Volume: 12 Issue:04 Pages: 4621-4626(2021) ISSN: 0975-0290

A Novel Approach to Restructure the Input Java Program

Dr. R. N. Kulkarni[1] and Aparna K.S[2]
[1]Department of Computer science & Engineering, Ballari Institute of Technology & Management, Ballari.
rn_kulkarni@rediffmail.com
[2]Assistant Professor, Department of CSE, RYMEC, Ballari, Research scholar, VTU, Belagavi.
aparna.vastard@gmail.com

ABSTRACT

During the last few years, we can find a lot of developments in the software industry concerning customized applications. Most of the businesses are automated using Java programming language because of its object-oriented features. The applications developed, might have undergone perennial need-based modifications. The original structure of the program is lost because of the need-based modifications and further, the program becomes ill-structured. This ill-structured program is not appropriate for the abstraction of design information.

In this paper, we are proposing a novel approach that takes executable java program as input and restructures the program without modifying its functionality. The proposed approach comprises the phases like appending the externals files to the main program, eliminating comment lines, blank lines, converting multi-statement lines to the single statement, the multi-line statement to single statement line, physically allotting line numbers, and removing unused variables.

Keywords: **Restructuring, ill-structured, functionality, design information.**

Date of Submission: Jan 15, 2021 Date of Acceptance: Feb 08, 2021

1.INTRODUCTION

In today's ever-changing business world, the only thing that doesn't change is 'change' itself. Nothing is constant or predictable in the current challenging business environment. The software industry is dominated by java programming for the past two decades because of its versatility in programming concepts. The majority of the software applications are developed using java and these applications[1] are in great usage in different sectors for decades and hence have to maintain as they have a lot of business rules and policies embodied[2], which are mandate for the present working models.

Due to rapid technological advancements and perennial changes happening in the business, these software applications are undergoing a lot of changes[3] which make the software lose its originality, leading to poor program comprehension. Maintaining such software systems to abstract good design information is the real challenge which is done through restructuring.

2.TERMINOLOGY

Restructuring is the process of making changes to the software structure externally without changing its functionality.

Program comprehension is the process of understanding the functionality of the software using the given program code.

Functionality is the actual working principles of the given input with desired outcomes.

Software maintenance is the process of optimizing the given software with a feasible modification that increases the throughput.

3. LITERATURE SURVEY

In paper[4], the author has discussed in detail the role of comments in the maintainability and reusability of the software, provided, these comment lines should also be updated as the software gets updated. This paper has helped us to know the role of comments in program comprehension.

In this paper[5], the different code smells and the sequence of elimination are discussed which has a great impact on the maintenance of the software metrics like maintainability index, relative logical complexity, and so on. This paper has helped us to know the impact of code on software metrics.

In paper[6], the author has highlighted that replacing or re-building the existing software system is the most time-oriented and also economically infeasible to the industry because of the outdated technology and has proposed to modernize the system using reengineering and reverse engineering concepts.

In paper[7], the different techniques of restructuring java programs are discussed. It has given sufficient input for our methodology. These methods are applied with appropriate modifications.

Volume: 12 Issue:04 Pages: 4621-4626(2021) ISSN: 0975-0290

In paper[10], the different types of comments and their purposes for comprehending the program are discussed and this has helped to know the role of comments.

In paper[9,12] a thorough study is conducted on the different refactoring techniques for improving the design level information. The study signifies various refactoring operations improving maintainability, understandability, modifiability, and analyzability of the refactored software. It has given an idea on different levels of refactoring which is appropriately modified and applied.

In paper[11], the different tools used for code quality analysis are discussed. The code is checked for different refactoring methods are tried and concluded that some refactoring techniques are feasible and cost-effective and other higher forms are very complex, changing the structure and leading to costly affairs. This has helped me to know the different refactoring techniques which are feasible and cost-effective. The higher forms of refactoring are very complex changing the structure of the original program, which is not advisable.

In paper[13] author signifies the role of both inner and outer comments. The association of comments and the fault proneness of the code are highlighted. It is proved that methods having inner comments tends to be more faulty. It reveals the role of inner comments as faulty methods. This paper has helped me to know the significance of retaining comments in the software code.

