An Ameliorated Methodology for Comprehension of Legacy System

Dr. Shivanand M. Handigund
Bangalore Institute of Technology, Bangalore, 56004, India
smhandigund@gmail.com

A. A. Chikkamannur and K. Ananthapadmanabha, and H. R. Shashidhara
REVA Institute of Technology & Management, Bangalore, 560064, India
{ac.ajeet@gmail.com, siridevikap@yahoo.co.in, shashi_dhara@yahoo.com}

Abstract **– The legacy systems are the executable code(s) developed with huge investment, incorporation of changes made in the business rules over a long period of time.These systems are evolved and accumulated perennially the then needs of organization from time to time. The advancement of technology and perennial modification of the code to the changing needs of the business have weakened the productivity of these legacy systems and have put the system on the brink of software crash. This has compelled for a paradigm shift in the technology and knocks the human resource either to waist their precious time in abstracting or sifting useful business rules buried across the legacy system.**

To enable the existing code to be amenable to changes in the business norms and to ease the process of strong cohesion and weak coupling, there is a necessity to understand the code so as to either modify and reuse existing program or migrate the program to another programming language code. There is a need to transform the program code in to natural language text.

The methodology proposed in this paper translates the legacy system to the English language by substituting the English language constructs in the place of token, which are having the finite meaning and reserved by the programming language. Further a platform is provided for common people to understand the code in English language syntax and asking them to abstract the concept by their experience and intelligence. Since the legacy system is expressed in near English language, proficient and non-proficient people are involved in the understanding process, which leads to the correct abstraction of the concept(s).

Index terms **– token; programming language; natural Language; understanding; legacy system**

I. INTRODUCTION

Maintenance and upgradation of a legacy system is the challenging task, when the relevant documentation or reliable information like variable names, comments etc. in a system is not available. On other hand the evolutions of technology have weakened the productivity of a legacy system but, at the same time, these systems are developed by the huge investments with incorporation of business rules over a period of time. The evolution of technology compels the organization to shift their existing system to newer system but the system developers are struggling with the problem of optimizing with oxymoron concept of huge involvement of human resource and freezing long accumulated business rules.

"Program comprehension" is a process of gaining knowledge about the computer program and this knowledge is useful in the activities like bug detection, reuse, reengineering etc. The program comprehension activity is a complex task in the absence of relevant documentation and on the verge of crash.

In the absence of relevant documentation for a legacy system, most of the resources and time is devoted to only maintenance [1, 2, 3] rather than development and the greatest part of a maintenance process is depleted for understanding the system only [4]. Hence, to boost the maintenance or migration process, the process of understanding a legacy system is to be extended.

This paper proposes a conceptual methodology of program comprehension, which takes the advantage of natural language comprehension. The methodology translates the tokens of a source code to the English language tokens for enhancing the readability of the program comprehension process. The translation to a near English language syntax mingles the proficient and non-proficient people in the process of comprehension for abstracting concept in a system.

The section 2 discusses the various methods of program understanding like program slicing, concept assignment. The section 3 presents the framework of methodology with illustration with C source code. The section 4 is the conclusion and future work.

II. BACKGROUND

This section presents the some methods of program comprehension.

Manuscript received October 13, 2009; revised April 10, 2010; accepted April 19, 2010.

All Indian Council of Technical Education (AICTE) F. NO.: 8023 / BOR / RID / RPS – 99 / 2007-08, corresponding author A. A. Chikkamannur

doi:10.4304/jait.1.2.92-96

A. Program Slicing

The slicing is a process of gathering the data definitions and control flow for the desired set of data element values from the slice point and the irrelevant data element is discarded. The concept of program slicing was introduced by Weiser [5, 6] to formulate the process of debugging. Weiser defined a program slice as a collection of program statements affecting certain variable. This slice is with respect to a slicing criterion pair comprising the line number of the program and the affected attribute. This method is called as static backward slicing. The program slicing had become a research topic, which is available in the papers [7, 8, 9, 10].

The ordered pair comprising these attributes along with statement number forms a slicing criterion. This is developed by Weiser and subsequently modified to suit the abstraction of functionality by Phatak and Handigund. Here, in the modified algorithm, they have considered the group of attributes that together affect the program statements. In a program, normally dependent attribute is at the logical end of the program. Thus, by applying this modified algorithm [11] and then by traversing the program statements, in backward control flow order, the directly and indirectly relevant statements are abstracted and will help in the comprehension of the program. By appropriately choosing the attributes group, the statements affecting those attributes, different granularity level functionalities are abstracted to suit different intellectual level skills to understand the program.

Researchers [12, 13, 14, 15] have suggested that program slices are helpful in a program comprehension process because the slice is simplified version of the original program. Then the simplified code is comprehended by the domain expert depending on their intelligence.

