed Abd El-Aziz is currently a Software Developer at the Ministry of Planning, Cairo, Egypt. She worked as Teaching Assistant in Modern Sciences and Arts University in Egypt for four years. She is a Masters Student at the Computer Science Department, Faculty of Computers and Information, Helwan University, Cairo, Egypt. Her current research interests include software engineering and Human Computer Interaction.

Amal Elsayed Aboutabl is currently an Assistant Professor at the Computer Science Department, Faculty of Computers and Information, Helwan University, Cairo, Egypt. She received her B.Sc. in Computer Science from the American University in Cairo and both of her M.Sc. and Ph.D. in Computer Science from Cairo University. She worked for IBM and ICL in Egypt for seven years. She was also a Fulbright Scholar at the Department of Computer Science, University of Virginia, USA. Her current research interests include parallel computing, image processing and software engineering.

Mostafa-Sami M. Mostafa is currently a Professor of computer science, Faculty of Computers and Information, Helwan University, Cairo, Egypt. He worked as an Ex-Dean of faculty of Computers and Information Technology, MUST, Cairo. He worked also as an Ex-Dean of student affairs and Ex-Head of Computer Science Department, faculty of Computers and Information, Helwan University, Cairo, Egypt. He is a Computer Engineer graduated 1967, MTC, Cairo, Egypt. He received his MSC 1977 and his PhD 1980 from University of Paul Sabatier, Toulouse, France. His research activities are in Software Engineering and Computer Networking. He is awarded supervising more than 80 Masters of Sc. and 18 PhDs in system modeling and design, software testing, middleware system development, real-time systems, computer graphics and animation, virtual reality, network security, wireless sensor networks and biomedical engineering.