In paper[15], the different techniques of restructuring are done on the C program for multi-core architectures are discussed. The author has discussed that the software becomes obsolete, at some point, and discarding the entire system is not an economically viable solution and has adapted the strategies used for the accelerated development of software with reusability concepts. In [21][22], the author has proposed the methodology for the restructuring of the C program for the abstraction of design information. In [23], the dead code is eliminated by identifying the unused variables, classes and methods. These redundant variables are identified using data flow analysis and single assignment property for the variables. This paper has given basic idea of dead code elimination. In our paper, the unused variables , are removed using regular expressions. In paper[24], the code quality can be improved by different refactoring methods for effective implementation. Further these complex refactoring techniques may change the structure of the system and require additional code for refactoring which leads to chain reaction. In this paper, the novel approach of restructuring are addressed by these following steps

i. Appending the packages, classes and interfaces to the input file
ii. Eliminate Comment lines
iii. Eliminate blank lines
iv. Transform multiple declaration statements into a single declaration statement
v. Allotting Line numbers to executable statements
vi. Identifying and Removing unused variables

4. METHODOLOGY

The various approaches used are discussed in detail in the literature survey and the methodology proposed in this paper is carried out using the following steps, which are made amenable for reengineering.

i. Appending the packages, classes and interfaces to the input file:

In Java , the concept of reusability is supported by importing the required classes, package and abstract interfaces from one file to another. In our methodology, we import these concepts physically, by copying the entire code from one location to the input file for easy parsing.

ii. Eliminate Comment lines :

The Comment lines give an overview of code and these comment lines have to be updated periodically as and when the software is modified but this does not happen in reality. These comment lines may also be used for commenting functions that were not needed, but the reason for commenting may not be specified. So the feasible way of restructuring the input program is to consider only the executable statements and ignore the comment lines in our input program.

iii. Eliminate blank lines:

The conventional method of programming is to insert blank lines for improving the readability of the program but these blank lines are removed as they do not contribute to any program logic. So the proposed methodology restructures by eliminating these blank lines in the input.

iv. Transform multiple declaration statements into a single declaration statement:

Consider the scenario where multiple statements are declared in a single line, separated by the delimiter "; ". These statements will be considered as a single node, which does not carry any meaning. So these multiple statements are restructured to appear as a single statement separated by the delimiter ", " in all declaration statements of the input program.

***iv. Transform multi-line statements into a single statement line*:**

While programming, some statements in the input program may be represented as the multi-line statements

Volume: 12 Issue: 04 Pages: 4621-4626(2021) ISSN: 0975-0290

and will be considered as separate nodes, without any meaning. So these are restructured to appear in a single line, contributing to a meaningful node.

v. Allotting Line numbers to executable statements:

The executable statements are kept on track by assigning some numbering by the tool which helps in better program comprehension. Hence in our methodology, the line numbers are allotted uniquely for all the executable statements after completion of the above restructuring steps.

vi. Identifying and Removing unused variables: During programming, the changes are done dynamically. In this process, some variables, are declared but not used due to the repeated modifications because of need-based requirements. Hence such variables are restructured by identifying and removing them as they occupy a lot of memory space and execution time in the program. Similarly the same can be done for classes and member functions.

5. ALGORITHM- RESTRUCTURING THE JAVA PROGRAM

INPUT: Java application program
OUTPUT: Restructured java application
Input: Executable java program
Output: Restructured java program
STRG[1…n]←file1.java

```
Step 1:[Append the imported packages]
        if(STR[i].contains "import package.*") then
            merge_file( file1.java, package.name)
        end if
Step 2:[Append the interfaces]
        if(STR[i].contains "import package.classname") then
            merge_file( file1.java, package.interface)
        end if
Step 3:[Append the imported classes]
        if(STR[i].contains "import package.classname") then
            merge_file( file1.java, package.classname)
        end if
Step 4:[Eliminate Comment lines]
        for i←1 to n
            Comment=false
            if STRG[i].contains="//"
                Comment=true
            end if
            if STRG[i].contains "/*" and ends with "*/"
                comment=true, i<-j ,print(STRG[i])
            end if
        end for
Step 5:[Eliminate blank lines]
        for i <- 1 to n
            if (!STRG[i].isEmpty()) then
                write-to(STRG[i]);
                write-to("\n")
            end if
        end for
Step 6:[Transform the multiple statements into single line]
        while (STRG!= NULL) do
        STRG = STRG
.restoreall( ";" "," "; \n")
        Writer-to(STRG);
        end while
Step 7:[Transform multiline statements into single line]
        while (STRG [1..n]!=';)
        STRG = STRG + STRG.restoreall( ";" , "; \n")
        Writer-to(STRG);
        end while
Step 8:[Allot line numbers]
        Lineno<- 1
        for i<- 1 to n
            Print(lineno + STRG[i])
        end for
Step 9:[Identifying variables declared but not used]
        for i←1 to n
        count[i]=1
  Storing variables in array a[i]
   for i<-1 to n
   if (STRG[i]== a[i]) count[i]++;
   if (count[i]==1) print("Variable a[i] not used")
```

