Volume 3, Issue 5, May 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Efficient Data Mining Approach for Complex Clone Detection in Software

Swarupa S. Bongale
CSE dept., D. Y. Patil College of Engg. & Tech. Shivaji University, Kolhapur, India

Prof. K. B. Manwade
CSE dept., Ashokrao Mane Group of Institutions, Vathar tarf Vadgaon Shivaji University, Kolhapur, India

Prof. G. A. Patil
CSE dept., D. Y. Patil College of Engg. & Tech. Shivaji University Kolhapur, India

Abstract*— *A code clones are similar or identical fragments of code in a single source file or multiple source files. With the introduction of Cloning in software systems is known to create problems during software maintenance phase of software life cycle process. To detect similar code fragment, called simple clones. Recurring patterns of simple clones often indicates big picture of higher-level similarities, called complex clones. The proposed scheme concentrates on find out more complex types of code similarities. The novelty of approach includes the formulation of the complex clone concept and the application of data mining techniques to detect higher-level code similarities in software.

Keywords*— *Software clones, higher-level similarities, clone instance, software reuse, pattern mining, maintenance.

I. INTRODUCTION

Software development, testing, implementation and the extension over time is the major software life cycle and there are a number of software engineering techniques to deliver efficient software. But the actual Skelton of software is its written code. The working efficiency of the software reflects its quality and strength but it does not reflect the strength of the inner implementation of code which is directly related with the enhancement and maintenance. Software maintenance cost contributes major part to the total development cost. This research basically focuses over the clone components of software. Similar program structures are called code clones, commonly found in software systems. Software clones may increase or decrease the cost, size and complexity of software maintenance. Cloning is active area of research, with multiple clone detection techniques has been proposed in the literature [5], [6], [1], [8]. Duplication may complicate the changes in software. Any missing can leads to update. Many different techniques and tools have been concentrated in this research regarding code clone detection, ranging from similar code fragments (simple clones) to design-level similarities or higher-level clones (complex clones). Existing researches suggest that the code clone or duplicated code is one of the main factors that degrades the design and the structure of software and lowers the software quality such as readability, changeability and maintainability. Recent research has provided evidence that it may not always be practical, feasible, or cost-effective to eliminate certain clone groups. Copying and pasting source code is common practice, also known as software reuse. Sometimes these clones are modified slightly to adapt them to their new environment or purpose. When programmers copy, paste, and then modify source code, the once-identical code fragments (code clones) can become indistinguishable as the software evolves over time. It is believed that identical or similar code fragments in source code, also known as code clones, have an impact on software maintenance. The limitation of considering only simple clones is known in the field [9]. Some clone detection tools are reported to simple clone in a huge number of ways. Another way is to detect clones of larger granularity than simple clones [1], [9].

II. LITERATURE REVIEW

Code clones are similar program structures of considerable size and significant similarity. Several techniques have been proposed to detect code clones. These techniques differ in the code representation used for analysis of clones, ranging from plain text to parse trees, Abstract Syntax Suffix Trees, Program Dependence Graphs and Code Clone Finder etc.

[1] A Multi-Linguistic Token-Based Code Clone Detection System for Large Scale Source Code: Proposes a new clone detection technique, which consists of transformation of input source text and token-by-token comparison. They used a suffix-tree matching algorithm to compute matching, in which the clone location information is represented as a tree with starting nodes for leading identical sub-sequences, and the clone detection is performed by searching the leading nodes on the tree but, this approach have certain issues in clone detection like identification of structures, regularization of identifiers, measuring clones etc. For its implementation with several useful optimization techniques, they developed a tool named CCFinder, which extracts code clones in C, C++, Java, COBOL and other source files.

[2] Identification of High-Level Concept Clones in Source Code: This approach is to examine the source code text (comments and identifiers) and identify implementations of similar high-level concepts (e.g., abstract data types). The approach uses an information retrieval technique (i.e., latent semantic indexing) to statically analyze the software system

and determine semantic similarities between source code documents (i.e., functions, files, or code segments). These similarity measures are used to drive the clone detection process but, in some cases, the developers choose to entirely rename the data structure and operation names in a cloning (re-implementation) of a list (e.g., a list of new records), then comments are also discarded, this technique is unable to detect such similarities.

