AN ANALYSIS OF DIFFERENT CODE SMELL APPROACHES WITH EGAPSO APPROACH

[1]Mr.S.James Benedict Felix, M.Sc., M.Phil., (Ph.D), [2]Dr.Viji Vinod
[1]*Research Scholar, Bharathiar University, Coimbatore – 641046, India.*
[2]*Professor & Head, Department of Computer Applications, Dr.MGR Educational and Research Institute University, Chennai – 600095, India.*

***Abstract*—** Software maintenance is an important component of any software that discovers its use in the day-to-day activities of any organization. Software maintenance is a difficult process if code smells exist in the software. The impact of the poor design of code is called code smells. Majority of the code smell detection approaches are rule-based, where rule-based approaches represent the combination of metrics and threshold. In this approach, rules are defined and detect the code smells are time to consume because identifying the accurate threshold value is a tedious job. For this issue, Euclidean distance based Genetic Algorithm and Particle Swarm Optimization (EGAPSO) approach is used. The approach is tested on the open source projects, likely Gantt Project and Log4j for identifying the five code smells namely Functional Decomposition, Blob, Spaghetti Code, Data Class and Feature Envy. Finally, this approach is compared with code smell detection using Genetic Algorithm (GA), DEtection and CORrection (DECOR), Parallel Evolutionary Algorithm (PEA) and Multi-Objective Genetic Programming (MOGP).

Keywords— *Code smell, Software metrics, EGAPSO*

I. INTRODUCTION

Software evolution and maintenance makes high costs of the development process, particularly as systems become more complex and larger. Software maintenance and evolution process is difficult in the structural design. These software design problem is known as code smells. The most of the code smells detection approaches are in the code level [1–5]. Code smell detection in the model level is very difficult as all the metrics are not supported in model level [6]. In rule-based approach, code smells cannot be detected because finding the right threshold value of the metrics is a tedious task. To be precise, a class may be considered as a large class in a program can be an average class in some other program. The EGAPSO is compared with the code smell detection approaches namely genetic algorithm (GA), DEtection and CORrection (DECOR), Parallel Evolutionary Algorithm (PEA) and Multi-Objective Genetic Programming (MOGP).

II. BACKGROUND AND PROBLEM STATEMENT

This segment provides the essential background material used for the detection of code smells. The basic definition of code smells, what are the metrics used for code smell detection and EGAPSO are discussed below:

A. Code smell

Code smells are first defined by Fowler and Beck. They defined code smells as the symptoms of code and design problems [9]. In this paper, the below five code smells were considered and evaluated.

Blob: The code smell Blob is found in designs where one large class monopolizes the behaviour of a system. It is a large class that has several fields and methods with low cohesion and most of the class does not have parent class and children. It is a class which implements various responsibilities and has the large size.

Functional Decomposition (FD): It occurs when a class is designed with the intent of performing a single function. The code smell FD will find in a class, when inheritance and polymorphism are poorly used. This kind of code smell may find in class diagram which is developed by non-experienced object-oriented developers.

Data Class (DC): DC is a class that has only data. It does not have any processing on the data. Data class is having state and does not perform any operation. The getter and setter methods are defined by this class.

Feature Envy: This code smell occurs when a method get fields of another method in some other class than the one it is actually implemented in. It is described by a large number of dependencies. It increases the coupling and reduces the cohesion of the class.

Spaghetti Code: This kind of code smell occurs when the code does not use suitable structuring mechanism. It prevents the use of object-oriented mechanisms, namely inheritance, and polymorphism. Typically causes by Inexperience design with object-oriented technologies.

B. Software Metrics

Software metrics gives useful information that facilitates assesses the quality of the software [10]. It can also be used to identifying the similarities between the software systems. Here, ten metrics are considered and these metrics are interrelated to the class entity in the class diagram. The metrics give information about the number of attributes and methods in the class diagram except NAss and Ngen. The metrics NAss and Ngen gives the relationship between classes. The description of metrics is listed in Table I.

TABLE I. METRICS DESCRIPTION

S.NO	Metrics	Description
1	NA	The total number of attributes per class
2	NPvA	The total number of private attributes per class
3	NpbA	The total number of public attributes per class
4	NprotA	The total number of protected attributes per class
5	NM	The total number of methods per class
6	NPvM	The total number of Private methods per class
7	NPbM	The total number of Public methods per class
8	NprotM	The total number of Protected methods per class
9	NAss	The total number of Associations
10	Ngen	The total number of Generalization relationships

III. EUCLIDEAN DISTANCE BASED GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION (EGAPSO)

The EGAPSO is a population-based heuristic search optimization algorithm and it was developed by Kim and Park. The EGAPSO is a hybrid approach of Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) [8] which is based on Euclidean distance. This algorithm is used to tune the proportional integral derivative (PID) controller in a steam temperature control system of the thermal power plant, biomedical process and industrial system of the chemical process [7]. Each particle in EGAPSO is called as individuals and a group of particles are called as a swarm. In the swarm, each particle has its own position and its velocity. In the beginning, the particles are placed at random positions in the search space. The velocity of the particle is represented as zero. The position of the particle and its velocity can be updated using the following formula.