The slicing process mines the portion of a code from the program for comprehension but in extreme cases, if there is a no reduction in the code after slicing, then the comprehension of a code is complex task.

B. Concept Assignment

The concept is defined as an abstraction of a reality or human intellectual thinking [16] and the concept assignment is a process of knowing the concepts from the domain of real world and matching them to the portion of system [17]. In the concept assignment process, the code is comprehended by assigning the computational intent to the source code and many researchers [17, 18, 19, 20, 21] contributed in the concept assignment.

The complexity involved in the concept assignment is the identification of a domain oriented concepts and searching them in a legacy system. This needs the domain experts to formulate the concept and that has to be searched by matching with the code but only few domain experts can contribute and their concept may or may not match.

C. Languages

The English language is worldwide popular medium of communication between people to express their thoughts and over a long period. It is serving the purpose without much evolution in the language constructs. The thoughts expressed in natural language covers the comprehensive spectrum of human expression and wide range of the subject matter.

The programming languages are developed to facilitate communication between machine and people [22]. More over, these languages are derived from the English language with constraint laid by the machine representation. Further programming languages are developed depending on the time to time need of the specific users and hence the spectrum of subject expression is narrow.

Two [23] differences between natural language and programming language are observed. First, programming languages have the narrow expressive domain of thought and are expressed in terms of algorithm or computation. Second, programming languages facilitates the communication of algorithmic or computational thoughts between the people and machine.

We propose here that, if the computational thought of programming language is expressed in the natural language then many people can understand the thought and abstract the buried business rule or concept in the system.

III. Frame Work

The proposed methodology reduces the understanding gap between the actual concept of legacy system and the interpreted concept of the system by various intellectual skills. This is achieved by replacing the programming language constructs of finite semantics with their equivalent natural language constructs. The finite meaning programming language tokens includes the keywords (reserve words), operators, built-in function names, symbols etc.

The design of methodology is divided in to two parts: In the first part, the programming language tokens of finite meaning are collected and their equivalent natural language words are stored in a data dictionary of database. These words are collected manually from the document of a particular programming language and their equivalent English language constructs.

Then the source code is read, line by line and each line is decomposed to number of tokens. If the decomposed token is present in the dictionary of database, then the word is replaced by its equivalent English language construct. This will result in mapping of programming language statements to corresponding English language statements. The algorithm for converting legacy source

statements to nearer natural language statements is shown in figure1.

Input: source code
Output: natural language code

1. Read the source code by one line
2. Break the line in to tokens
3. For each token
   If (token == database token)
   Replace token by natural language meaning
4. Repeat steps 1,2 and 3 until end of lines

Figure 1: Algorithm for translation

Token	Natural English language meaning
#include	add the file
<stdio.h>	for standard input and output device
main ()	starting point of execution
int	integer identifier
char	character
getchar	read the characters from file
=	is equals to
==	is equated to
‘ ‘	blank
‘\n’	new line
‘\t’	tab
printf	display the result
;	. (full stop symbol for end of line)

Figure 2 Database of finite meaning tokens

IV. Case Study

To exemplify our methodology, the C programming language code given by the authors Keith B. Gallagher and James R. Lyle [24] is considered and shown in the figure 3. The code suffers with comments, reliable variables names and documentation. In such condition, How to understand the concept amalgamated in the code?

The source code given in the figure 2 will consist of words “#include”, “stdio.h”, “int”, “char”,” ||”, ‘\n” etc and these words have the finite meaning in the natural language. The reserve words of example and their equivalent meaning in natural language are given in the figure 2. (For illustration only few tokens are considered).

The first step of methodology takes the source code as input and starts reading one line at a time. Consider the first line of the example code

#include <stdio.h>

In this statement the token #include is replaced by the “add the file” and the token <stdio.h> by “for standard input and output device”. The equivalent nearer English language word constructs of the statement is:

“Add the file for standard input and output device”

If the token is not available in the database, then the same token is retained in the translated code. The methodology proceeds for entire lines of code and resulting code is in the English language constructs (which is nearer to the English language sentences) and translation of entire example code is given in the figure 4.

```
1    #include <stdio.h>
2    #define YES 1
3    #define NO 2
4    void main( )
5    {
6        int l=0;
7        int w=0;
8        int c=0;
9        int inword = NO;
10       int c=getchar();
11     while(c!=EOF)
12      {
13       char ch= (char) c;
14        c =c + 1 ;
15        if (ch == ‘\n’)
16          1=1+1;
17          if (ch == ‘ ‘ || ch == ‘\n’ || ch ==’\t’)
18                inword = NO;
19          else if (inword = = NO)
20            {
21              inword = YES;
22              w = w + 1:
                }
23        c =getchar( );
        }
24        printf(“ %d \n “, l );
25        printf(“ %d \n “, w);
26        printf(“ %d \n “, c);
    }
```

Figure 3. Example code

When the translated code in figure 4 is closely analyzed, the natural language words like new line, blank, tab, end of file and characters are appeared. The presence of new line, character and word are checked in a file and if they are found, there is a variable added with 1. Our knowledge specifies that *any thing added with 1 for number of times* is nothing but a *counter*. Hence, there is a counter for the new line, word and character and the concept in the code is *to count the number of lines, words and characters.* Once the concept is known, then that can be migrated to any paradigm of technology.