[3] A Language Independent Approach for Detecting Duplicated Code: Techniques for detecting duplicated code exist but rely mostly on parsers, technology that has proven to be brittle in the face of different languages and dialects. In this paper shown that is possible to circumvent this hindrance by applying a language independent and visual approach, i.e. a tool that requires no parsing, it is able to detect a significant amount of code duplication. This technique is scalable but they want to qualify how much of duplication they are missing.

[4] "Cloning Considered Harmful": Patterns of Cloning in Software: Describes several patterns of cloning that have observed in various case studies and discusses the advantages and disadvantages associated with using them. This paper introduces eight cloning patterns that have uncovered during case studies on large software systems. These patterns present both good and bad motivations for cloning, also introduces the notion of categorizing high-level patterns of cloning in a similar fashion to the cataloging of design patterns or anti-patterns. There are several benefits that can be gained from this characterization of cloning but, there are several problems associated with cloning. Code cloning can lead to unnecessary increase in code size. Cloning code can lead to unused, or "dead", code in the system that left unchecked can cause problems with code comprehensibility, readability, and maintainability over the life time of the software system. Maintenance efforts can be increased when bugs have to be fixed multiple times, and these changes could be prone to errors. Clones of code that is not well understood can introduce bugs. For example, variables may be shared and modified unknowingly. Program comprehensibility can be affected by the increased code size, as well as the need to understand the differences between the duplicates.

Literature on these topics of code cloning often asserts that duplicating code within a software system is a bad practice, that it causes harm to the system's design and should be avoided.

III. PROPOSED APPROACH

A. Problem Definition

Developing a technique to detect some useful types of complex clones. The novelty of approach includes find out simple clones as well as, the formulation of the complex clone concept and the application of data mining techniques to detect these higher-level similarities.

As an input give a single source file or multiple files. You can give input file as a .txt file of c, cpp or java language, applying a customizable tokenization strategy, used pattern mining algorithm to find out repetitive patterns called code clones. We introduced the concept of complex clones (higher-level similarities) as a repeating configuration of lower-level clones. Classify the complex clone types such as

1) Similar Classes
2) Similar Functions
3) Similar Structures
4) Similar patterns within { }
5) Finds all higher-level clones including simple clones

B. Block Schematic for Proposed System

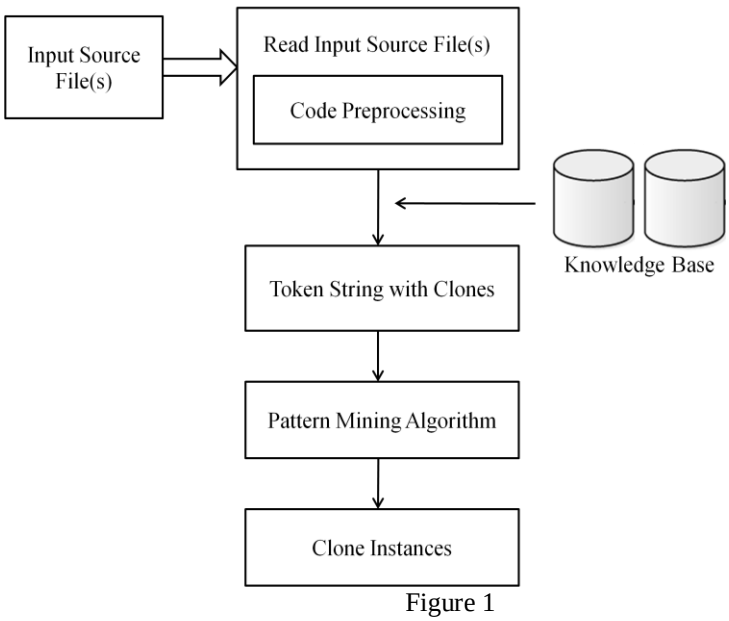


Figure 1

Figure1 shows block diagram of proposed system. The main idea behind our enhancement in this approach is to achieve better code similarity (clones).