$\upsilon i\ (t+1) \longleftarrow \omega\ \upsilon i\ (t) + c1r1(pbest(t) - xi\ (t)) + c2r2(gbest(t)$ ---------- (1)

$xi\ (t+1) \longleftarrow xi\ (t) + \upsilon i$ ---------- (2)

In the Equations (1) and (2),

xi (t) and xi (t+1) represent the position of the particle at the time (t) and time (t + 1), respectively. υi (t) is the velocity of the particle at time (t).

pbesti(t) is the best position of the particle found.

gbesti(t) is the global best position of the particles, c1 and c2 are the acceleration coefficients that influence the best position of the particles.

r1 and r2 are the random variables and x represents the inertia weight of the particles.

The position of a particle is lead by local best (pbest) and global best (gbest) factors in the search process. The best visited position for the particle by itself is the factor local best (pbest) and it arrives at the best position obtained so far by any particle in the neighborhood is global best (gbest).

IV. A SEARCH BASED APPROACH FOR DETECTING CODE SMELLS

In this paper, the EGAPSO approach examines detect the code smells, namely blob, data class, spaghetti code, functional decomposition and feature envy. The quality focus is the detection accuracy on code smells when compared to the GA approach, while the perspective is of other researchers, who want to evaluate the effectiveness of the approach in identifying code smells to build better recommenders for developers. The context of the study consists of three open source projects, like Gantt-Project, Log4j, and Xerces-J. Gantt-Project is a cross-platform tool for project scheduling. Log4j is a software package of Java-based. Finally, Xerces-J is a software package for parsing XML. Gantt-Project and Log4j are usually called the initial model and Xerces-J is used as the base example.

A. Precision

Precision signifies the fraction of correctly detected code smells over the detected code smells. From the value of the precision, someone can infer the probability that the detected code is accurate.

Precision ={ (Relevant Code Smells) ∩ (Detected Code Smells) } / (Detected Code Smells)

B. Recall

Recall represents the fraction of correctly detected code smells in the set of manually detected code smells to find out how many code smells have not been missed. From the value of recall, one can infer the probability that an expected code smell is detected.

Recall = {(Relevant Code Smells) ∩ (Detected Code Smells) / (Relevant Code Smells)

C. Average number of defects detected

Average Number of Defects Detected (ANDD) is equivalent with the fraction of the defects detected by the approach over by the number of defects that are actually present.

Average Number of Defects Detected=Number of defects detected / Number of defects actually present

D. $F_{measure}$

Fmeasure is defined as the harmonic mean of precision and recall.

Fmeasure = 2* |Precision * recall / Precision + recall| %

Experimental setup - For the code smell detection approach, three open source software are reused namely GanttProject, Log4j, and Xerces-J. In this approach, the initial models are GanttProject, Log4j. Xerces-J is the base example. First metrics are computed from the initial model using UML generator plugin in Netbeans. After that code smells are detected from the base example. Lastly, the code smells in the initial model are detected using EGAPSO. Open source software including a number of classes and the number of detected code smells are listed in Table II. Table III described the parameter of EGAPSO and GA.

TABLE II. CODE SMELL PRESENT IN GANTTPROJECT.

S.No	Code Smells	Gantt project	Log4j.
1	Number of classes	245	227
2	Number of blobs	10	3
3	Number of data class	10	5
4	Number of functional decomposition	17	11
5	Number of feature envy	11	2
6	Number of spaghetti code	16	8

TABLE III. PARAMETER SETTING OF EGAPSO AND GA.

S.No	Algorithms	Parameters	Values
1	EGAPSO	Population size	100
		Number of generations	500
2	GA	Population size	100
		Number of generations	500

TABLE IV. PRECISION AND RECALL VALUES FOR GA.

S.No	Open Source Software	Defects	Precision (%)	Recall (%)	ANDD (%)	F measure (%)	Specificity (%)	AUC (%)
1	Gantt Project	Blob	100	90	90	94	100	95
		Functional Decomposition	100	47	47	64	100	73
		Feature Envy	100	81	82	90	100	90
		Data Class	90	88	100	90	99	93
		Spaghetti code	90	62	68	74	99	81
2	Log4j	Blob	100	97	100	100	100	99
		Functional Decomposition	100	63	64	77	100	82
		Feature Envy	50	47	100	50	99	73
		Data Class	75	60	80	66	99	80
		Spaghetti code	80	50	62	61	99	75

TABLE V. PRECISION AND RECALL VALUES FOR EGAPSO.

