ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, *25-27 June*, *2017*, *pp. 1224 -1229*
doi: 10.18178/wcse.2017.06.213

Detection of Stored Procedure Bad Smells

Sutthikan Naowarat [1 +] and Pornsiri Muenchaisri [1]

[1] Department of Computer Engineering, Chulalongkorn University, Bangkok Thailand,

Abstract. Stored procedures are commonly used in accessing and manipulating data in large-scale system development to optimize the query database, reduce the application workloads and reduce the traffic problems between the database and the application. If source code of stored procedures have bad smells, it will have impact in modification, and eventually have a negative impact on their quality and maintainability. This research proposes Tree Diagram and Context Analysis approach in detecting six different bad smells of stored procedures. The tree diagram approach is the comparison tree diagram of bad smells and source code which is written in PL/SQL. The context analysis approach is the creation of rules and qualifications of bad smells for increasing the accuracy in detection. In addition, this research explains the overview process, the algorithm process, and has uses example source code. The evaluation uses MI (Maintainability Index) to present the approach which are practical and effective.

Keywords: stored procedures, detection bad smells, tree diagram, context analysis, maintainability index

1. Introduction

Stored procedures are commonly used in accessing and manipulating data in large-scale system development to optimize the query database, reduce the application workloads and reduce the traffic problems between the database and the application. The inefficient source code developed by programmers for stored procedures creates bad smells. The stored procedures bad smells have serious problems on the overall system, have impact on modification, and eventually have a negative impact on their quality and maintainability[1][2].

Most of the previous researches about stored procedures aimed to explain the meaning and discover techniques[3][4] to improve or to identify flaws according to custom conditions[5]. Also, the bad smells researches mostly about object oriented language. However, stored procedures in this research is developed with PL/SQL which is a structured language and designed as block, declaration section block, execution section block, and exception section block[6].

This research proposes an approach to detect bad smells of stored procedures using Tree Diagram and Context Analysis. The tree diagram is a process of analyzing bad smells to find the attributes and generate bad smells tree diagram, and then generate source code tree diagram to compare tree diagram structure or members of the child node set using propose algorithm and BFS (Breadth First Search). The context analysis is extracted using textual analysis techniques to create conditions or rules for bad smells which can be used to identify bad smells in the source code. The evaluation of the research uses the results of maintainability index which collects pre-experiment and post-experiment values as a performance benchmark.

The remainder of the paper is organized as follows: Section 2 states the related work. Section 3 presents the process and illustrates the structure and algorithm of the approach for detecting bad smells. Section 4 mentions case study and Section 5 presents the conclusion.

2. Related Work

[+] Corresponding author. Tel.: + 66816947784; fax: +6622867021.
E-mail address: Sutthikan.N@Student.chula.ac.th.

Dimas C. Nascimento et al. [5] presented PL/SQL Advisor tool to improve the efficiency and code quality of stored procedures by using control dependence tree (CDT) and control flow graph. Tan Baohua and Zeng Ling [3] analyzed the meaning and technique of using the stored procedures and presented other approaches to improve the performance of RDBS, such as the using stored procedures to improve the running performance of an application.

Nattha Y. [7] proposed an approach and tool to detect flaws in software design model by using graph diagram and tree diagram. That tool generated both graph diagram and tree diagram of bad smells and software design model and compared the two diagrams to detect bad smells. Thisana P. [8] proposed a method for detecting bad smells by using object-oriented software metrics to measure the values of bad smells to identify their types. Gregory T. et al. [9] presented a new technique to eliminate SQL injection by capturing the parse tree structure at runtime and parse tree structure after inserting user-supplied input. The SQL injection validation is compared to both parse trees. Yuki Ito et al. [10] proposed a method for detecting the factors behind bad smells by using declarative meta programming (DMP). The detection is performed by comparing bad smells with the abstract syntax tree (AST) of the target source code and then representing the comparison using Prolog.

Palomba F. [11] used TACO (Textual Analysis for Code smell detectiOn) to detect Long Method smell, which is able to detect between 50% and 77% of the smell instances with a precision ranging between 63% and 67%. Walter B. and Martenka P. [12] investigated two Large Class to find related patterns by using data mining. The patterns can be applied to identify bad smells.

