International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-9 Issue-3, February 2020

Edu-APCCM: Automatic Programming Code Constructs Mining from Learning Content

Maitri Jhaveri, Jyoti Pareek

Abstract: *The current education ecosystem is moving towards centralized online blended learning. Online learning repositories have replaced traditional libraries. Learning repositories contain learning materials, which can be located with the help of associated metadata. Associating metadata to the content (definition, program, example, figure, and table) of individual learning concept (topic) from the learning material also leads to a better search. If a student knows the prerequisites of the topic s/he wants to learn then the study of current topic would be more fruitful. The prerequisites of a computer science topic can be obtained from its explanation and the programming code snippet used for its implementation. This paper proposes a metadata "code construct as a prerequisite of a code snippet". For example "recursion and function call are prerequisite to understand recursive module of binary tree traversal". It also proposes the framework to automatically identify, extract and present the code constructs used in code snippets included in a computer science learning material. Thus obtained list of code constructs act as prerequisites for understanding the corresponding code snippet. Rule-based pattern mining approach is used for the identification of code snippet in the learning material and identification of code constructs in the code snippet. A pattern set is designed for the same. Natural language tool kit of python is used to identify the code snippet. The algorithms are tested on the programs of C, C++ and Java. Accuracy and efficiency of the developed algorithms is checked against the manual results given by subject experts. An average F1 score of 92% is obtained.*

Keywords: prerequisites, rule based mining, code constructs, learning material, text extraction and analysis

I. INTRODUCTION

A. Importance of Automatic Identification and Extraction of Programming Code Constructs

If a student wants to learn a concept, it is important for him/her to get knowledge of its prerequisites. For example, if a student wants to implement a logic of 'traversing a binary tree' the he/she must be aware of recursion, stack, function definition, function call, array declaration, loop structure, if-then-else structure. This can also help in creating the prerequisite path for the student. For example, the prerequisite map of 'recursion' can be "control structure -> loop structure -> defining and accessing an array -> stack -> recursion". A tool is required which automatically generates the prerequisite map. This paper proposes a pattern based text mining approach for extracting the prerequisites from the learning content available in university repositories. The prerequisite path has multiple nodes with neighbors of each node as its prerequisite (left node) and subsequent (right node). For example, consider a topic "pre-order traversal" of "data structures". In order

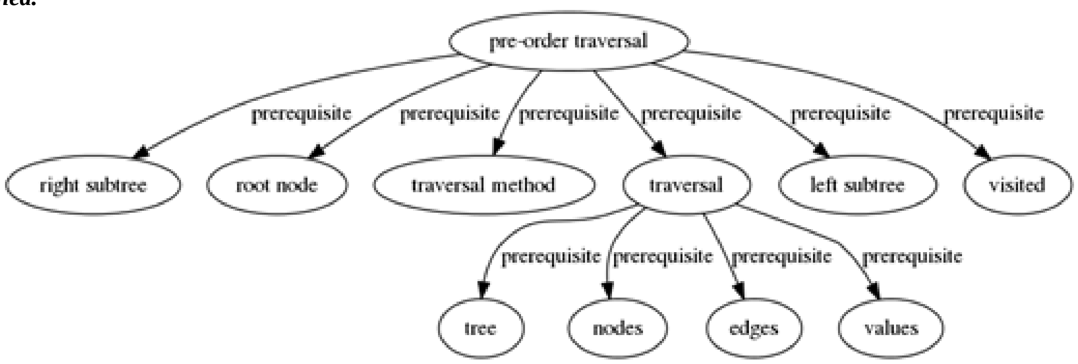

Fig.1. Prerequisites of "pre-order traversal" [7]

Table I. Sample set of code constructs used in programming languages

Function call	Control structure	Looping structure	Recursion
Inline function	Friend Function	Polymorphism	Operator overloading
Function overloading	GOTO statements	Comments	Array
Structure	Union	Class	Inheritance
Linked List	Switch-case	Parameter passing (By Reference/value)	Function definition
Include/define directives	Pointers		

to identify its prerequisites one should look at places where it is been defined, explained and implemented in a specific programming language. Automatic extraction of prerequisites from the textual content of a learning material is already done by the authors [7] as displayed in figure 1.