V. Conceptual Results

The developed algorithm will translates the legacy system to the English language sentences (nearer) by the interpretation of fixed meaning tokens in terms of English words existing in the code. All the programming languages are constituted with reserved words and the reserve word’s finite semantics in an English language words is exploited. This will ease the reading of the code for a non-proficient people i.e. naive user of a programming language. Hence, *our methodology bridges the gap between the proficient and non-proficient intellectual skills interpreted knowledge through*

program comprehension process. This is absolutely necessary in the condition, where the reliable information or relevant documentation of the legacy system does not exists and the system is to be migrated to a newer technology.

```
Add the file for standard input and output device
Replace YES 1
Replace NO 2
Void starting point of program
 {
    Integer identifier l is equals to 0
    Integer identifier w is equals to 0.
    Integer identifier c is equals to 0.
    Integer identifier inword is equals to NO.
    Integer identifier c is equals to get char.
    While (c is not equals to the end of file)
     {
       character identifier ch is equal to the character value of c.
       c is equals to c added with 1.
        if (ch is equates to the new line)
       nl is equals to the l added with 1.
       if (ch is equates to blank or ch is equates to new line or
           ch equates to tab)
        inword is equals to NO.
       else if (inword is equates to NO)
        {
            inword is equals to YES.
            w is equals to w added with 1.
        }
       c is equals to the read the character.
    }
      Display the result (" integer value new line ", l).
      Display the result (" integer value new line", c).
      Display the result (" integer value new line", w).
```

Figure 4: Translated code

VI. CONCLUSION

For the comprehension of legacy system, many researchers used the variables names, comments, relevant documents, data structures and algorithms etc. Their techniques are based on variable name, comments etc. The concept of program understanding is based on the natural way of understanding but the comprehension of code is difficult when they are not reliably available. On other way the algorithmic and data structure methods of program understanding are based on the experience of domain expert involved in the analysis process but few people are expert in the domain.

In this methodology the dictionary of database that is used in the mapping between programming language and English language constructs is developed only for a programming language. The methodology abstracts the concept from the legacy code, which suffers from relevant documentation, reliable information by the natural language phrase. A platform is provided to understand the system by translating reserve words or tokens to equivalent English language constructs and inquired to abstract the concept by including proficient and non-proficient people's knowledge.

The methodology exploits the tokens from the legacy system having the finite meaning in English language constructs and extends the program comprehension process. The methodology is illustrated with the example of C code. In future, the process of program comprehension of a legacy system is to be developed for the multi-core architecture system, so that the legacy system is migrated to a multi-core architecture system.

ACKNOWLEDGEMENT

The authors acknowledge the grant provided by All Indian Council of Technical Education (AICTE) under Research Promotion Scheme through it's F. NO.: 8023 / BOR / RID / RPS – 99 / 2007-08.

REFERENCES

[1] Barry. W. Boehm, "Software Engineering Economics". Prentice Hall, 1981

[2] I. Somerville, "Software Engineering", 6th edition, Addison-Wesley 2001

[3] B. Lientz, E. Swanson, and G. E. Tompkins. "Characteristics of Application Software Maintenance." *Communication of the ACM*, 21(6), June 1978

[4] T. M. Pigoski, "Practical Software Maintenance: Best Practices for Managing your Software Investment", Wiley Computer Publishing, 1996.

[5] M. Weiser, "Program Slicing" *IEEE transaction on Software Engineering*, Vol SE-10, No. 4, pp. 352-357. 1984

[6] M. Wieser" Programmers use slices when debugging ", *Communications of the ACM*, vol. 25, no 7, pp.446-452, 1982,

[7] F. Tip, "A Survey of program slicing techniques", *Journal of Programming Language*, Vol 3, pp 121-189, 1995.

[8] D. Binkley and K. B. Gallagher, " Program Slicing", in *Advances in Computers*, Vol 43, Marvin Zelkowitz, Editor, Academic Press, San Diego, CA, 1996, pp. 644-657.

[9] David Binkley and Mark Harman, "A Survey of Empirical Results on Program Slicing", *Advances in Computers*, Vol 62, Academic Press San Diego, CA, 2004.