As an input give single source file (.txt file) or give a folder which containing multiple source files of c or cpp or java language.

The algorithmic working is as follows:
Step 1: Perform the Code Pre-processing.
Step 2: Find out Token String with Clones.
Step 3: Apply Pattern Mining algorithm to find out repeated code patterns.
Step 4: Find out the clone instances.

C. Methodology

1) *Code Preprocessing:* Detection of code clones provides useful information for maintenance, reengineering, program understanding, and reuse. Clone detection based on lexical tokens involves minimal source language dependency and has good recall.

Table 1 shows a simple tokenization scheme. A single large token string is generated from the input source file(s). Identical segments of this token string are reported as clones, which can be exact clones according to the proposed tokenization table.

Table1: Tokens with their unique id's

ID	KeyChars	ID	KeyChars	ID	KeyChars	ID	KeyChars	ID	KeyChars
57	include	75	return	53	using	128	native	6	[
58	cout	76	static	81	abstract	129	package	7	]
59	if	77	do	54	function	130	strictfp	24	,
60	void	41	while	55	new	131	super	25	"
61	int	42	if	56	constants	132	synchronized	0	0
62	float	43	for	82	open	133	throws	26	.
63	for	44	catch	83	close	134	transient	27	'
64	clrscr	45	switch	84	eof	135	^	28	&&
65	char	46	case	85	bof	1	#	29	||
66	long	47	else	86	binary	2	<<	30	&
67	public	78	bool	87	seekg	3	<	31	|
68	protected	48	true	88	seekp	4	>>	32	%
69	private	49	false	89	end	5	>	33	_
70	friend	50	short	90	try	12	(	34	0
71	getch	79	signed	91	throw	13	)	35	~
72	class	80	unsigned	92	terminate	8	{	36	!
73	this	51	return	93	append	9	}	37	?
74	virtual	52	namespace	100	register	10	::	38	->
94	empty	97	break	101	typedef	11	;	126	instanceof
95	auto	98	enum	102	extern	14	++	127	interface
96	double	99	struct	103	union	15	--	23	=
104	const	108	goto	112	operator	16	+	22	:
105	static	109	sizeof	113	delete	17	-	21	|
106	continue	110	volatile	114	asm	18	/	116	mutable
107	deault	111	inline	115	typename	19	*	117	assert
122	final	120	default	124	implements	20	\	118	boolean

The source file(s) are transferred into token string. Each identifier related to types, variables, and constants is replaced with a special token. Token and its related ID is stored in database shown in table1.

2) *Token String with clones:* Our requirements for clone detection differ from the other tools and techniques. Hence propose a customizable tokenization strategy. In this scheme, a separate integer ID is assigned to each token found in the source code. Identical segments of these id's are reported as clones. The classification of tokens is totally customizable. For example, if the user does not want to differentiate between the types {int, short, long, float, double}, we can have the different ID to represent every member of the above set of types. In this way, all those code fragments that differ only in the type of certain variables become exact replicas of each other in the token string.

3) ***Pattern Mining:*** This pattern mining step is designed to handle set-typed data, where multiple values occur; thus, a naive approach is to discover repetitive patterns in the input. However, there can be many repetitive patterns discovered and a pattern can be embedded in another pattern. We detect every consecutive repetitive pattern and merge them (by deleting all occurrences except for the first one) from small length to large length.

It shows, repeating line numbers and their related pattern only once. Also shown count of which pattern is how many times repeated.