V. ALGORITHM- RESTRUCTURING THE JAVA PROGRAM

INPUT: Java application program
OUTPUT: Restructured java application
Input: Executable java program
Output: Restructured java program
STRG[1…n]←file1.java

```
Step 1:[Append the imported packages]
        if(STR[i].contains "import package.*") then
            merge_file( file1.java, package.name)
        end if
Step 2:[Append the interfaces]
        if(STR[i].contains "import package.classname") then
            merge_file( file1.java, package.interface)
        end if
Step 3:[Append the imported classes]
        if(STR[i].contains "import package.classname") then
            merge_file( file1.java, package.classname)
        end if
Step 4:[Eliminate Comment lines]
        for i←1 to n
            Comment=false
            if STRG[i].contains="//"
                Comment=true
            end if
            if STRG[i].contains "/*" and ends with "*/"
                comment=true, i<-j ,print(STRG[i])
            end if
        end for
```

Volume: 12 Issue:04 Pages: 4621-4626(2021) ISSN: 0975-0290

Step 5:[Eliminate blank lines]

```
for i <- 1 to n
    if (!STRG[i].isEmpty()) then
       write-to(STRG[i]);
       write-to("\n")
    end if
end for
```

Step 6:[Transform the multiple statements into single line]

```
while (STRG!= NULL) do
STRG = STRG
.restoreall( ";" "," "; \n")
 Writer-to(STRG);
end while
```

Step 7:[Transform multiline statements into single line]

```
while (STRG [1..n]!=';)
STRG = STRG + STRG.restoreall( ";" , "; \n")
 Writer-to(STRG);
 end while
```

Step 8:[Allot line numbers]

```
Lineno<- 1
for i<- 1 to n
     Print(lineno + STRG[i])
end for
```

Step 9:[Identifying variables declared but not used]

```
for i← 1 to n
count[i]=1
Storing variables in array a[i]
 for i<-1 to n
 if (STRG[i]== a[i]) count[i]++;
 if (count[i]==1) print("Variable a[i] not used")
```

VI. RESULTS AND DISCUSSIONS

Table 6.1: Restructuring java-code by Importing packages to the original code.

```
import MyPackage.MeClass;
public class PName
{
  public static void main(String args[])
  {
    // Initializing the String variable
    // with a value
    String name = "restructuring";
    int myN = 5;
    int myFNum = 5.99f;
    char myLett = 'D';
    boolean myBo = true;
    String myTe = "Hello";
    System.out.println(myN);
    System.out.println(myFNum);
    System.out.println(myLett);
    System.out.println(myBo);
    System.out.println(myTe);
    // Creating an instance of class MeClass in
    // the package.
    MeClass obj = new MeClass();

    obj.getNames(name);
  }
}
```

Figure: 6.1 Java program before importing

```
package Mepack;
public class MeClass
{
   public void create-Names(String s1)
   {
      System.out.println(s1);
   }
}
public class PName
{
 public static void main(String args[])
  {
    // Initializing the String variable
    // with a value
    String name = "restructuring";
    int myN = 5;
    int myFNum = 5.99f;
    char myLett = 'D';
    boolean myBo = true;
    String myTe = "Hello";
    System.out.println(myN);
    System.out.println(myFNum);
    System.out.println(myLett);
    System.out.println(myBo);
    System.out.println(myTe);
    // Creating an instance of class MeClass in
    // the package.
    MeClass obj = new MeClass();
   obj.getNames(name);
  }
}
```

Figure: 6.2 Java program after importing

```
1
2 package mePack;
3 public class MeClass
4 {
5    public void create-Names(String s1)
6    {
7       System.out.println(s1);
8    }
9 }
10 public class PName
11 {
12   public static void main(String args[])
13   {
14      String name = "restructuring",myTe = "Hello";
15      int myN = 5, myFNum = 5.99f;
16    char myLett = 'D';
17    boolean myBo = true;
18    System.out.println(myNum);
19    System.out.println(myFloatNum);
20    System.out.println(myLetter);
21    System.out.println(myBool);
22    System.out.println(myText);
```

```
23      MeClass obj = new MeClass();
24      obj.getNames(name);
25    }
26  }
```

Figure: 6.3 Restructured java code

In Fig 6.1 a small java program using user-defined import packages is taken as input. This program code is getting restructured by substituting the entire imported package, as shown in Fig 6.2. This java code in Fig 6.2 acts as input to the next level of restructuring, which is done using the above-discussed techniques. The output java code after this restructuring is shown in Fig 6.3. This restructured code has imported the entire package with no comment lines, no blank lines, multiline declarations in a single line, has physical line numbers.

