International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-7 Issue-4, September 2018

Code-Smells Identification by using PSO Approach

G. Ramesh, Ch. Mallikarjuna Rao

Abstract*: Code-smell defines the smells which arise in coding part of the software development life cycle. It is very crucial to detect smells in software projects. If smells are detected earlier then the possibility of occurrence of errors, faults will be reduced. Hence, quality of the software is improved. The existing work used Bayesian approaches, manual approaches and search-based approaches to detect smells. These approaches lack in getting optimization solutions in detecting process. So, paper makes use of one of the popular optimization technique called Particle Swarm Optimization (PSO) for detecting the smells in programming part. The technique shows how intelligently the smells are detected and mainly concentrated on five types of smells namely Long Methods, Long Parameters, Large Classes, Duplicated Codes, and Primitive Obsessions. Implementation of this technique is, considering source-code of any software applications or programs and injecting PSO technique into the system. Here, PSO has trained to detect five types of smells whenever their appear in the source-code. Detecting the smells in initial stages of the project gives best performance of the software, and in other hand quality of the software is achieved. Experimental results are shown by using PSO technique, where searching time will be less consumed and accuracy of the system is gained.

***Index Terms*: Code-Smells, Duplicated Codes, Software Quality, PSO.**

I. INTRODUCTION

Software development life cycle contains the activities that are performed according to plan. Activities include requirement analysis, designing, coding, testing, and maintenance. At coding phase code-smells are identified. Smell is a hint that tells something is going wrong somewhere in the program. Suppose, smells are neglected obviously they turns to errors, faults, anomalies and sometimes software failures will also occurs.

To detect smells in existing work, Wael Kessentini et al. [1] used various strategies and approaches like manual, probability, metric-based [2], visualized-based and search-based approaches [3]. Each approach have their advantages and disadvantages over other approaches, Wael concentrated mainly on search-based approaches, in search-based approaches for detecting smells each and every line of the program is searched. Due to this, searching time will increases and time complexity issues are considered.

To overcome this, paper address one of the stochastic technique called Particle Swarm Optimization, for detecting some of the smells called Long Methods, Large Classes, Long Parameters, Duplicated Codes, Primitive Obsessions. The reasons why the paper only addressing these smells is because, detecting "n" number of smells are not worthy. Smells should be identified based on the severity that is affecting in the source-code of a program. These are the smells that will occur frequently in source-code of the software project or applications.

A. Organization of the paper:

Section II talks about the work that has already given for detecting design-smells and the approaches that have been used. Section III describes the proposed implementation on code-smells that occur in coding phase and the technique used is also explained. Section IV speaks about the results that have been experimentally performed by using Netbeans software and the detectors are shown in graph format. Conclusions and future work are explained in section V and last but far from list is all about references.

II. RELATED WORK

In this section, smells like BLOB (Binary Large OBject), spaghetti code, and functional decompositions are detected based on search-based approaches by using genetic algorithms [4] and genetic programming [5]. These are the smell that rises in design phase of the software development and the type of the smells is called design-smells.

By using search-based approaches, searching time of smells takes more, because each and every line of the coding part is considered for detection of smells and another thing is optimization of the approach is quit low. To this extension, considering five frequently affecting code-smells in the coding phase of the software development life cycle, to overcome optimization problems one of the dynamic technique called particle swarm optimization is considered for detecting and identifying the mentioned code-smells called Long Methods, Large Classes, Long Parameters, Duplicated Code, and Primitive Obsessions.

III. PARTICLE SWARM OPTIMIZATION

Proposed work is the extension of desig-smells i.e., detection of code-smells that arise in coding part of the cycle so called as code-smells and definition is given by fowler [6] "Code-smell is a surface indication that usually corresponds to a deeper problem in the system". The mentioned code-smells are detected by using PSO technique. Before describing about PSO, five types of smells are expalined in detailed.

Revised Manuscript Received on 30 September 2018.
***** Correspondence Author
Dr. G. Ramesh*, Associate Professor, Department of Computer Science and Engineering, Gokaraju Rangaraju Institute of Engineering and Technology, Bachupally, Hyderabad (Telangana), India.
Dr. Ch. Mallikarjuna Rao, Professor in CSE Department, Gokaraju Rangaraju Institute of Engineering and Technology, Bachupally, Hyderabad (Telangana), India.

a) Long Method

As the name specifies the method which is long is usually difficult to read, understand to the programmer or viwer of the program. So, all other things being equal, a shorter method is easier to read, to troubleshoot and to understand. Normally, signs of Long Method is that contains too many Lines of Code (LOC). Adding too many methods to a lines of code looks ugly. If the method is shorter then it is easier to understand even reading of the method is also more comfortable. To detect the mentioned code-smells one of the dynamic and popular tehnique has been proposed and it is PSO.

