Retrieval of Java Program Code Components using Case Based Reasoning (CBR)

Mahadev K. Patil, Pallavi P. Jamsandekar, Shabnam S. Mahat

Abstract: Object Oriented Programming (OOP) facilitates to create libraries of reusable software components. The reusability approach in developing a new system can be applied to an existing system with prior modifications. The reusability definitely decreases the time and effort required for developing the new system. To support reusability of program code, a proper code retrieval process is necessary. It makes possible to search the similar code component of java programming environment. OOP paradigm has specific style of writing the program code. The program code is a collection of objects, classes and methods. It is very easy to store the cases and reuse or revise wherever necessary. To get the similarity between the program code components, it is necessary to have an efficient retrieval method. The retrieval phase can retrieve the program code components as classes, methods, and interfaces depending on components selection by the user. A purely case-based approach is adopted for revising or reusing the existing cases to solve the new problems. Case Based Reasoning (CBR) is the process of solving new problems based on the experience coming from similar past problems.

Keywords: reusability, retrieval, similarity measure, CBR

I. INTRODUCTION

Object Oriented Programming language facilitates to create libraries of reusable software components. One of the challenges in software industries are the increase in demand of software development in different functional areas. [1]

The technology advancements evolved in functionalities will increase the complexity in reuse. The reusability approach was proposed by software engineering discipline. The idea behind this is to develop a new system by adopting the existing one with prior modifications. The reusability definitely decreases the time and effort required for developing the new system. To support reusability of program code, a code retrieval process that makes it possible to search the similar code component of java programming environment. It can be based on code component retrieval and its reusability. It can retrieve the java classes, methods or interfaces based on user requirement. It reduces the time and efforts required in the development of the new system.

The retrieval is one of the phases of CBR cycle. It is very important to retrieve the most similar cases from the stored cases in problem solving. OOP paradigm has specific style of writing the program code.

Revised Manuscript Received on February 05, 2020.
* Correspondence Author

Dr. Mahadev K. Patil*, Department of Computer Applications, Bharati Vidyapeeth (Deemed to be) University, Pune

Prof. Dr. Pallavi P. Jamsandekar Professor, Bharati Vidyapeeth (Deemed to be) University, Pune

Dr. Shabnam S. Mahat, Department of Computer Applications, Bharati Vidyapeeth (Deemed to be) University, Pune

The program code is a collection of objects, classes and methods. It is very easy to store the cases and reuse or revise wherever necessary. To get the similarity between the program code segments, it is necessary to have an efficient retrieval method. A purely case based approach is adopted for revising or reusing the existing cases to solve the new problems. [1] The case base is dynamic in nature which causes slow processing in a retrieval of cases where users are continuously adding the new cases in case base.

II. CASE BASED REASONING (CBR)

CBR can be described as the process of solving new problems based on the experience coming from similar past problems. In general, CBR cycle [2] can be described by the following:

Retrieve When a new problem arrives the most similar cases are *retrieved.*

Retrieve is the process of remembering a relevant experience or set of experiences

Reuse their solutions *reused* to provide a proposed solution

Revise a proposed solution which may be *revised* after testing to create a final solution

Retain As a final stage the new problem and solution can be *retained* as a new case in the case base, allowing the system to learn new knowledge

Case Base stores previously solved problems with their solutions

Case records several features and their specific values occurred in that situation

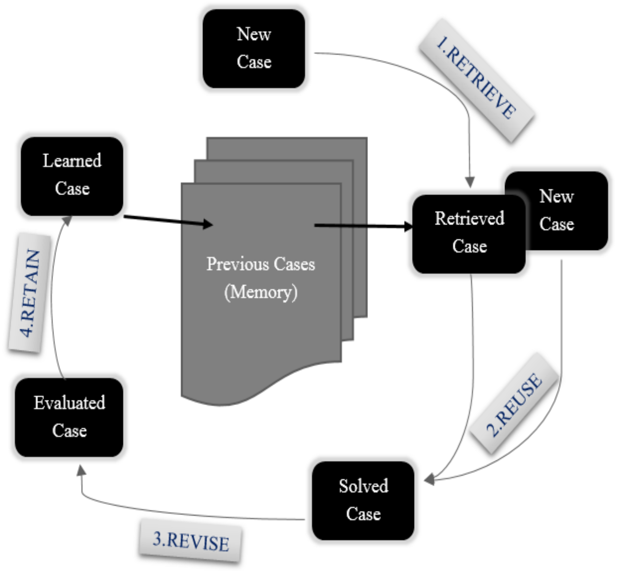


Figure (a): CBR Cycle [2]

To have an efficient retrieval, CBR plays vital role by reusing the similar past experiences of problem solving. OOP paradigm has some style of problem solving which is generalized to store as case and reuse wherever applicable. To retrieve the most appropriate experience (stored case) one need to have efficient retrieval method. A purely case-based reasoning approach is adopted for OOP class library reuse.