Bad smells in this research are from Knowledge Xpert for PL/SQL which is a comprehensive Windows-based technical resource that covers the entire lifecycle of PL/SQL programming. The program provide thousands of topics, background information, best practices, and examples. In addition, the selected six PL/SQL bad smells used in this research are as follows: 1. Business Logic in Exception Sections (important parts of business logic source code should not be in exception sections), 2. Converting Cursor Loops with DML (using cursor loops with DML effect to query speed), 3. Move Initialize and Clean Up Logic (initialize and clean up should be separated and executed through the modules), 4. Replace Literals (using characters directly in commands should be invoked via a constant variable or function), 5. Range Values in Numeric FOR Loop (null numeric value of loop operation causes error), 6. Using SUBTYPES and Anchored Types (when the variable length is changed in data table, it does not affect the code).

3. The Approach for Detecting Bad Smells of Stored Procedures

The overview of bad smells detection process is depicted in Fig. 1. The three major phases consists of Creation Bad Smells Tree Diagram, Bad Smells Detection, and Refactoring and Measurement.

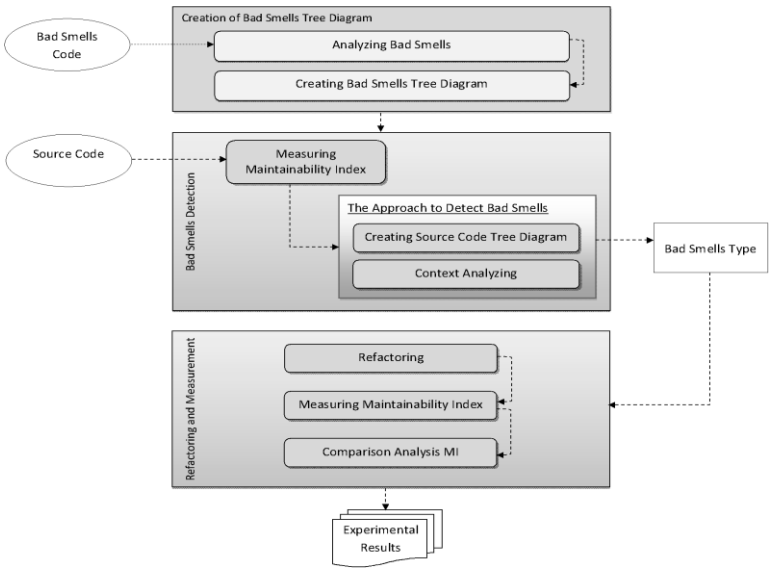

Fig. 1: Overview of the approach for detecting bad smells of stored procedures

3.1. Creation of Bad Smells Tree Diagram

The first phase analyzes six stored procedure bad smells for generating bad smells tree diagram, employing two processes: Analyzing Bad Smells and Creating Bad Smells Tree Diagram. **3.1.1.Analyzing Bad Smells** is a process of analyzing six stored procedures bad smells written in PL/SQL language structure designed as block, declaration section block, execution section block, and exception section block. The results of this process are the characteristics of six bad smells, such as the sequence structure of the statement in section block, and the node in bad smells tree structure. **3.1.2.Creating Bad Smells Tree Diagram** is a process of generating bad smells tree diagram from the mentioned characteristics. Fig. 2 and Fig. 3 are examples of bad smells tree diagram.

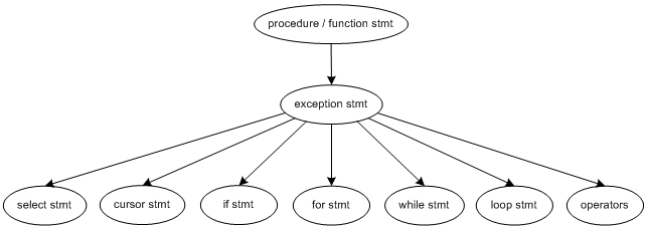

Fig. 2: Business Logic in Exception Sections

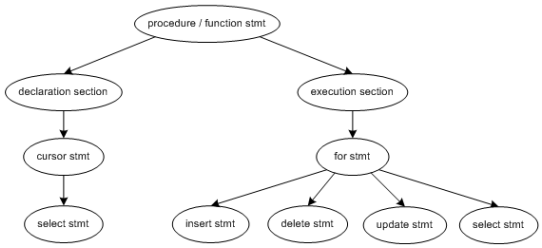

Fig. 3: Converting Cursor Loops with DML

3.2. Bad Smells Detection

This phase measures maintainability index before refactoring and detects bad smells of source code which consists of Measuring Maintainability Index and An Approach to Detect Bad Smells. In addition, it explains the bad smells detection algorithm and calculation of maintainability index.