b) Long Parameter List

The parameters that are assigning to the method should be relevant to particular method. Limit the number of parameters that are needed in the method. The more parameters the method as the more complex it is. In general for short programmes say, more than tree or four parameters found for a particular method then list of parameters are considered as long parameter list. Providing unnecessary parameters to a particluar method leads to long parameters. There is a need to check what are the parameters that are passed to a method in a class. These parameters also called as arguments, if there are several unrelated data elements, sometimes one can merge several arguments into single parameter objects.

c) Large Class

If the class is having too many methods, parameters, attributes, variables automatically class becomes large. Whatever is needed to the class, those components should be mentioned. If the class contains too many responsibilities it is difficult to analyse the program. Sometimes large classes cannot be broken into smaller or unable to resturucture too. So, proper techniques should be used and it is PSO, which is successfully applied on behaviour of group of birds that are searching for the piece of food in a particluar search area and same idea is applied on detecting smells in the source-code.

d) Primitive Obsessions

Obsession means that number in more that means having more than enough in the form of irrational type. The primitive data types whatever used in a particluar class should not be more primitives are often used to “simulate” types. Primitives are namely integers, characters, float data types. The data type used for finding square root of a number is float data type but instead of float, integer or character data type is used then it is the major problem indicated with irrelevant primitives. Using more number of data types in a class is also one of the major reasons for primitve obsessions. Uses PSO for detecting primitive obsessions and can replace data values with objects creating primitive fields are very easy than making whole new class.

e) Duplicated Code

Not only in the coding phase, duplicates should not encourged in any phase of the software development. Due to this inconsistency, redundancy will increases. By this, there will be affect on performance of the software. In simple, whenever the repeated names or part of the repeated code is found just identifgy and removing should be done. It is like poison in any software development process. Programmers sometimes unknowingly mention same data twice or morethan twice and unable to identify also, if the program is short identifying duplicates is little easy. Suppose, same happened in bulky programs that time due to duplicates, quality of the software will be reduced. To identify duplicates their is a method called particle swarm optimization is used and for correcting the smells refactoring will be used.

This is complete description about the types of the smells that are taken in the present paper. For detecting all five types of smells PSO technique has been used. Generally PSO [7, 8] is an stochastic algorithm, where each particle is found intelligently in an stimualted time. It is an most popular algorithm used in any field in different reasearches. In this paper optimization is used for identifying the smells that arasies in coding part. At first, this technique is performed on bunch of the birds and it works as follows.

A. Study of Algorithm:

The story begins with the group of birds. In this study, each bird is considered as a particle. In this process birds are very hungry therefore they starts searching for the food in a particular search area. Now, group of birds are splitted themselves for finding the piece of food. Say, in the first iteration the bird din’t found the food, then it makes a sound that they is no food in an particular area. By doing this the other birds will alert and won’t go to the place where first bird have searched, by this time complexity and searching process will be compressed. This process will goes on untill the piece of food is found. By this procedure one can understand how intelligently the bunch of birds are searching for a food in a given search area. In every iteration the distance of searching will be reduced, to the end every bird is satisfied with the food they have found.

Now same methodology is applied for detecting the code-smells. At first, the optimizer starts identifying the code-smells that are mentioned like long methods, long parameter list, large class, duplicated codes and primitive obsessions. Detector starts finding the smells in the given file. The detector is trained with the constraints like, if more than 750 lines of code is exists in the given class file then that smell is consider as large class. In the same manner other smells also be detected. Following are the rules provided for smells mentioned.

R1: IF length of the method is > 80 lines THEN smell = long method

R2:IF((LOC>=700)|((LOC>=350) AND (NOC>=20))) THEN smell = large class

R3:IF((N>2)&(N>AVG.PARAMETERS+2))| (N>10) THEN smell = long parameter

Where, N = number of parameters of m

M = number of methods in C

C = class

$$\text{AVG. PARAMETERS} = \frac{\sum n \text{ parameters of a method}}{M}$$

For all the methods in C.

R4: IF number of parameter variables exceeds the max. Number of the corresponding variable value THEN it is considered as primitive obsession.

In this process one of the popular mertic precision also used in order to find the fitness value.

$$Precision = \frac{number\ of\ smells\ that\ have\ been\ acknowledged\ and\ resolved\ by\ the\ user}{total\ detected\ smells}$$

Functionality of the detector:

B. Developer:

Normally the framework starts with the developer who is developing the project. At first, developer won't bother about smells, just he starts writing the code. Nothing but initialization of the process.