Pattern mining algorithm:
Given: Input source file(s).
Step 1: Computes possible pattern length and return maximum pattern length for all patterns in the list.
Step 2: Starting from smallest pattern length, that looks for first pattern in the list.
Step 3: starting pattern compare with next occurrence of pattern, if match founds returns true.
Step 4: The algorithm continues to find more matches of patterns until the end of the list has encountered.
Step 5: If a pattern is detected, the algorithm modifies the list by deleting all occurrences of the pattern except for the first one.
Step 6: Finally, recomputed the possible pattern length for each pattern in the modified list, reinitializes the variables to be ready for a new repetitive pattern and continues the comparisons for any repetitive patterns in the given list of patterns.

4) ***Clone Instance:*** A clone relation is an equivalence relation (i.e., reflexive, transitive, and symmetric relation) on code portions. A clone relation holds between two code portions if (and only if) they are the same sequences. For a given clone relation, a pair of code portions is called clone pair if the clone relation holds between the portions. An equivalence class of clone relation is called clone class. That is, a clone class is a maximal set of code portions in which a clone relation holds between any pair of code portions.

The found clone instances from input source file(s) are highlighted in different color. So, we can easily distinguish they as complex clones
1) Similar Classes
2) Similar Functions
3) Similar Structures
4) Similar patterns within { }
5) View repeated patterns- Finds all repeated patterns including simple clones.

To find clone instances following is the procedure:
Four main menus:
Example: Here given an example a busrervation.txt project file.
1. Home: Read the input source file(s).

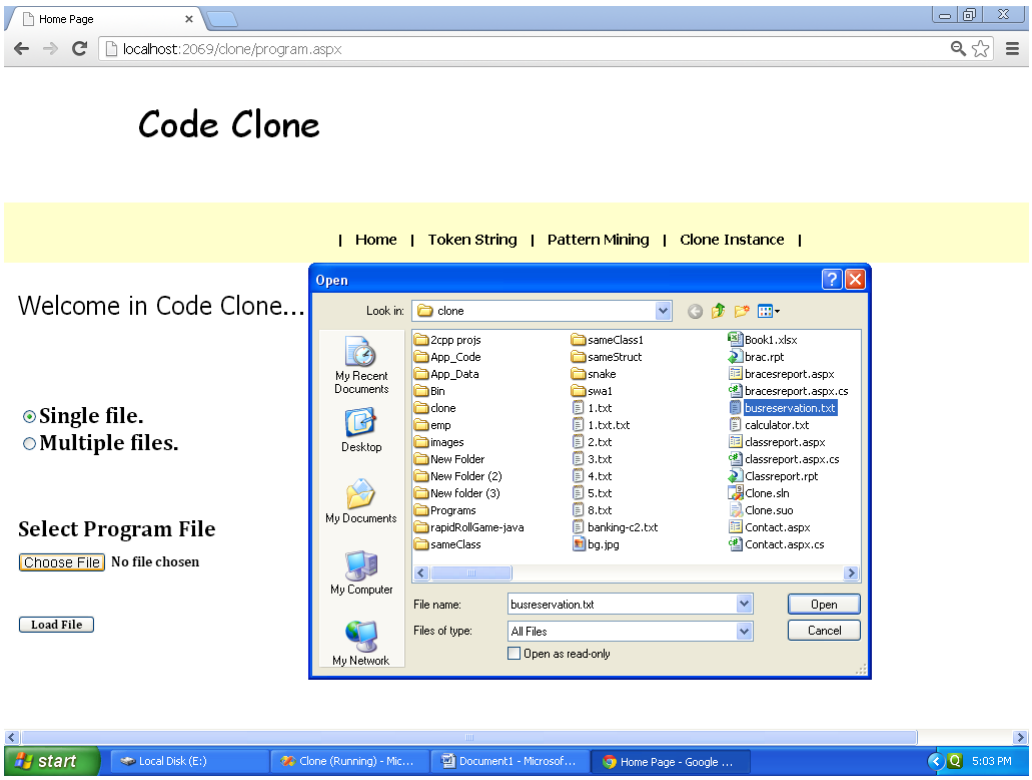


Figure2: Selection of input source file (Example-busreservation.txt)

2. Token String with clones: Perform the code preprocessing, shows token string with their related id's. Identical id's are reported as clones.