III. HASHING TECHNIQUE

The programmer can use case indexing approach for case storage and case retrieval. The hash indexing technique can be applied for faster retrieval. The hashing indexing searches a case by determining the index value using a key value without scanning entire case base. It took lesser retrieval time, small storage and easier to implement. The case retrieval is the process of finding possible cases from case base that is very similar to the inputted case. For a given new case C' = {x1, x2, x3…xn, θ}, where θ is the decision to be determined. The case retrieval is the process of finding old cases C those are close to C'. The mapping of C and C' is represented as (C, C') where cases C = {C1, C2, C3…Cm}, and C' is a query case. The similarity between both cases (C' € C) will be determined based on the similarity, Sim (C', C). [3]

The **case indexing technique** played a very important role to control over the searching process. It scans entire case base or portion of it. It improves quality of retrieval by applying the concept of buckets – group of records as a cluster. The programmer can design a different bucket for program code components classes, methods or interfaces. To overcome the problems in sequencing searching, hashing indexing method with search key improves the searching process and retrieval speed with growing cases in the database.

IV. RESEARCH METHODOLOGY

The authors followed ***Design and Creation Research*** strategy. It focuses on developing new IT product, also called artifacts. The IT artifacts include construct, model, method, instantiation.
The work is divided into four steps:

1. Learning via creation process is dealing with case structure design.
2. System development methodology covers case retrieval algorithm implementation aspect.
3. The data generation technique is used to gather the outcome of students while executing the program code.
4. The evaluation viz, the different test cases inputted to the program code and find out the results, the entire execution of program code is evaluated by the student who are currently learning java programs.

The above four steps are explained in following section (V) working model. Its sub section explains the case structure design, system development – program code execution, and the evaluation, results/outcome by operating the entire program code execution through different students. The working model is designed based on above four steps including CBR's four phases.

V. WORKING MODEL

The working model focuses on CBR cycle. The model is suitable for retrieval of cases in java program code. The theory behind the phases of CBR has been explained.

A. User Input (Source Code)

The user can input or upload a new case to the system. It is a simple <java> file. A user can upload the <java> file to the system. It can be stored in a case folder for further reuse.

With the help of **java reflection method**, the uploaded <java> file can be extracted into the different program code components such as class, package, interface or methods etc. Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine.

The **data preprocessing technique** can be applied on case base for the correctness of user case. This is one of the steps used in data uploading to a system. This will check whether a user has correctly uploaded the <java> file or not. If a user uploads a wrong program file to a system, it will generate an error message to the user. Its not mandatory that use need to upload a program file which is compiled one. The user can upload any java program file. The system can provide automatic compilation facility. The duplicate files can be controlled automatically by the system.

B. Case Base

The case base is the case repository. The reflection technique is applied in each case. It extracts the components of the uploaded java file. The extracted components viz, class, methods, interfaces are stored in the database. It's a better option to apply hash indexing to technique. The extracted components are treated as experiences and maintained as cases. The cases are organized to facilitate the search operations. The refracted components of program code are stored in the case base. All the attributes should be maintained properly with its corresponding values. The programmer may use any database packages for storing cases. It may have filename, classes, packages, interfaces and the relevant fields. The most probable java program code components.