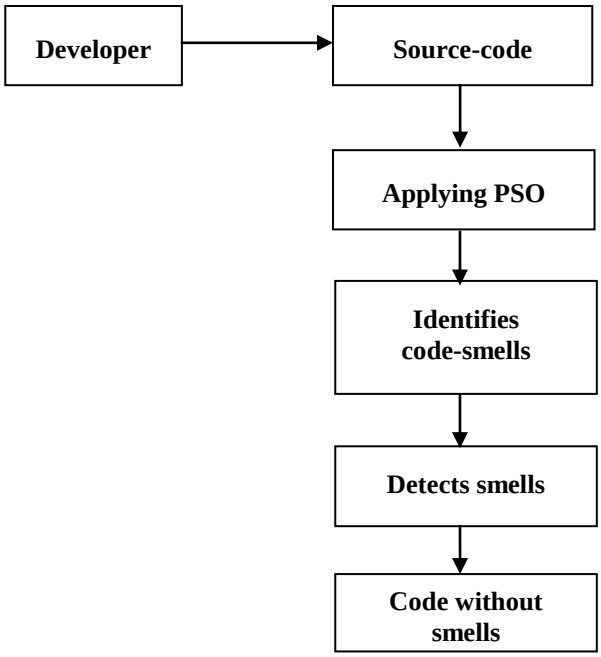

Figure 1: Procedure for Detecting Smells

C. PSO Detector:

At this stage, the proposed implementation comes into picture. Previously many techniques are used for detecting code-smells. Compare to those, PSO is faster, cheaper and even used in many research applications. Before applying on the concept it have successfully performed on finding the food for bunch of birds.

D. Identifying Code-Smells:

Finding smells through this technique is little more easy for the developer as well as the testers. If one wants high quality in their project code, they have to look after the smells without neglecting. Finding more severe smells gives more accurate results as well as improvement in quality. Accuracy will depends on the metric used i.e., precision and precision tells "code-smells that are correctly detected among the set of all detected code-smells"

After identifying or detecting the smells one can remove or correct the smells based on their criteria. For correcting the smells they are huge amount of refactoring tools like crawler, move method, J Refectory, In refactoring [9, 10].

IV. RESULTS

Java file is taken as the input and the programs are written in java language by using Net Beans software on windows operating system. After writing the program the detector is injected into the system for finding the various smells in a particular program. For every method or class, rules are provided as mentioned in previous section. If the rules are crossed then they are considered as smells. These smells are identified based on detector called PSO. Where, it searches the smells very intelligently in every iteration. After each iteration the developer or tester are very nearer to the smells.

At first code-smell detector should run, it displays a dialog box that contains files, directory and then process. For every program whatever files, classes, methods, constructors, variables and line count along with comments will be displayed.

Detecting each of the smells given are shown in NetBeans software by writing in java language for example showing the number of duplicates that a program contains. If the program is small then almost developer will be aware to not repeat the part of the code. If it is large program, unknowingly number of lines may be repeated with parameters. It should not be done because space wastage will occurs. So undoubtedly, remove the duplicates in the program before execution of the code. Figure 3 shows the duplicates in the program of the software project.

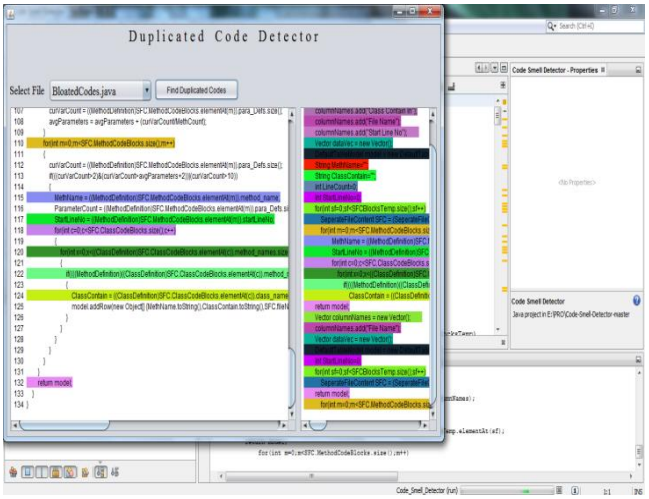

Figure 3: Duplicates in the Program

Counting the number of repeated lines or variables or methods can be done internally, this smell is very dangerous, so detecting or stamping out the duplicates are very important as well. Still now various types of smells have been detected by using PSO technique. For example, the program actually contains ten duplicates but the PSO have founded eight in that, precision is calculated i.e., number of smells that have been acknowledged based on the user and total number of smells found. By this, accurate probability is found 8/10 = 0.8, based on these calculations accuracy of the software project is known, by using the particle swarm optimization searching time will also gradually decreases in each iteration.