3.2.1.Measuring Maintainability Index is a process of measuring MI (maintainability index) of source code using program ClearSQL to obtain MI values before refactoring. MI calculations are as follows:

MI (Maintainability Index) = 171 - 5.2 * ln(V) - 0.23 * (G) - 16.2 * ln(LOC)

V = Halstead Volume

G = Cyclomatic Complexity

LOC = count of Source Lines Of Code (SLOC)

CM = Percent of lines of Comment (optional)

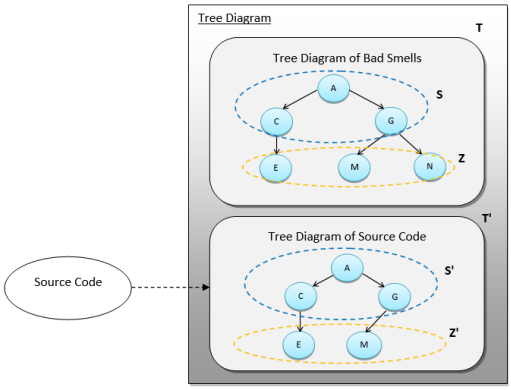

Fig. 4: Example Tree Diagram

S and S' are all nodes from Level 1 to Level N-1

(1). ∀ S' ∈ T' = ∀ S ∈ T ← Level 1, 2, ... , N-1

Example. {A, C, G} ∈ S' = {A, C, G} ∈ S ← Level 1 and 2

Therefore S' is equal to S

Definition 2 (Level N)

Z and Z' are all nodes from Level N

(2). Z' ∈ T'; Z ∈ T | {Z' ∈ T' } ⊂ {Z ∈ T} ← Level N

Example. {E, M} ∈ Z' ; {E, M, N} ∈ Z | {E, M} ⊂ {E, M, N} ← Level 3

Therefore Z' is a subset of Z

3.2.2.The Approach to Detect Bad Smells is a proposed algorithm to identify bad smells type of source code which consists of two processes: Creating Source Code Tree Diagram and Context Analyzing.

- **Creating Source Code Tree Diagram** is a process of generating tree diagram of source code (Fig. 4) to better understand the algorithm. The brief algorithm is all nodes, except leaf node required the same structure (the same members of set) as the bad smells tree diagram. The leaf node is a set or a subset of bad smells tree diagram. The BFS theory is used in searching tree diagram for this research.

Definition 1 (Level 1 to Level N-1)

The algorithm for detecting bad smells uses the child node set and the tree level as shown in Fig. 5 and Fig. 6. The algorithm is divided into two steps as follows:

First, verification of the node level in T', if "Level < N and Level ≠ N-1" find the child node set and compare it to the child node set of T. The comparing condition is "All Child of T'_i ≠ All Child of T_i", if it is true then the T' structure is not the same as T, then this algorithm RETURN False, illustrate that the source code tree diagram does not have bad smells, but if it is not true then the T' structure is the same as T then use the loop to the next level. The example from Fig 4 shown the values of N is 3, N-1 is 2 and Level 1 of T' is node A, where A = {C, G} and Level 1 of T is A, where A = {C, G}. If the compared child node {C, G} to {C, G} is equal then used loop to the next level.

Second, verification of the node level in T', if "Level < N and Level = N-1" find the child node set and compare it to the child node set of T. The comparing condition is "All Child of T'_i ⊂ All Child of T_i", if it is true then the T' structure is the subset of T, then this algorithm RETURN True, illustrate that the source code tree diagram has bad smells**,** but if it is not true then the T' structure is not the subset of T, then this algorithm RETURN False, illustrate that the source code tree diagram does not have bad smells. The example from Fig 4, the Level 2 of T' is C and G, where C = {E}, G = {M} and Level 2 of T is C and G, where C = {E}, G = {M, N} The compared child node {E} is a subset of {E} and {M} is a subset of {M, N}, therefore RETURN True, it identify the source code having bad smells.