Revised Manuscript Received on February 05, 2020.
* Correspondence Author
Dr. Maitri Jhaveri, Lecturer, Department of Computer Science, Gujarat University, India.
Dr. Jyoti Pareek, Professor, Department of Computer Science, Gujarat University, India.

In this paper the authors propose the work done by them in automatic extraction of prerequisites from the code snippet which implements the given topic. In computer science domain, programming language code constructs also serve as the programming perquisites for the specified topic. For example, it is important to know beforehand the concept of 'recursion', 'functions call', and 'if/else' constructs to understand the code of "traversing a tree using recursion". Edu-APCCM automatically extracts the code constructs required to understand the related code snippet. Table 1 shows the set of programming constructs that are present in C, C++ and Java programming languages.

II. LITERATURE REVIEW

An approach to identify prerequisites is proposed [12] based on the concept that if concept1 occurs in the definition of concept2 then concept1 is the prerequisite of concept2. The sequence of learning goes from lower learning level to upper learning level. The learning level is calculated on basis of three features. They are range in which the topic is covered, number of incoming links from other learning materials and number of outgoing links to other learning materials of the repository. A personalized assessment model is also proposed that is based on the principle of on identification of learning gaps of the students. They intend to reduce these learning gaps as much as possible. The learning gaps are identified by constructing a hierarchical prerequisite map. A tool is proposed [9] to decide the sequence in which documents should be read, with documents having basic (general) concept to be read first followed by the documents covering the advanced (specific) concepts. The documents are multiple Wikipedia pages. Collection of documents is organized in the form of a tree. This aids a learner to choose a reading sequence of the documents. For identifying the prerequisite relations among concepts, a reference distance (RefD) [10] is proposed. It is a link-based metric that measures the relationship among learning concepts. The relations are classified as asymmetry and irreflexivity. Its contribution is a set of 1336 concept pairs in computer science domain and mathematics domain. Statistical methods and machine learning techniques are exploited [2] to identify prerequisite concepts and defined concepts from learning resources available on the web. Formatting features are used as a principal technique for identification of prerequisites and definitions. A model for concept map construction from textbooks is presented [13]. It combines the knowledge from Wikipedia and the way corresponding textbook is structured. The knowledge is gained from the index of the textbooks, where prerequisites appear earlier in the index. A three-level prerequisite path construction approach is proposed [8], where each concept is given weightage. The weights are based on the frequency of learning concept in the learning material. Prerequisite pairs are hence identified. The corresponding map is acyclic. Norm reference technique is used to differentiate between relevant and irrelevant items. Applications of competence-based knowledge space theory in web based online learning are discussed in [11]. Their approach focuses on utilization of prerequisite path for achieving personalization and adaptivity in distance education and web-based online learning. Their work is based on the usage of concept maps and semantic networks for deriving prerequisite concepts. Bayesian network is used to establish prerequisite relations from multiple learning materials. It is implemented by developing a component-based MEDEA architecture [1]. Difficulty of each knowledge unit is also established. A Clique Grow model [6] is proposed to establish all those prerequisite relations which do not follow and are not captured by a specific pattern. These relations are identified based on the transitivity among prerequisite concepts identified by a specific pattern. Versus query logs are used to establish the graph, where nodes correspond to those learning concepts whose attributes can be compared. Machine learning techniques are exploited for annotating (prerequisite or outcome) learning concepts present in the learning material [3]. The annotation is contextual. Methods are proposed to access and score student's performance. These annotations of learning resources such as web documents lead to a sequenced learning path for study. A prerequisite path is created [4] using machine learning techniques with various learning objects as its nodes. A rule-based approach is used for identification of prerequisites and defined outcomes from a learning material are proposed [5]. This approach requires the knowledge and use of subject domain. The domain is represented in form of ontology.

Existing methods identify prerequisites of a concept from wrongly answered queries, concept definition statement and statements embedding the concept. If the entire article belongs to the concept then all those keywords which are not defined in the article are considered prerequisites.

Automatic extraction of prerequisites from the analysis of code snippet is not yet explored. A survey of students (MCA, M.Sc and M.Tech from Gujarat University) was conducted and it was found that if students knew the code constructs present in a code then they could understand the code easily. The proposed model first extracts all the statements of code snippet pertaining to a given learning concept and then analyses them to classify in any one of the programming code constructs.