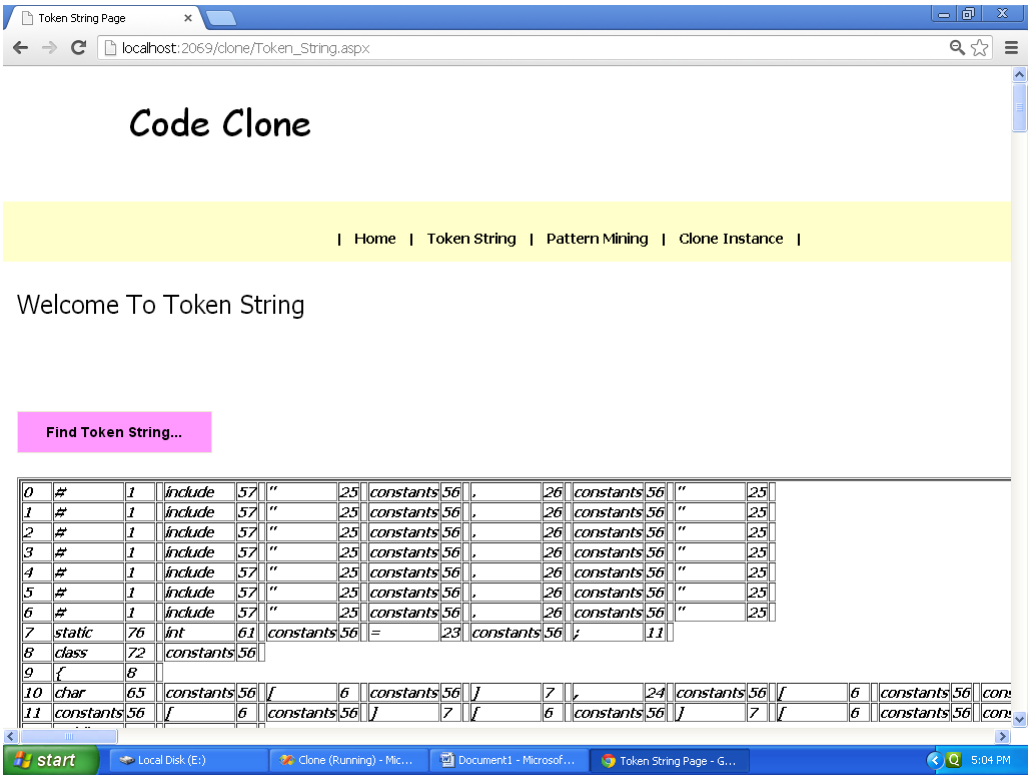


Figure3: Token string with clones (Example-busreservation.txt)

3. Pattern Mining: Apply the pattern mining algorithm to find clone instances. It shows repeated pattern line numbers and their related pattern id's.