[10] M. Kamkar,"An overview and comparative classification of program slicing techniques", *The Journal of Systems and Software*, vol 31, 1995, pp. 197-214.

[11] S M. Handigund, "Reverse Engineering of Legacy COBOL systems", Ph. D. thesis Indian Institute of Technology Bombay, 2001

[12] Andrea De Lucia, Anna Rita Fasolino, and Malcolm Munro." Understanding function behaviors through program slicing", In *4th IEEE workshop on Program Comprehension*, Berlin, Germany, March 1996

[13] Andrea De Lucia and Malcom Munro, "Program Comprehension in a reuse reengineering environment", In Malcolm Munro, editor, *1st Durham workshop on program comprehension*, Durham University, UK, July 1995.

[14] Mark Harman, Sebastian Danicic, Yogasundary Sivagurunathan, "Program comprehension assisted by slicing and transformation". In Malcolm Munro, editor, *1st Durham workshop on program comprehension*, Durham University, UK, July 1995.

[15] Daniel Jackson and Eugene. J. Rollins. Chopping: "A generalization of slicing" Technical Report CMU-CS-94-

169, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, July 1994
[16] Vaclav Rajlich, Normath Wilde, "The Role of Concepts in Program Comprehension" *Proceedings of IWPC 2002*, IEEE Computer Society Press, Los Alamitos, CA, pp. 271-278. 2002
[17] T. J. Biggerstaff, B. Mitbander, D. Webster, " The concept assignment problem in program understanding", 15th *International conference on Software Engineering*, Baltimore, Maryland, IEEE Computer Society Press Los Alamitos, California, USA, 1993
[18] P. Devanbu, R. J. Brachman, P. G. Selfridge, B.W. Ballard, " LaSSIE: A Knowledge-Based Software Information System", *Communications of the ACM*, Vol. 34, No. 5, pp. 35-49, May 1991
[19] L. M. Wills, "Automated Program Recognition by Graph Parsing", PhD Thesis, AI Lab, Massachusetts Institute of Technology, July 1992.
[20] V. Karakostas, "Intelligent Search and Acquisition of Business Knowledge from Programs", *Software Maintenance: Research and Practice*, Vol 4, pp. 1-17, 1992.
[21] N. E. Gold. "Hypothesis-based concept assignment to support software maintenance", *IEEE international Conference on Software Maintenance* (ICSM'01), Florence, Italy, pp 545-548, 2001.
[22] Allen Tucker and Robert Noonan, "Programming Languages Principles and Paradigm", Tata McGraw-Hill Publishing Company Limited, New Delhi, 2002.
[23] Kenneth. C. Louden, "Programming Languages Principles and Practice", Thomson Asia Pte Ltd, Singapore, 2003.
[24] Keith B. Gallagher and James R. Lyle, "Using Program Slicing in Software Maintenance", *IEEE Transactions of Software Engineering*, Volume 17, Issue 8, 1991, pp 751-76.

Prof. Shivanand M. Handigund received his Ph.D. degree from the Indian Institute of Technology Bombay in 2001. Currently, he is working as a full Professor and Head Super Computer Division, Department of Computer Science and Engineering at Bangalore Institute of Technology, Bangalore. His research interests include Software Engineering, Reverse Engineering, Database Management Systems, Object Technology and Computer Graphics. He is teaching several courses of Academia and Industry for last thirty five years and has published number of papers in national and international journals/conferences. He has delivered number of keynote addresses/technical lectures at various colleges and platforms. He has organized number of conferences and involved as reviewer for IEEE conference technical papers.

Ajeet A. Chikkamannur received his M. Tech. degree in Computer Science and Engineering in 2001 from the Visvesvaraya Technological University, India. Currently pursuing the Ph.D. and the research is focused on Design of Fourth Generation Languages. His research interests are Object Oriented System Development, Database Management Systems, System Simulation and Modeling. Presently working as Professor, Department of Computer Science and Engineering and teaching for graduate courses for last twenty one years.

K. Ananthapadmanabha received his M. Tech. degree in Computer Science and Engineering in 2002 from the Visvesvaraya Technological University. Belgaum, India. His research interests are Database Management Systems, Data Warehousing and Mining and Software Engineering. Presently working as a Professor in the Department of Information Science and Engineering. He is teaching for graduate, post graduate courses for last Eighteen years.

H. R. Shashidhara received his M. Tech. degree in Computer Science and Engineering in 2003 from the Visvesvaraya Technological University. Belgaum, India. His research interests are Data Warehouse and Data mining, Software Engineering. Presently working as Professor in the Department of Computer Science and Engineering. He is teaching for graduate and post Graduate Courses for last Eighteen years.