6. CONCLUSION

In this paper, an attempt is made to restructure the input java program. The proposed tool restructures the input java program by first appending the imported files to the main program and then eliminating the comment lines, blank lines, and converting the multi-statement lines to the single statement, the multi-line statement to a single statement line, assigning line numbers, eliminating unused variables. The tool is tested for its correctness and completeness and presently working for all the restructuring aspects discussed above.

Further this work will be continued to address the issues related to the speed and memory.

7. REFERENCES

[1] Mohammed Furqan Shreetha Bhat "The Secrets Behind the Product-Making Process ", Int. Jnl. of Advanced Networking & Applications (IJANA) Special Issue, Volume 10, Issue 5(March-April 2019)

[2] Dr.Raghavi K Bhujang, Dr. Suma V "Need for Effective Risk Management Strategy duringSoftwareDevelopment Process – A Holistic View", Int. Jnl. of Advanced Networking & Applications (IJANA) Special Issue, Volume 10, Issue 5(March-April 2019).

[3] Mubarak Albarka Chen Zhanfang "A Comparative Study of Dynamic Software Testing Techniques ", Int. Jnl. Advanced Networking and Applications Volume: 12 Issue: 03 Pages: 4575-4584(2020) ISSN: 0975-0290 4575 "

[4] Elman Abdullah Alomar, Mohamed Wien Mkaouer, Ali Ouni, Marouane Kessentini "On the Impact of Refactoring on the Relationship between Quality Attributes and DesignMetrics",978-1-7281-2968-6/19/$31.00©2019IEEE,10.1109/ESEM.2019.8870177.

[5] Dr. R N Kulkarni, Pani Ram Prasad, " Restructuring of Java Program to be amenable for Reengineering", Journal of Engineering Science and Technology Volume 02, Issue 06 (2019) P.

[6] Ally s. Nyamawe, hui Liu 1, Shandong niu1," Recommending Refactoring Solutions Based on Traceability and Code Metrics", 2018. Page no: 49460 – 49475, 06 September 2018, ISSN: 2169-3536.

[7] Satwinder Singh, Sharanpreet Kaur, "A systematic literature review: Refactoring for disclosing code smells in object-oriented software", http://dx.doi.org/10.1016/j.asej.2017.03.002, December 2018, Pages 2129-2151

[8] Mehmet Kaya1, Shannon Conley2, Zhala S. Othman3, Asaf Varol, "Effective Software Refactoring Process", 2018 6th International Symposium on Digital Forensic and Security (ISDFS), 22-25 March 2018, DOI: 10.1109/ISDFS.2018.8355350

[9] Yukti Mehta, Ashish Sureka, Paramvir Singh "Analyzing Code Smell Removal Sequence for Enhanced Software Maintainability", 2018 Conference on Information and Communication Technology (CICT), 26-28 Oct. 2018, https://doi.org/10.1109/INFOCOMTECH.2018.8722418

[10] SeetharamaTantry, Muralidhar N.N. "Implications of legacy software system modernization – a survey in a changed scenario ", International Journal of Advanced Research in Computer Science,http://dx.doi.org/10.26483/ijarcs.v8i7.4556 Volume 8, No. 7, July – August 2017.

[11] Shaik Ismail, Sai Koparthi, "Automatically restructuring of java comments", Published in conference July 2017, Washington D.C, U.S.A. PP-1-4.

[12] Dr. R N Kulkarni, Padma Priya Patil, "Restructuring of Legacy 'C' programs to be amenable for multicore architecture", Science Direct, Elsevier, 876-6102@2017, 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon).

[13] Luca Pascarella, Alberto Bacchelli, "Classifying code comments in Java open-source software systems", 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), 20-21 May 2017, 10.1109/MSR.2017.63.

[14] Raja Sehrab Bashir, Sai Peck Lee, Chong Chun Yung, "A methodology for impact evaluation of refactoring on external quality attributes of a

software design", 0-7695-6347-3/17/$31.00 ©2017. 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon), 17-19 Aug. 2017, DOI: 10.1109/SmartTechCon.2017.8358389.