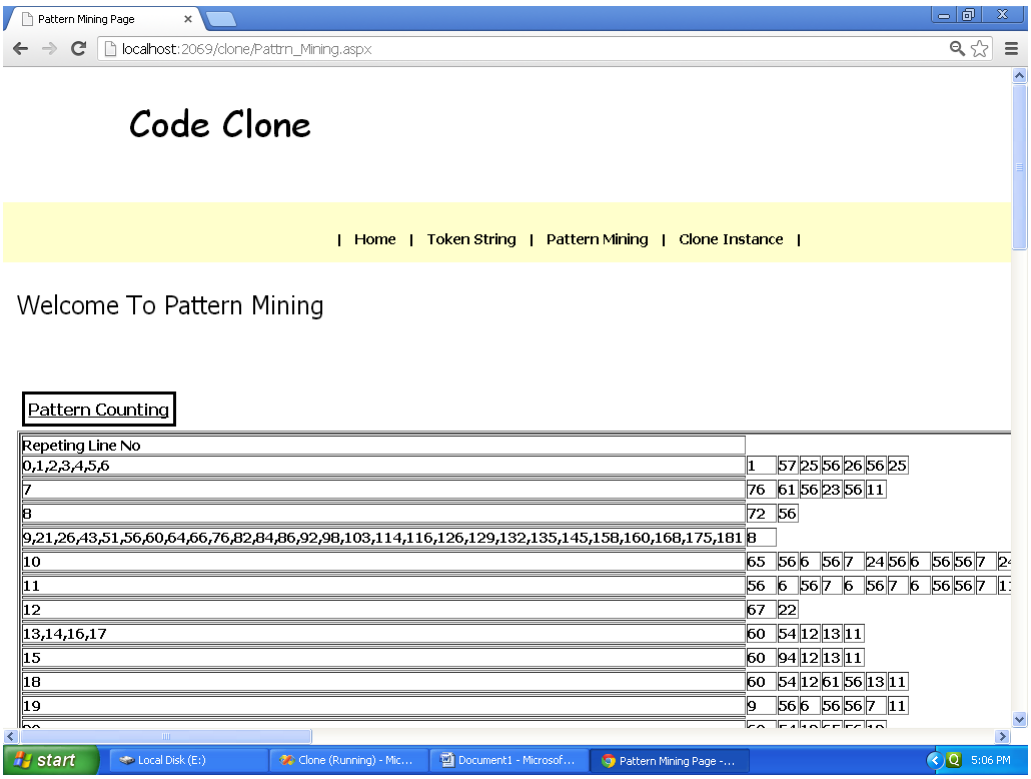


Figure4: Pattern mining process (Example-busreservation.txt)

4. Clone Instance: Find out similar clone patterns. It shows similar code fragments i.e. Four buttons: similar classes, similar functions, similar structures, and similar patterns within { } in different highlighted color.

Two buttons:

1) View Program – Open source code of input project file in notepad.

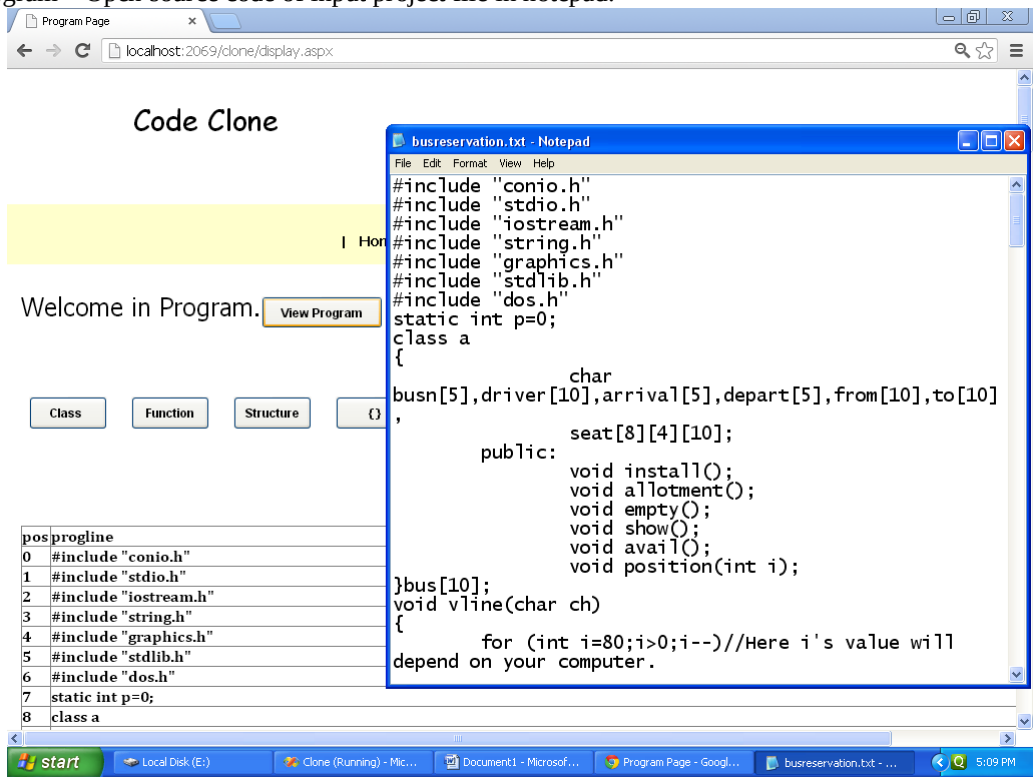
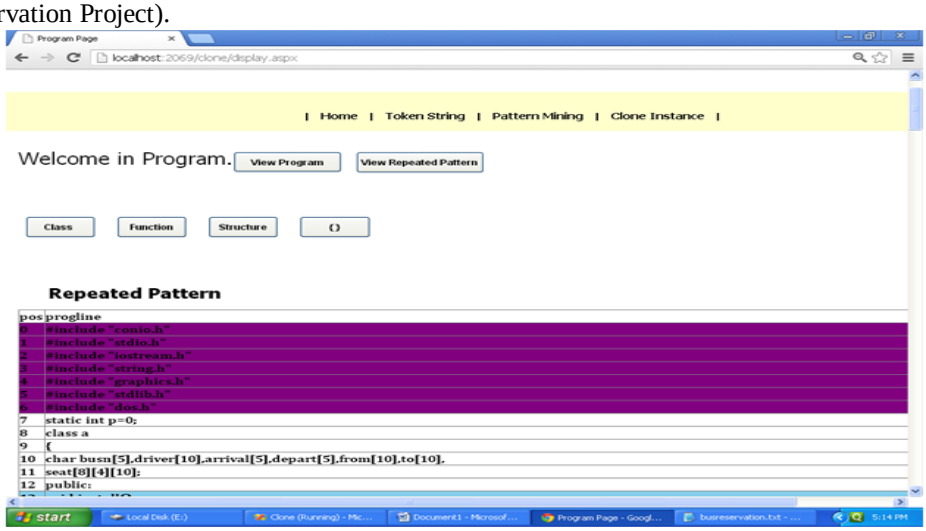