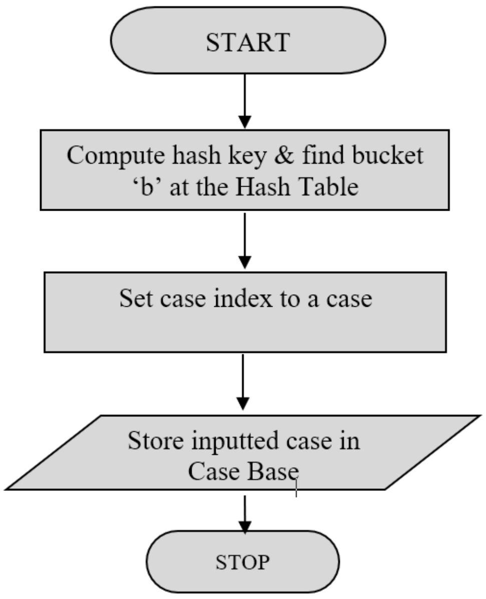
The program code or its components are defined as case or case base.

Files (fileID, filename)
Package (packageID, packageName, importedPackage, codeContent, fileID)
Classes (classID, className, inheritedClass, inheritedFrom, implementedFrom, implementedTo, codeContent, fileID)
Interface (interfaceID, interfaceName, inheritedClass, codeContent, fileID)
Methods (methodID, methodName, argumentType, argumentName, returnValue, codeContent, fileID)

1. Hash Key Computation in Case Store

The hash key computation before storing cases in the case base is represented in the flowchart (a).

The basic steps are a computation of hash key based on the cases and find possible bucket available, set the case index to a case and then store the case into case base.


START
Compute hash key & find bucket 'b' at the Hash Table
Set case index to a case
Store inputted case in Case Base
STOP

Flowchart (a): Hash Key Computation in Case Store

To retrieve case from the case base it is common to use hashing indexing. This technique has been developed to access large files residing on external storage, not just for accessing fixed size files but files those grow over their time.

The programmer can apply hash indexing technique which may have following steps:

i. It will check the size of existing case or case base.
ii. The user can input the case to the system.
iii. Then, it will calculate search key from the available buckets (a term used in indexing concept).
iv. As per the cases available in the said bucket, the appropriate counter take place and stores further cases using indexing technique.

C. Similarity Search

This is a part of searching. It has a responsibility for identifying the similar code exists between the case base. It can be designed to sort the cases in rank wise of their similarity checking percentages. The highest similarity showed first, then second and so on.

The similarity between two cases is computed by the formula:

Similarity (T, S) = $\sum$ f (T_i, S_i) x Wi for i=1 to n

where,

T= user inputted case

S= case from existing case base

n= number of attributes in each case

i= individual attribute from 1 to n

f = similarity count for the selected attributes from the cases T, S

W= relevance of attribute in similarity check

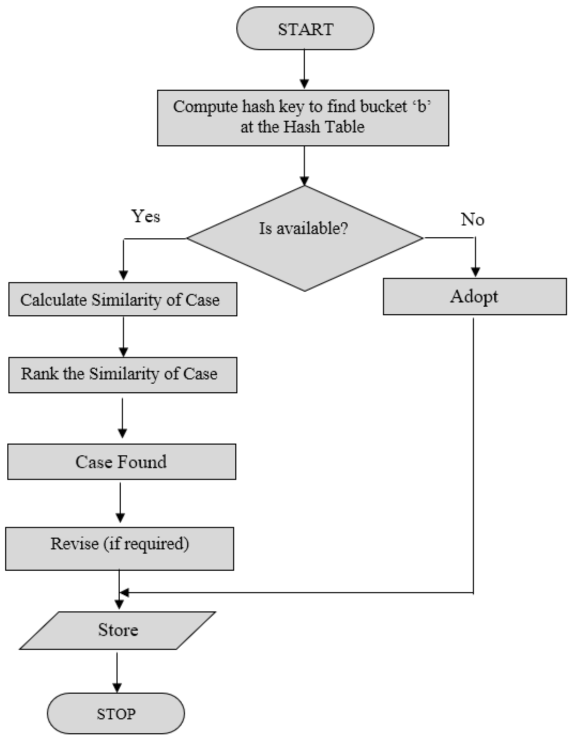
The similarity search can be represented as:

i. It begins with accepting user case.
ii. It checks existing case base for similarity with user case.
iii. If the corresponding similarity has been found, then it updates case base.
iv. The updated case can be re stored in the case base.

D. Case Retrieval

It is a step where a programmer can use similarity checking process as well as hashing. It is responsible for retrieving the cases as per the case inputted by the user. This retrieval phase can retrieve the program code components as classes, methods, and interfaces depending on components selection by the user.