This algorithm does not require the level n or leaf node of T' to have the same structure with T, which is just a subset, however it can identify bad smells, but the levels of 1 to n-1 require the same structure.

***Algorithm* Bad Smells Detection**

```
1   FUNCTION Bad_Smells_Detection(T,T' tree-diagram) RETURN Boolean IS ;
2     N  Number; ← All Level of T'
3     i Number; ← i is node id
4   BEGIN
5     FOR i IN T' LOOP
6       IF Level of  Node T'i < N THEN
7             Get All Child of T'i ;
8             IF Level of  Node T'i = N-1 THEN ← N-1 is level before the leaf node level
9                   IF All Child of T'i ⊂ All Child of Ti THEN
10                        RETURN True; ← therefore T' is bad smells tree diagram.
11                  ELSIF All Child of T'i ⊄ All Child of Ti THEN
12                    RETURN False; ← therefore T' is not bad smells tree diagram.
13                  END IF;
14            ELSIF Level of  Node T'i ≠ N-1 THEN
15                  IF All Child of T'i ≠ All Child of Ti THEN
16                    RETURN False; ← therefore T' is not bad smells tree diagram.
17                  END IF;
18            END IF;
19      END IF;
20    END LOOP;
21  END Bad_Smells_Detection;
```

Fig. 5: The Algorithm of Bad Smells Detection

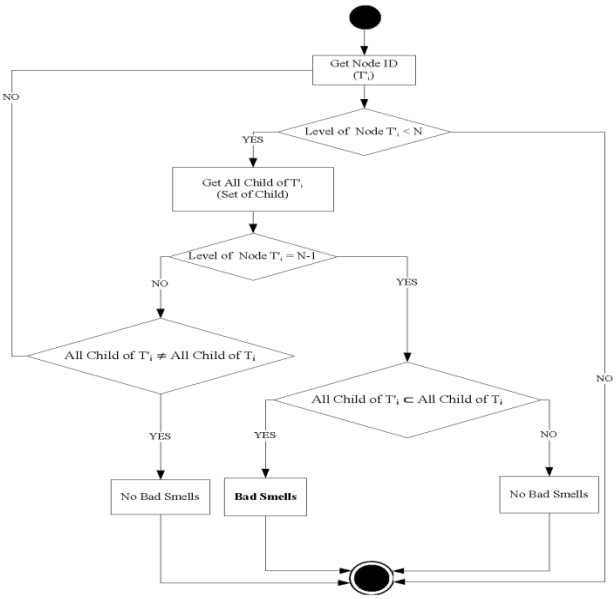

Fig. 6: Workflow of Algorithm

▪ **Context Analyzing** is extraction using textual analysis techniques to create the condition in increasing the accuracy to detect bad smells. The example of the context analysis for "Business Logic in Exception Sections" are the verification in the exception section and LOC <= 20.

3.3. Refactoring and Measurement

This final phase describes refactoring and measurements consisting of three processes: Refactoring, Measuring Maintainability Index, and Comparison Analysis Maintainability Index.

3.3.1.Refactoring is to refactor the bad smells source manually following the suggestions and the refactoring steps from PL/SQL Knowledge Xpert.

3.3.2.Measuring Maintainability Index is measuring source code after refactoring. The calculation of measurements in 3.2.1. uses ClearSQL program to obtain MI values.

3.3.3.Comparison Analysis Maintainability Index is comparing the MI values before and after refactoring to evaluate the efficiency of the proposed approach, where MI(1) is the measurement MI of source code before refactoring, and MI(2) is the measurement MI of source code after refactoring. The higher values of MI(1) than MI(2) revealed a positive performance due to the reduction of MI in the source code. The values of MI(1) equal MI(2) showed unaltered performance due to constant MI in the source code. The less values of MI(1) than MI(2) indicated a negative performance due to increased MI in the source code.