Figure5: View program (Example-busreservation.txt)

2) View Repeated Pattern – It shows all repeated patterns i.e. simple clones as well as higher-level clones in project file. (Ex. Bus Reservation Project).



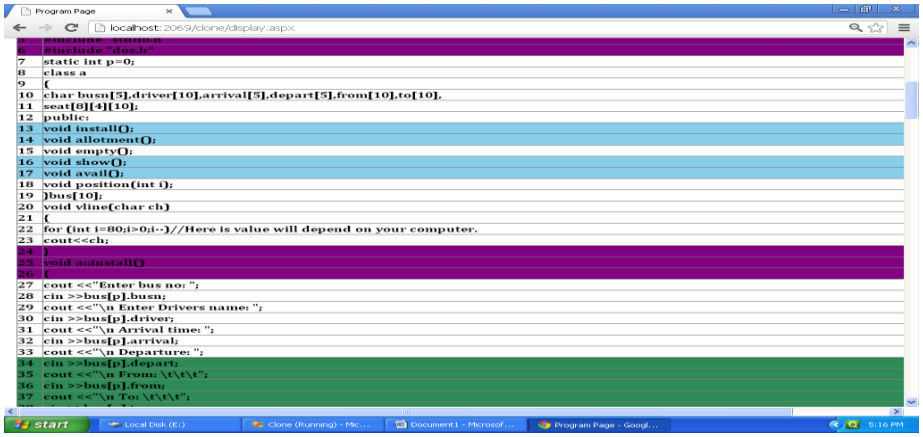


Figure6: View repeated patterns (Example-busreservation.txt)

D. Experimental Results

*1) **For Single File –*** Here as an input given a single source file.