The computation of hash key in a retrieval of cases is shown in flowchart (b).


START
Compute hash key to find bucket 'b' at the Hash Table
Is available?
Yes
No
Calculate Similarity of Case
Rank the Similarity of Case
Case Found
Revise (if required)
Adopt
Store
STOP

Flowchart (b): Hash Key Computation in Retrieval Process

It may be defined by considering following aspects:

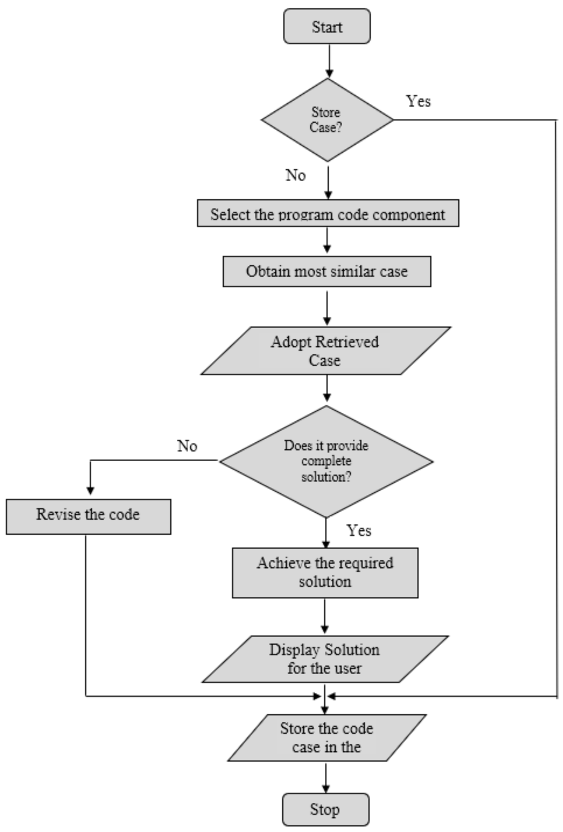
i. Compute hash key to find buckets of any size at the hash table
ii. Check the availability of any cases in the said bucket
iii. If Yes, then obtain its similarity and arrange accordingly otherwise use regular search technique – linear or binary search technique
iv. Store the resulting case in the database

The overall retrieval process is depicted in the following flowchart (c).

i. It starts with adding new cases to case base. The case base holds the different cases in a particular format.
ii. The similarity function will retrieve the most promising cases from the case base.

iii. The retrieved case can be used for further retrievals. This case can be a new one to the existing case base. The programmer can provide a facility to add/modify the content of the code base.

iv. The detailed solution is displayed to the user.


Start
Store Case?
Yes
No
Select the program code component
Obtain most similar case
Adopt Retrieved Case
Does it provide complete solution?
No
Revise the code
Yes
Achieve the required solution
Display Solution for the user
Store the code case in the
Stop

Flowchart (c): Steps in retrieval

For the faster retrieval of the program code from the case base, hash indexing can be adopted. In this case, the hashing creates buckets for storing program code components or code. Based on it, hash key can be computed and used as one of the functionalities in similarity checking process.

D. Adaption/Reuse

This area focuses on reusing the retrieved program code component/code base. The entire case can be re compiled by JVM functionality of Java. Its program code components can be re stored by prior data preprocessing technique. A count can be used for checking purpose that, how many cases can be reused by the user.

E. Revise

Sometimes, it may be a situation where we need to revise the retrieved code / code component. The revise function can revise the entire code by adding or modifying the existing program code. So, the user can revise the retrieved code as per the requirement. Then it can be stored return to the existing code base.

F. Retain

This function can be used to store the retrieved code/code component as retrieved by the retrieval function. The user can directly store as it is in the existing code base.