III. PROPOSED WORK

A. Automatic Identification of Code Constructs

Figure 2 shows the framework for identification of code constructs. The proposed algorithm is implemented using python natural language processing toolkit. DOCX parser, PdfMiner and BeuatifulSoup parsers are used to parse DOC files, pdf files and HTML pages. Raw text, set of images and set of tables are obtained. Output screens are developed using JavaScript.

A.1 Code Snippet Identification

The learning concept to which the code belongs can be found by its presence in the caption of code snippet or 2 lines above and 2 lines below of each code snippet. The presence of a code snippet is identified by continuous presence of language specific words. Each sentence from the code is analyzed for the presence of code construct.

Pattern matching engine looks for the existence of a specific pattern in each sentence. A data set is prepared for the patterns used to identify code constructs. It is prepared by gathering the patterns after an extensive study of programs written in C, C++ and Java languages. Using sentence tokenizer and word tokenizer each word is separated and tested against the patterns of table II and table III.

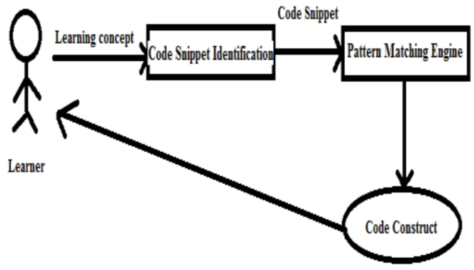

Fig.2. Proposed Framework of Edu-APCCM

A.2 Pattern Matching Engine

A.2.1 *Algorithmic steps taken to identify various code constructs*

Table 2 lists different Code constructs to be identified and corresponding rules for identifying their occurrences. Column 1 list the code constructs to be identified. Column 2 lists corresponding set of core steps to be executed for automatic extraction.

Table II. Steps taken for identification of code constructs

Code Construct	Core Algorithmic Steps
Function call	• Check each code statement • Check whether each word is any one word from return_types, • Occurrence of word after all return_type words followed by '(' or '<operator>('is the function name
Recursion	• extract function name with the help of succeeding '('parenthesis • check each statement in code segment • before occurrence of '}' if same function name appears then it shows presence of recursion
Function overloading	Three ways of overloading 1. number of parameters 2. type of parameters 3. sequence of parameters But in all cases name of function remains the same Split by function name Identify two function names with same name
Parameter passing by references	Split the code statement by function name After '(' if the word is from return_type followed by '&' without space
Parameters passing by value	Split the code statement by function name After '(' if the word is from return_type followed by space
Looping structure	Check each code statement Presence of any looping_structure_keyword
Friend function	Ocuurence of 'friend' keyword followed by occurrence of a function 'friend' keyword Return_type '('
Go to	Presence of 'goto' statement
Comments in Python, Java, C, C++	Presence of // /*..*/ # but not '#include' or '#define'
Class	Presence of 'class' keyword
Array	Presence of any word from Return_types Followed by '['
Switch-case	Presence of 'switch' and 'case'
Struct	Presence of 'struct' keyword
Union	Presence of 'union' keyword
Operator overloading	Check each code statement If two statement with same function names followed by same operator
Inline function	If the code statement contains '#define' in C or 'lambda' in python
Polymorphism	If the code statement contains 'extends'
Pointer	If the code statement starts with 'struct' and followed by a word ending with '*'.

A.2.2 *Supportive patterns used for automatic identification of code constructs*

Table 3 list the supportive patterns used to identify the occurrences of return types, looping structures, operators and special words. These patterns help in executing steps to identify code constructs.

Table III. Supportive patterns

Return_types	'int', 'integer', 'float', 'char', 'void', 'double', 'static', 'boolean', 'decimal', 'string', 'point'
Loop structure keywords	'if', 'else', 'for', 'while', 'do'
Operators	'\+', '\-', '*', '\/', '\='
Special_words	'struct', 'void', 'bool', 'char', 'return', 'push', 'namespace'

IV. RESULTS AND ANALYSIS

This section includes sample input (table IV) to the algorithm, results (Figure 3) obtained for the sample input and results (table V) for 7 other programs written in JAVA, C and C++. Table VI presents the tabular result analysis.