S.No	Open Source Software	Defects	Precision (%)	Recall (%)	ANDD (%)	F measure (%)	Specificity (%)	AUC (%)
1	Gantt Project	Blob	100	95	100	100	100	98
		Functional Decomposition	100	76	76	86	100	98
		Feature Envy	100	82	82	100	100	91
		Data Class	100	98	100	90	100	99
		Spaghetti code	100	81	81	94	100	91
2	Log4j	Blob	100	97	100	100	100	99
		Functional Decomposition	100	81	81	89	100	91
		Feature Envy	100	97	100	100	100	99
		Data Class	80	77	100	80	99	88
		Spaghetti code	83	62	75	71	99	81

V. A SEARCH BASED APPROACH FOR DETECTING CODE SMELLS

The evaluation of EGAPSO approach was performed on the open-source system, namely GanttProject, Log4j, and Xerces-J. The initial models are Gantt Project and Log4j. The Xerces is a base example. First, the metrics are calculated from the initial model and base example, then the code smells are detected from the base example using code smell detection tool (infusion). Then with the help of EGAPSO approach, the code smells are identified from the open source project. The EGAPSO approach can recognize the code smells namely blob, Functional decomposition, Feature envy, Data class and Spaghetti code. These code smells are more accurately when compared with the approach of Genetic Algorithm (GA).

The measures precision, recall, average number of defects detected (ANDD), Fmeasure and area under the ROC curve (AUC) have been calculated for the accuracy of EGAPSO approach. The computed values of all the measurement for genetic algorithm and EGAPSO are listed in Table-IV and V.

TABLE VI. COMPARISON OF PRECISION VALUES FOR THE CODE SMELL DETECTION APPROACHES WITH EGAPSO APPROACH.

Open Source Software	Defects	Precision of EGAPSO(%	Precision of GA(%)	Precision of DECOR(%)	Precision of PEA(%)	Precision of MOGP(%)
Gantt Project	Blob	100	100	90	93	83
	Functional - Decomposition	100	88	26.7	88	77
Log4j	Blob	100	100	100	82	--
	Functional - Decomposition	100	100	54.5	93	--

To further analyze the effectiveness of the proposed approach, the precision of the EGAPSO approach is compared to the other state of the art approaches such as GA (Ghannem et al., 2016), DECOR [11], PEA [12] and MOGP [13] using precision values mentioned in TableVI. The precision values for the state of the art approaches in TableVI are obtained directly without implementation from these works. The comparison of the approach EGAPSO over the existing GA in terms of precision, recall, average number of defects detected and $F_{measure}$ values reveals the efficiency and accuracy of the EGAPSO over the GA.

VI. CONCLUSION AND FUTURE WORK

Using the EGAPSO approach, five code smells namely blob, functional decomposition, feature envy, data class and spaghetti code has been detected from the open source software namely Gantt Project and Log4j. The advantage of the approach is to identify the code smells present in the model level of the open sources. At the same time, most of the existing approaches cannot be used for identifying code smells present in the model level of the software. The approaches EGAPSO and GA are evaluated code smells and the results revealed the completeness and correctness of EGAPSO over the existing Genetic Algorithm. The EGAPSO approach is also increased the average precision, recall, average number of defects detected and $F_{measure}$ over the existing approach. In future, EGAPSO approach will be applied to other open source projects.

VII. REFERENCES

[1] H. AliKacem, H. Sahraoui, De´tection D'anomalies Utilisant UnLangage De Description De Re`gle De Qualite´ , in: Actes Du 12e Colloque LMO, 2006, pp. 185–200.

[2] K. Erni, C. Lewerentz, Applying Design-Metrics to Object- Oriented Frameworks, in: Proceedings of the 3rd International Software Metrics Symposium, IEEE, 1996, pp. 64–74.

[3] G. El Boussaidi, H. Mili, Understanding design patterns—what is the problem?., Software: Practice Experience 42 (12) (2012) 1495–1529.

[4] R. Marinescu, Detection strategies: metrics-based rules for detecting design flaws, in: Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM),

2004, pp. 350–359.

[5] M. Kessentini, H. Sahraoui, M. Boukadoum, M. Wimmer, Search-based design defects detection by example, in: Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software, Springer, (Saarbru¨Cken, Germany), 2011, pp. 401–415.

[6] H. Kim, J. Park, Improvement of genetic algorithm using PSO and Euclidean data distance, Int. J. Inf. Technol. 12 (3) (2006) 142–1487

[7] M. Harman, L. Tratt, Pareto optimal search based refactoring at the design level, in: Proceedings of the 9th

Annual Conference on Genetic and Evolutionary Computation, ACM, London, England, 2007, pp. 1106–1113.

[8] D.E. Goldberg, Genetic Algorithms in Search, Addison-Wesley, Optimization and Machine Learning, 1989.

[9] M. Fowler, K. Beck, J. Brant, Refactoring: improving the design of existing code, in: Proceeding of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods, Springer, 1999, p. 256.

[10] N.E. Fenton, S.L.Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Co., Boston, MA, 1998.

[11] N. Moha, Y.G. Gueheneuc, L. Duchien, A.F. Le Meur, DECOR: a method for the specification and detection of code and design smells, Software Eng., IEEE Trans. 36 (1) (2010) 20–36

[12] W. Kessentini, M. Kessentini, H. Sahraoui, S. Bechikh, A. Ouni, A cooperative parallel search-based software engineering approach for code-smells detection, IEEE Trans. Software Eng. 40 (9) (2014) 841–861.

[13] U. Mansoor, M. Kessentini, B.R. Maxim, K. Deb, Multi objective code-smells detection using good and bad design examples', Software Quality J. (2016) 1–24.