VI. SAMPLE CODE

A. CaseDescriptor (String File) – IO Process

i. Input <*.java> file located in any drive

ii. Identify the possible modifier: get_modifier()

```
public static String get_modifier(int M1)
{
String str=null;
if(Modifier.isAbstract(M1)){ str="Abstract ";}
else if(Modifier.isFinal(M1)){str="Final ";}
else if(Modifier.isInterface(M1)){str="Interface ";}
else if(Modifier.isNative(M1)){str="Native ";}
else if(Modifier.isPrivate(M1)){str="Private ";}
else if(Modifier.isProtected(M1)){str="Protected ";}
else if(Modifier.isPublic(M1)){str="Public ";}
else if(Modifier.isStatic(M1)){str="Static ";}
else if(Modifier.isStrict(M1)){str="Strictfp ";}
else  if(Modifier.isSynchronized(M1)){str="Synchronized
";}
else if(Modifier.isTransient(M1)){str="Transient ";}
else if(Modifier.isVolatile(M1)){str="Volatile ";}
return str;
}
```

iii. Extract the file content into case descriptor: retrieval_Code()

```
Class c = c1.getClass();
Class sc = c.getSuperclass();
while (sc != null) {
String cn = sc.getName();
get_all_interface(c);
get_all_constructors(c);
get_class_annotation(c);
get_all_Methods(c);
get_all_Fields(c);
```

iv. Apply hash index on code components: GetHashData()

```
public byte[] GetHashData(byte[] bytes)
{ var code = CODE.Create();
byte[] hashData = code.ComputeHash(bytes);
foreach (byte b in hashData)
Console.WriteLine(b);
return hashData;}
```

v. Store in database

```
int l=0;
Connection con=db.setConnection(con1);
Statement stat=con.createStatement();
ResultSet rs=stat.executeQuery("Select * from
Programs ");
while(rs.next())
```

```
{jTable1.setValueAt(rs.getString(1), l, 0);
jTable1.setValueAt(rs.getString(2), l, 1);
jTable1.setValueAt(rs.getString(3), l, 2);
l=l+1;}
```

B. similarityMeasures(String argument list)

i. Input parameter
ii. Apply threshold on selection
iii. Set k value
iv. Apply k – nearest neighbor for searching
v. Import java classifiers: KNN()
vi. Read file using BufferedReader technique
vii. Select the appropriate case or use default selection.

```
BufferedReader datafile = readDataFile("FileName");
Instances data = new Instances(datafile);
data.setClassIndex(data.numAttributes() - 1);
Instance first = data.instance(0);
Instance second = data.instance(1);
data.delete(0);
data.delete(1);
Classifier ibk = new IBk();        //   imported   classifier

ibk.buildClassifier(data);
double class1 = ibk.classifyInstance(first);
double class2 = ibk.classifyInstance(second)
```

VII. RESULTS

The code is executed successfully in java programming environment. The execution environment is being operated in realistic environment. The code tested by different students who are learning java. The data collected randomly for **100 students of Under Graduate in Solapur City.**

The table (a) represents the case retrieval outcome by successful execution of the program code. It represents overall percentage of case components retrieved by the different students and the percentage of similar cases found in retrieval process. The authors do various experiments on different aspects of program code component's and to get the desired result based on the inputted case.

Components / Users		Students
Percentage of program code Components retrieved	Classes	15
	Methods	13.5
	Interfaces	3.5
	Both Classes and Methods	17.5
	Both Classes and Interfaces	2
	Both Methods and Interfaces	0
	All	48.5
Percentage of Similarities exist in retrieval process	Found Cases	76.5
	Revised Cases	28.5
	Reused Cases	15.5
	Not Found Cases	23.5

Table (a): Case retrieval outcome

From the table (a) it is clear that, in some cases around 23.5% cases are not retrieved to the students. This has happened due to unavailability of the similar cases which does not exist in the case base. It can be resolved by retaining the cases in the case base. About 15% cases, classes are retrieved by the students, 13.5% methods and 17.5% both methods and classes are retrieved based on program case inputted. The similarity between the classes are retrieved around 76.5%. As per the CBR's four R's are considered, 28.5% cases were revised and 15.5% cases were reused.

From the chart (a), it is clear that, most of the students were satisfied with overall retrieval. The executed code gave effective results in retrieving all the components.