The graph is plotted between accuracy and searching time of the code-smells and is shown in figure 2, results are found by applying particle swarm optimization which is very famous technique for social behaviour of birds, ants, bacteria as mentioned. Innovatively by taking an idea same PSO has been applied for detecting code-smells in the source-code of any program given is the proposed work of the paper.

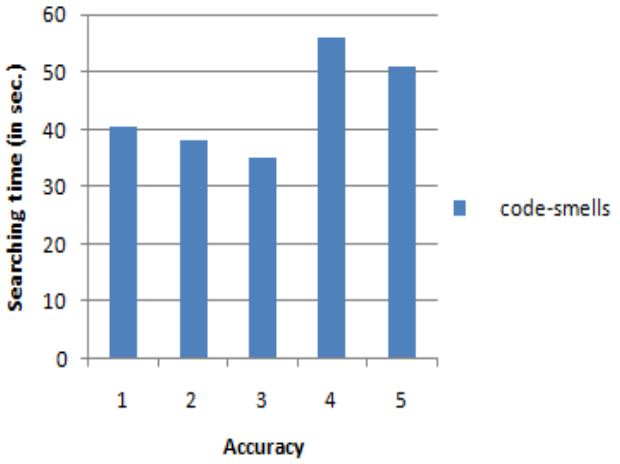

Figure 2: Accuracy of Detected Code-Smells

In the graph accuracy is calculated based on number of smells that have been detected with respect to number of lines of code. Overall detecting is performed on number of smells that have been acknowledged by the user by using PSO technique to the total number of actual smells that are present. Based on these calculations accuracy of the software system is achieved. By detecting smells in the programming phase gets quality software. PSO, is a stochastic optimization does not contain any genetic operators and searching of smells are performed intelligently. There by, graph is plotted between the searching time of each smell with respective to the accuracy of the detected smells. along with accuracy, efficiency of the system is achieved.

V.CONCLUSIONS & FUTURE WORK

In this paper for detecting the smells a new approach has been introduced. Previously many evolutionary algorithms are used for identifying the smells. Innovatively the popular technique called as Particle Swarm Optimization has been used for identification of various smells that are found in the execution of the program. At initial stages itself one should identify the smells by injecting detector technique, so that the possibility of errors, faults, even failure occurrences will be less. By this, performance and quality of the software will be improved. For every smell fitness value is calculated based on the severity of the smell. For calculating fitness value one of the metric called precision is used. The more precision value, the more accuracy of the system is achieved. For every iteration threshold values should be adjusted. If the value exceeds the given constraints then particular method, parameter or class is considered as smell and therefore, detection or removal of the smell is done.

This work only focused on five smells namely long method, long parameter, large class, duplicated code, and primitive obsessions. Future scope includes additional smells like too many literals, feature envy can also be allowed for detection. Secondly, in past design-smells are identified, now code-smells are detected in coding phase, likewise, future work may also includes smells those are raised in testing phase and can be mentioned as test-smells.

REFERENCES

1. Wael Kessentini, Houari Sahraoui, Slim Bechikh, and Ali Ouni, “A Cooperative Parallel Search-Based Software Engineering Approach for Code-Smells Detection”, IEEE Transactions on Software Engineering 40(9):841-861 · September 2014.
2. S. R. Chidamber and C. F. Kemerer, “A metrics suite for objectoriented design,” IEEE Trans. Softw. Eng., vol. 20, no. 6,pp. 293–318, Jun. 1994.
3. M. Harman, S. A. Mansouri, and Y. Zhang, “Search-based software engineering: Trends, techniques and applications,” ACM Comput. Surv., vol. 45, no. 1, 61 pages.
4. D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA,USA: Addison Wesley, 1989.
5. M. Tomassini and L. Vanneschi, “Guest editorial: Special issue on parallel and distributed evolutionary algorithms, part two,”Genetic Program. Evolvable Mach., vol. 11, no. 2, pp. 129–130, 2010.
6. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, Refactoring: Improving the Design of Existing Code. Reading, MA,USA:Addison Wesley, 1999.
7. Hemalatha. K, A. AnandaRao and Ramesh. G “Detection of Code-Smells by Using Particle Swarm Optimization Technique (PSO)”, South Asian Journal of Engineering and Technology Vol.2, No.28 Pp. 10–13, 2016.
8. Eberhart & Dr. Kernnedy, “particle swarm optimization”, IEEE, pp. 1942-1948, 1995.
9. P. Vignesh and P. Ramya, “Detection and Removal of Bad Smells instantly using a InsRefactor” , international Journal of Computer Science & Engineering Technology(IJCSET), Vol. 5 No. 04 Apr 2014.
10. W. J. Brown, R. C. Malveau, W. H. Brown, and T. J. Mowbray, Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis. Hoboken, NJ, USA: Wiley, 1998.