A. An Example of 'Java' program (Sample Input to the proposed algorithm)

Table IV shows a sample input which is a code snippet covering various programming concepts of a java program. Figure 3 gives the output screenshot containing the list of automatically extracted code constructs obtained as a result. Table V shows the experimental results obtained on 7 programs.

Table IV. A sample java code snippet incorporating different programming constructs.

```
import java.util.*;
class One  {
public void display(){
  System.out.println("One");}}
class Two extends One {
@Override
public void display() {
  System.out.println("Two");
}
public int add(int x, int y)  //Parameter Passing{
  return x+y;
}
//Overload
public double add(double x,double y) {
  return x+y;
}}
abstract class TwoWheeler {
public abstract void run();
}
class Honda extends TwoWheeler{......}
public class MainClass
static int factorial(int n){
          if (n == 1)
            return 1;
          else
            return(n * factorial(n-1));
    }
public static void main(String[] args) {
  One a=new One();
  a.display();
  Two b=new Two();
  b.display();
  System.out.println(b.add(4,2));
  System.out.println(b.add(5.,2.)); //polymorphism
  TwoWheeler test = new Honda();
  test.run();          //function call
  int x = 10;
  while( x < 20 ) {
      System.out.print("value of x : " + x );
       x++;
      System.out.print("\n");
  }
 if(x > 18)
      System.out.println("above 18 ");
Else

 System.out.println("below 18 ");  System.out.println("Factorial of 5 is: "+factorial(5));
int week = 0;
Scanner sc = new Scanner(System.in);
week = sc.nextInt();
      String day;
switch (week) {
         case 1:
           day = "Sunday";
           break;
         case 2:……..
}}
```

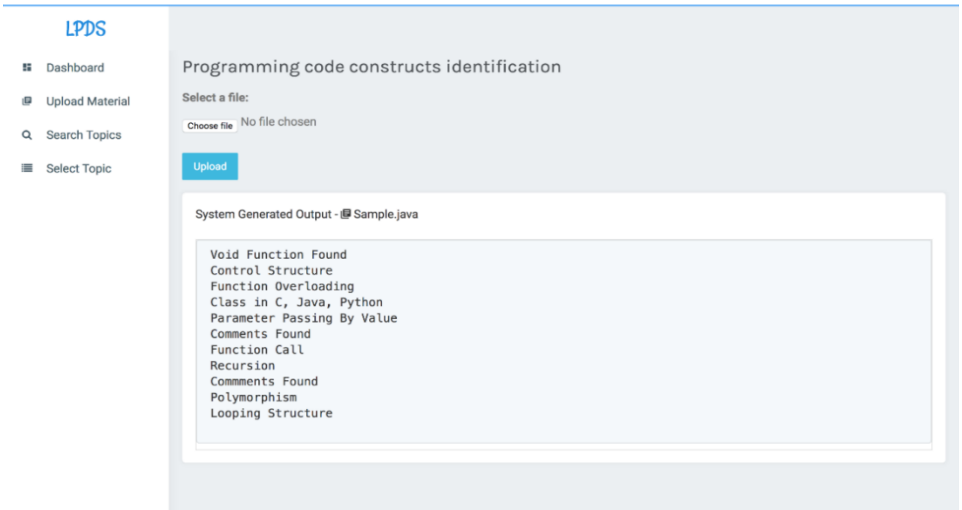

Fig.3. Result output screen displaying automatically identified programming code constructs

Table V. Experimental Results of Proposed algorithms on 7 programs

Sr. No.	Program Description	Programming Language	No of lines in program	Results (number of automatically extracted code constructs)
1	Program to convert infix expression to postfix expression	C	57	11
2	Program to insert and element in Binary search tree	C++	52	15
3	Program to delete an element from Heap tree	Java	35	8
4	Program to implement depth first traversal of a directed graph	C	27	6
5	Program to sort a list using bubble sort	C	25	9
6	Program to implement student's result system	Java	135	12
7	Program to calculate salary of bank employees	C++	165	14

B. RESULT ANALYSIS

Table VI lists the evaluation of the results obtained for 7 programs. The results were given to the subject experts for verification of correctness, completeness and sufficiency. A satisfactory response was obtained. The experimentation and evaluation were then continued with 100 code snippets written in C/C++/Java. Verification against expert generated manual results showed an F1 score of 92%. F1 score measure was used for measuring the accuracy of automatically generated results. The F1 score is calculated as weighted average of the precision and recall. F1 score attains its best value at 1 and worst at 0.