[15] A.Cathreen Graciamary and Dr.Chidambaram, "Enhanced Re-engineering Mechanism to Improve the Efficiency of Software Re-engineering", International Journal of Advanced Computer Science and Applications vol.7, No 11, 2016.

[16] Anna Vasileva, Doris Schmedding, "How to Improve Code Quality by Measurement and refactoring", DOI 10.1109/QUATIC.2016.28, 2016-IEEE 10th International Conference on the https://ieeexplore.ieee.org/xpl/conhome/7811933/proceeding.

[17] Isong Bassey, Nosipho Dladlu," Object-Oriented Code Metric-Based Refactoring Opportunities Identification Approaches: analysis", 978-1-5090-4871-7/16 $31.00 © 2016 IEEE, DOI 10.1109/ACIT-CSII-BCD.2016.24, 12-14 Dec. 2016.

[18] Dr. R N Kulkarni, Nidhi Jain C, Rashmi G, Vaishali B J, Zakiya Niyazi, "Abstraction of Test Cases From Input Java Program", International Journal of Combined Research & Development (IJCRD) ISSN:2321-25X; ISSN:2321-2241 Volume: 4; Issue: 5; May -2015.

[19] S.H. Kannangara1 and W.M.J.I. Wijayanayake2 "An empirical evaluation of the impact of refactoring on internal and external measures of code quality", International Journal of Software Engineering & Applications (IJSEA),2015, DOI:10.5121/idea.2015, IJSEA Journal, Vol.6, No.1, January 2015, PP. 51 – 67

[20] Oscar Chaparro, Gabriele Bavota, Andrian Marcus, Massimiliano Di Penta, "On the Impact of Refactoring Operations on Code Quality Metrics", 29 Sept.-3 Oct. 2014 29 Sept.-3 Oct. 2014, 978-1-4799-6146-7, ISSN:1063-6773

[21] Hirohisa Aman, Sousuke Amasaki, Takashi Sasaki, and Minoru Kawahara "Empirical Analysis of Fault-proneness in Methods by Focusing on their Comment Lines", 1530-1362/14 $31.00 © 2014 IEEE, DOI 10.1109/APSEC.2014.93, 1-4 Dec. 2014, INSPEC Accession Number: 15076974

[22] Dr. R N Kulkarni, Shilpa Jain, Bhavana S.H,Nethravathi .K, Shalini .S "Abstraction of UML Diagrams from Java Code ", International Journal of Combined Research & Development (IJCRD) eISSN: 2321- 225X; pISSN: 2321-2241 Volume: 2; Issue: 4; April- 2014.

[23] Rekha Naug1 and Kavita "A study of types of dead codes and their solutions", International Journal of Computer Science and Engineering (IJCSE) ISSN(P): 2278–9960; ISSN(E): 2278–9979 Vol. 9, Issue 2, Feb–Mar 2020, 1–6.

[24] Anna Vasileva, Doris Schmedding, "How to Improve Code Quality by Measurement and Refactoring", 2016 10th International Conference on the Quality of Information and Communications Technology.

[25] Dr. Shivanand M. Handigund, Rajkumar N. Kulkarni "An Ameliorated Methodology for the design of Object Structures from legacy 'C' Program", ©2010 International Journal of Computer Applications (0975 – 8887) Volume 1 – No. 13.

Biographies and Photographs

Dr.R.N.Kulkarni, Ph.D in Software engineering from visveswaraiah technological university, Belagavi, Karnataka, India. Presenthy heading the Department of Computer science and Engineering, Ballari Institute of Technology and Management, Ballari, Karnataka, India. He has more than 30 years of experience in teaching with passionate teaching and getting involved in many aspects of academic growth like International and National conferences ,possessing 65 publications. His Research domains include software engineering, DBMS, Soft Computing and object Technology. He is an active person in the process of NBA and NAAC. He has chaired many conferences and involved actively in various committees.

Aparna K.S working as Assistant professor in the Department of Computer Science and Engineering at Rao Bahadur Y Mahabaleswarappa Engineering college, Affiliated to VTU, Belagavi with teaching experience of 16 years, pursuing Phd degree in Software engineering. Passionate about the teaching learning process and involved in many innovative projects. Paticipated in many Interntional and National conferences Workshops and Seminars with many paper publications. An Active member in various academic committees like NBA and NAAC.