MI(1) > MI(2) is positive, MI(1) = MI(2) is unaltered, MI(1) < MI(2) is negative

4. Case Study

This section presents an example for detecting CCL(Converting Cursor Loops with DML) which affected execution speed of repetitive PL/SQL statements as illustrated in Fig. 7.

```
1  CREATE OR REPLACE PROCEDURE upd_for_dept1 (
2     dept_in IN employee.department_id%TYPE
3    ,newsal IN employee.salary%TYPE) IS
4     CURSOR emp_cur IS
5        SELECT employee_id, salary, hire_date
6          FROM employee
7         WHERE department_id = dept_in;
8  BEGIN
9     FOR rec IN emp_cur
10    LOOP
11       INSERT INTO employee_history (employee_id, salary, hire_date)
12            VALUES (rec.employee_id, rec.salary, rec.hiredate);
13       UPDATE employee
14            SET salary = newsal,
15                hire_date = rec.hiredate
16            WHERE employee_id = rec.employee_id;
17    END LOOP;
18 END upd_for_dept1;
```

Fig. 7: PL/SQL Source Code

The first phase is the Creation Bad Smells Tree Diagram. After this, CCL tree diagram is generated as shown in Fig. 8. ID represent node ID and PID represent Parent ID. The second phase is Bad Smells Detection which measures the source code before refactoring to obtain the MI(1) and create tree diagram of source code (Fig. 9). Context analysis condition of CCL bad smell to limit in searching source code in generating tree diagram was verified in declaration and exception section.

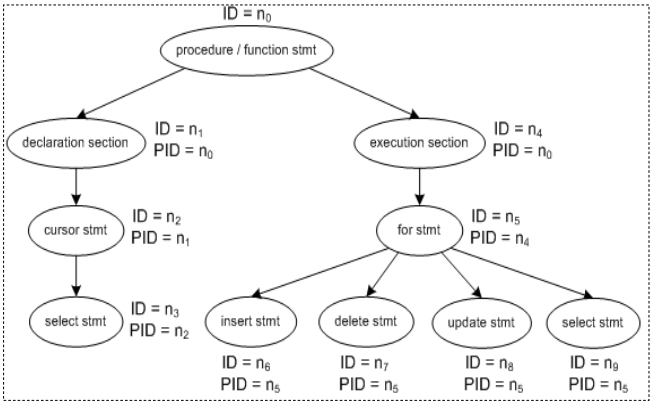

Fig. 8: Tree Diagram of CCL

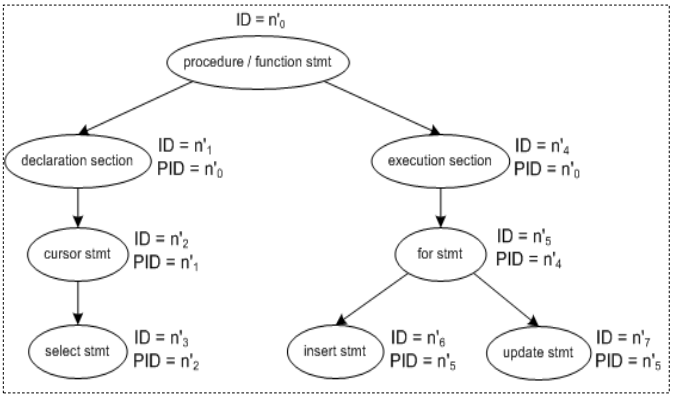

Fig. 9: Tree Diagram from source code

Then compare the structure using the proposed algorithm to identify the bad smells using members of child node set as show below is from tree diagrams (Fig.8 & 9).

$n_0 = \{n_1, n_4\}$ = {declaration section, execution section} and $n'_0 = \{n'_1, n'_4\}$ = {declaration section, execution section}
$n_1 = \{n_2\}$ = {cursor stmt} and $n'_1 = \{n'_2\}$ = {cursor stmt}
$n_2 = \{n_3\}$ = {select stmt} and $n'_2 = \{n'_3\}$ = {select stmt}
$n_4 = \{n_5\}$ = {for stmt} and $n'_4 = \{n'_5\}$ = {for stmt}
$n_5 = \{n_6, n_7, n_8, n_9\}$ = {insert stmt, delete stmt, update stmt, select stmt} and $n'_5 = \{n'_6, n'_7\}$ = {insert stmt, update stmt}

The results are $n'_0 = n_0$, $n'_1 = n_1$, $n'_4 = n_4$ and $n'_2 \subset n_2$, $n'_5 \subset n_5$ which is according to bad smells detection algorithm. Therefore, the detecting process identified CCL bad smell in the source code. The final phase is Refactoring and Measurement of source code according to steps of PL/SQL Knowledge Xpert to obtain the MI(2). The values in this case study of MI(1) is 102 and MI(2) is 93. The reduced numbers showed increased efficiency of source code. This research experimented the source code from PL/SQL Knowledge Xpert and the results are shown in Table I.