F1=2 * recall* precision / (recall + precision)

Precision = Total number of correct results identified by the tool /Total number of all results identified by the tool

Recall = Total number of correct results identified by the tool/Total number of results provided by experts

Figure 4 presents the graphical analysis of the results obtained. It plots the precision and recall of the results obtained. X-axis represents the program number out of 100 programs (test data) and Y-axis represents the range (0-1) of precision and recall. Precision is found to be higher that the recall. That means that, the tool is extracting some incorrect code constructs along with the correct code constructs. The incorrect results were found because the tool extracted code constructs from multiline comments as well. Work is under progress to improve the results.

Table VI. Tabular Analysis of Automatically extracted code constructs against expert generated manual results

	Program	Number of programming code constructs as suggested by the subject experts	Number of automatically extracted code constructs by the proposed algorithm	Number of correctly extracted code construct as per expert's opinion	Precision	Recall
1	Program to convert infix expression to postfix expression	11	10	10	1	0.9091
2	Program to insert and element in Binary search tree	15	17	15	0.882	1
3	Program to delete an element from Heap tree	8	7	7	1	0.875
4	Program to implement depth first traversal of a directed graph	6	6	6	1	1
5	Program to sort a list using bubble sort	9	9	9	1	1
6	Program to implement student's result system	12	11	10	0.9091	0.8333
7	Program to calculate salary of bank employees	14	13	12	0.9231	0.8571
	Best Case				1	1

	Worst Case				0.8823	0.8333
	Average Case				0.9592	0.9249
	Standard Deviation				0.0523	0.0737

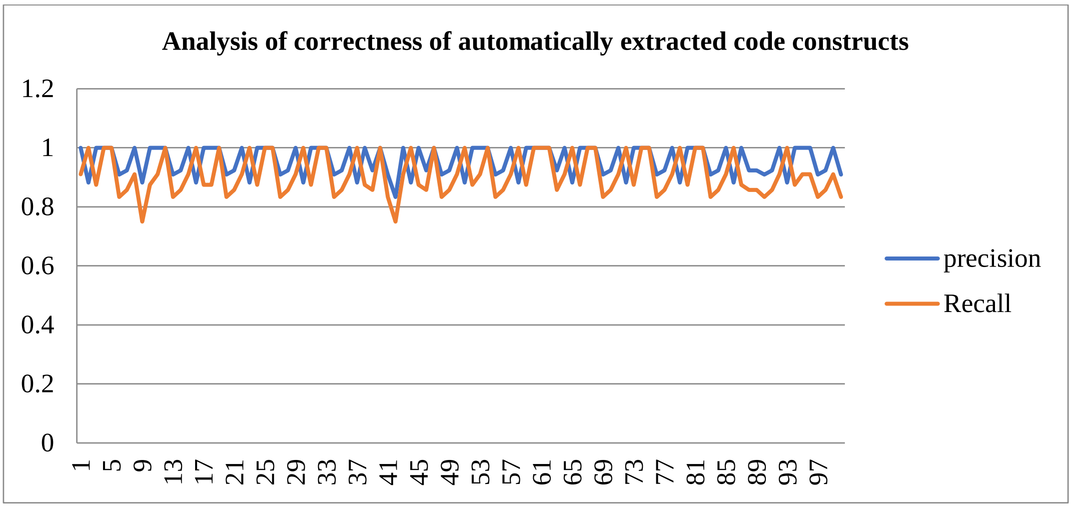

Fig.4. Graphical Analysis of Automatically extracted code constructs against expert generated manual results [X-Axis is program number; Y-Axis is range (0-1)]

V. CONCLUSION

This paper introduces the concept of programming code constructs as prerequisites for the code snippet pertaining to the topic specified by the learner. It discusses the methods developed
for automatic extraction of code constructs from the code snippets of programs written in C, C++ and Java languages. Code constructs in a code snippet can be found in different forms. The proposed algorithm tries to cover as many forms as possible. A pattern set developed for their identification is presented with corresponding rule set for automatic identification, extraction and manifestation. A sample code with corresponding auto generated output is also presented to show working of the proposed approach. Satisfactory evaluation against expert generated manual results is shown. Work is under progress to improve the F1 score. The authors are also working on inclusion of as many programming languages as possible. The task of identifying code snippet from an image is also in progress.