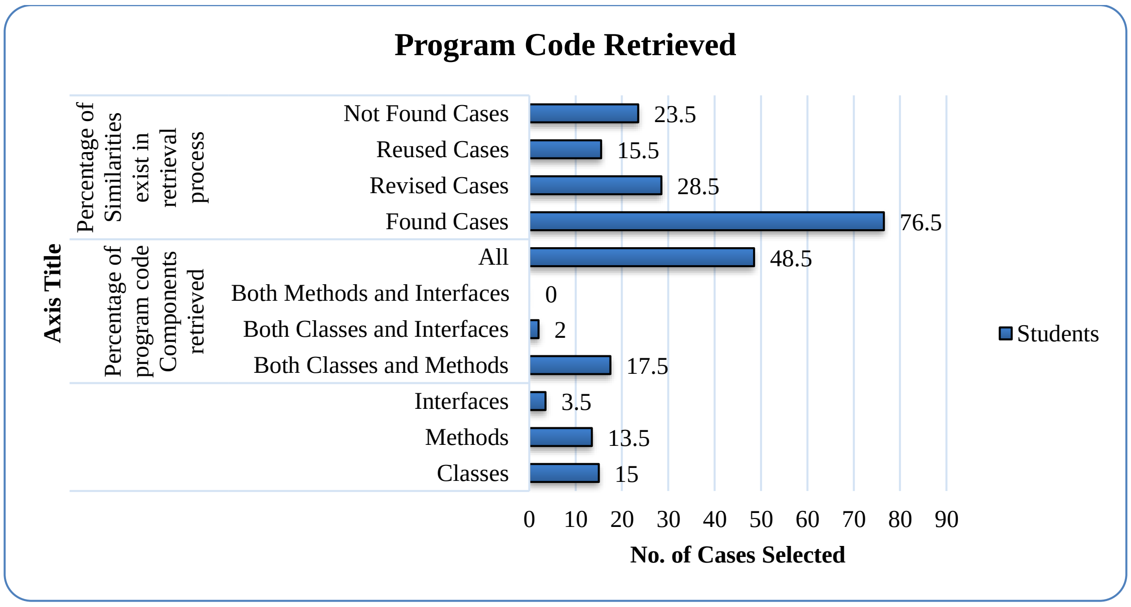


Chart (a): Program Code Retrieval

The performance of the system depends entirely upon the nature of the case library. If the case library has a large number of cases with representing diverse problems faced and their solutions the chances of a good diagnosis are good. The data loaded in the database are about 160 records (java program files) including all the components of the program.

VIII. APPLICABILITY

The mechanism is helpful for novice programmer who has an experience in working with an existing code repository. It turns further development towards the projects where effectively revised the available code repository. The researcher wants to make it easier for users, programmers and teachers to make use of these libraries. The major challenge in programming is to improve learning quality and productivity of programmer, teachers as well as students.

IX. CONCLUSION

The overall structure of the CBR 4R's has been explained. The working model has 4R modules. The module explained in detail the working of 4 R's of CBR. The functionalities of each R of CBR cycle explained. The retrieval process is represented in a model. The user has two modes to carry out the work. The retrieval process is depicted by combining case store mechanism and search similarity functionality. The first one, handles case base. The case base mechanism automatically performs the data preprocessing and hash indexing technique to store cases in the case base. The second part is user interaction in which user case can be handled in the retrieval. The user can edit the code and get the similarity. Based on retrieval, user has freedom either to
revise or retain as per the requirement. The modified hashing indexing involved has two main tasks one is to store the new case and retrieve the case.

REFERENCES

1. D. P. P. J. M. K. Patil, "Retrieval of Similarity Measures of Code Component," IRA-International Journal of Technology & Engineering, vol. 6, no. 3, pp. 38-43, 2017.
2. J. L. Kolodner, "An Introduction to Case-Based Reasoning," in Artificial Intelligence Review, Atlanta, GA, College of Computing, Georgia Institute of Technology, 1992, pp. 3--34.
3. S. I. Morisbak, The Road to ASCRARAD: The Development of Agent Support for a Case-based Reuse Application for RAD, June 22, 2000.

AUTHORS PROFILE

Dr. Mahadev K. Patil is an experienced Assistant Professor at Bharati Vidyapeeth (Deemed to be) University, Abhijit Kadam Institute of Management and Social Sciences, Solapur with a demonstrated history of working in the Higher Education industry.

Prof. Dr. Pallavi P. Jamsandekar is an experienced Professor Bharati Vidyapeeth (Deemed to be) University, Institute of Management and Rural Development Administration, Sangli

Dr. Mrs. Shabnam S. Mahat is an experienced Assistant Professor Bharati Vidyapeeth (Deemed to be) University, Abhijit Kadam Institute of Management and Social Sciences, Solapur