Table I: Maintainability Index Results

Bad Smells	MI(1)	MI(2)	MI(1)>MI(2)	MI(1)=MI(2)	% of Change
Business Logic in Exception Sections	58	54	●		6.90
Converting Cursor Loops with DML	102	93	●		8.82
Move Initialize and Clean Up Logic	88	84	●		4.55
Replace Literals	121	121		●	0.00
Range Values in Numeric FOR Loop	125	114	●		8.80
Using SUBTYPES and Anchored Types	137	137		●	0.00

5. Conclusions

This research proposes Tree Diagram and Context Analysis approach to detect bad smells of stored procedures focusing on six bad smells: Business Logic in Exception Sections, Converting Cursor Loops with DML, Move Initialize and Clean Up Logic, Replace Literals, Range Values in Numeric FOR Loop, and Using SUBTYPES and Anchored Types. The experiment was according to the algorithm of the approach to detect bad smells. A case study using CCL bad smell code of PL/SQL Knowledge Xpert was performed through the process of detection in order to obtain the measurement maintainability index to compare and evaluate the proposed approach. The results of the experiment are presented in Table I which show mainly positive measures of performance. The main contribution of this research is the improvement of source code maintainability, increase in speed execution, and reduction of time for comprehension and modification or extensibility.

The emphasis of this research is to increase the performance of stored procedures using the algorithm and the tree diagram and context analysis approach which could be applied to other various language programs for quality improvement.

6. References

[1] H.-S. Lee and K.-G. Doh, "Tree-pattern-based duplicate code detection," presented at the Proceedings of the ACM first international workshop on Data-intensive software management and mining, Hong Kong, China, 2009.

[2] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring," in International Conference on Software Maintenance, 2002. Proceedings., 2002, pp. 576-585.

[3] B. Tan and L. Zeng, "A performance optimization based on stored procedure in RDBS project," in 2010 International Conference on Computer and Communication Technologies in Agriculture Engineering, 2010, pp. 594-597.

[4] M. Habringer, M. Moser, and J. Pichler, "Reverse Engineering PL/SQL Legacy Code: An Experience Report," in 2014 IEEE International Conference on Software Maintenance and Evolution, 2014, pp. 553-556.

[5] Dimas C. Nascimento, Carlos Eduardo Pires, and T. Massoni, "PL/SQL Advisor: a Static Analysis-based Tool to Suggest Improvements for Stored Procedures," in 9th Brazilian, Symposium on Information Systems (SBSI'13), João Pessoa, Brazil, pp. 343-355, 2013.

[6] M. Alt, iota, iota, #351, iota, H. S, et al., "Automated procedure clustering for reverse engineering PL/SQL programs," presented at the Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, 2016.

[7] N. Yaowarattanaprasert, "Design and Development of an Approach for detecting Flaws in Software Design Model Using Graph Diagram and Tree Diagram," Master of Science Program in Software Engineer, Computer Engineer, Chulalongkorn University, 2013.

[8] T. Pienlert and P. Muenchaisri, "Bad-Smell Detection For Refactoring Using Object-Oriented SoftwareMetrics," International Conference on Computer Science, Software Engineering, InformationTechnology, e-Business, and Applications ,CSITeA,Cairo, Eypt, pp. 15-47, December 2004.

[9] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using parse tree validation to prevent SQL injection attacks," presented at the Proceedings of the 5th international workshop on Software engineering and middleware, Lisbon, Portugal, 2005.

[10] Y. Ito, A. Hazeyama, Y. Morimoto, H. Kaminaga, S. Nakamura, and Y. Miyadera, "A Method for Detecting Bad Smells and ITS Application to Software Engineering Education," in 2014 IIAI 3rd International Conference on Advanced Applied Informatics, 2014, pp. 670-675.

[11] F. Palomba, "Textual analysis for code smell detection," presented at the Proceedings of the 37th International Conference on Software Engineering - Volume 2, Florence, Italy, 2015.

[12] B. Walter and P. Martenka, "Looking for Patterns in Code Bad Smells Relations," in 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011, pp. 465-466.