REFERENCES

1. Carmona, C., Millán, E., Pérez-de-la-Cruz, J.L., Trella, M. and Conejo, R. "Introducing Prerequisite Relations in a Multi-layered Bayesian Student Model" published in *User Modeling 2005, Lecture Notes in Computer Science*, Vol 3538. Springer, Berlin, Heidelberg, pp. 347-356, 2005.
2. Changuel, S. and Labroche, N. "Distinguishing defined concepts from prerequisite concepts in learning resources" Proceedings of the *IEEE Symposium on Computational Intelligence and Data Mining*, Paris, France, pp. 22-29, 2011.
3. Changuel, S., Labroche, N. and Bernadette, B. "Resources Sequencing Using Automatic Prerequisite–Outcome Annotation". *ACM Transactions on Intelligent Systems and Technology*, vol. 6 No.1, pp.6:1-6:30, 2015.
4. De Medio, C., Gasparetti, F., Limongelli, C., Sciarrone, F. and Temperini, M. (2016) "Mining Prerequisite Relationships Among Learning Objects" Proceedings of the *International Conference on Human-Computer Interaction- Posters' Extended Abstracts*, pp.221-225, 2016.
5. Jain S., Pareek J "Automatic Extraction of prerequisites and learning outcomes from learning material", *International Journal of Metadata, Semantics and Ontologies*, Vol 8, Issue 2, pp. 145-154, 2013.
6. Jang, M., Park, J. and Hwang, S. "Predictive mining of comparable entities from the web" Proceedings of the *Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, Ontario, Canada*, pp. 66-72, 2012 .
7. Jhaveri, M. M., & Pareek, J. "Edu-ACoCM: Automatic Co-existing Concept Mining from Educational Content". *International Journal of Technology-Enabled Student Support Services (IJTESSS), 9(1)*, pp. 16-40, 2019.
8. Kavitha, R., Vijaya, A. and Saraswathi, D. "An augmented prerequisite concept relation map design to improve adaptivity in e-learning" Proceedings of the *International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012), Salem, Tamilnadu, India*, pp. 8-13, 2012.
9. Koutrika, G., Liu, L. and Simske,S. "Generating reading orders over document collections" Proceedings of the *IEEE 31st International Conference on Data Engineering*, pp. 507-518, 2015.
10. Liang, C., Wu, Z., Huang, W. and Giles, C. L. "Measuring prerequisite relations among concepts" *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1668-1674, 2015.
11. Steiner C.M. and Albert D. "Personalising Learning through Prerequisite Structures Derived from Concept Maps", *Advances in Web Based Learning, Lecture Notes in Computer Science*, Vol. 4823. Springer, Berlin, Heidelberg, pp. 43-54, 2007.
12. Wang, S. and Liu, L. "Prerequisite Concept Maps Extraction for Automatic assessment" Proceedings of the *25th International Conference Companion on World Wide Web*, pp. 519-521, 2016.
13. Wang, S., Ororbia II, A.G., Wu, Z., Williams, K., Liang ,C., Pursel, B. and Giles, C. L " Using Prerequisites to Extract Concept Maps from Textbooks" Proceedings of the *25th ACM International Conference on Information and Knowledge Management*, pp.317-326, 2016.

AUTHORS PROFILE

Dr. Maitri Jhaveri, is Ph. D. in Computer Science. She is working as Lecturer in Computer Science at Department of Computer Science, Gujarat University. She has 16 years of teaching experience and 8 years of research experience.

She has to her credit several published research papers. Her areas of interest are Natural Language Processing, Information Retrieval, Machine Learning, E-learning, and Object Oriented Paradigms.

Dr. Jyoti Pareek, is Ph. D. in Computer Science. She is working as Professor in Computer Science at Department of Computer Science, Gujarat University. She has 28 years of research and teaching experience. She has to her credit a Book and several published research papers. She has been the reviewer of the research papers and member of technical program committee at many International Conferences. Her area of interest is Natural Language Processing, Machine Translation, Information Retrieval, Machine Learning, E-learning, and Object Oriented Paradigms. She is senior Member of IEEE and Life member of Computer Society of India.