Project Source File Name	Language	Total Number of tokens presents in project	Similar Classes	Similar Functions	Similar Structures	Similar Pattern within { }	View Repeated Patterns
Bus Reservation System	c	1632	No	Yes	No	Yes	Yes
Student Record System	c	2881	No	No	No	Yes	Yes
Telephone Billing System	cpp	3657	No	Yes	No	Yes	Yes
Super Market	cpp	2645	Yes	Yes	No	Yes	Yes
Student Management	java	5005	No	Yes	No	Yes	Yes
Calculator	java	1338	No	Yes	No	Yes	Yes

*2) **For Multiple Files -*** Here as an input given a single folder, this contains multiple source files.

Project Folder Name	Language	Total Number of tokens presents in project	Similar Classes	Similar Functions	Similar Structures	Similar Pattern within { }	View Repeated Patterns
Library Management	c	6507	No	Yes	No	Yes	Yes
Snake Game	c	1901	No	Yes	No	Yes	Yes
Video Store	cpp	3851	No	Yes	No	Yes	Yes
Employee Database System	cpp	3663	Yes	Yes	No	Yes	Yes
Hospital Management	java	3239	No	Yes	No	Yes	Yes
Rapid Roll Game	java	2819	No	No	No	Yes	Yes

IV. CONCLUSIONS

Code clone are similar program structures that comes in variant forms in software systems. In software systems, maintenance phase playing an important role in software life cycle process. Software maintenance cost contributes part to the total development cost. We introduced the concept of complex clones (higher-level similarities) as a repeating configuration of lower-level clones. The process starts by finding simple clones (that is, similar code fragments). Increasingly higher-level similarities are then found incrementally using data mining technique, pattern mining algorithm.

Our technique is both scalable and useful. Simple clones as well as complex clone information leads to better program understanding, helps in different maintenance related tasks, and points to potential reusable components across a Product Line. Complex clones are also candidates for unification with generic design solutions. After such unification, programs are easier to understand, modify and reuse.

ACKNOWLEDGMENT

Many thanks are due to the following people for their invaluable support and guidance at different stages of the research and development: Prof. K. B. Manwade and Prof. G. A. Patil.

REFERENCES

[1] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: Multi-Linguistic Token-Based Code Clone Detection System for Large Scale Source Code", IEEE Trans. Software Eng.,vol. 28,no. 7,pp. 654-670, July 2002

[2] A. Marcus and J.I. Maletic, "Identification of High-Level Concept clones in Source Code", Proc. Int'l Conf. Automated Software Eng.,pp. 107-114, 2001.

[3] S. Ducasse, M. Rieger, and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code", Proc. IEEE Int'l Conf. Software Maintenance, pp. 109-118, 1999.

[4] Cory J. Kapser and Michael W. Godfrey ,"Cloning Considered Harmful" Considered Harmful: Patterns of Cloning in Software", Software Architecture Group (SWAG) avid R. Cheriton School of Computer Science, University of Waterloo.

[5] B.S. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems," Proc. Second Working Conf. Reverse Eng.,pp. 86-95, 1995.

[6] I.D. Baxter, A. Yahin, L. Moura, M.S. Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees," Proc. IEEE Int'l Conf. Software Maintenance, pp. 368-377, 1998.

[7] R. Koschke, R. Falke, and P. Frenzel, "Clone Detection Using Abstract Syntax Suffix Trees," Proc. 13th Working Conf. Reverse Eng., pp. 253-262, 2006.

[8] A. Walenstein, A. Lakhotia, and R. Koschke, "The Second International Workshop Detection of Software Clones: Workshop Report," SIGSOFT Software Eng. Notes, vol. 29, no. 2, pp. 1-5, Mar.2004.

[9] A. De Lucia, G. Scanniello, and G. Tortora, "Identifying Clones in Dynamic Web Sites Using Similarity Thresholds," Proc. Int'l Conf. Enterprise Information Systems, pp. 391-396, 2004.

[10] G. Grahne and J. Zhu, "Efficiently Using Prefix-Trees in Mining Frequent Itemsets," Proc. First IEEE ICDM Workshop Frequent Itemset Mining Implementations, Nov. 2003.

