International Journal on Advanced Science Engineering Information Technology

Vol.10 (2020) No. 3
ISSN: 2088-5334

Zsmell – Code Smell Detection for Open Source Software

Aziz Nanthaamornphong[a,1], Tanawat Saeang[a], Panyaprach Tularak[a]

[a] College of Computing, Prince of Songkla University, Phuket Campus, Kathu, 83120, Thailand
E-mail: [1]aziz.n@phuket.psu.ac.th

Abstract— **Today, open-source software (OSS) is used in various applications. It has played a vital role in information systems of many user groups such as commercials, research, education, public health, and tourism. It is also a source of additional knowledge for collaborators because this type of software is easily accessible through websites that provide management of version control services such as GitHub. However, a recent study shows an increasing trend in the existence of code smells. In OSS, there is a growing number of code smells that cause software errors. Having a code smell in software is a serious issue since it impacts the software in terms of deployment, maintenance as well as user confidence toward the software. Finding code smells in the early stages of software development would provide for better software maintenance and reliability; thus, researchers invented the Zsmell software system that helps search for code smells in the source code saved in GitHub. Developed systems display data related to code smells in each source code version that was modified by collaborators. Thus, the developers will be able to employ the proper refactoring method, which is a change in the internal structure of software without changing the original functionality of the software. We believe that this system will enable open source collaborators to improve the quality of their OSS, especially on code smell reduction and the understanding of various types of code smell commonly found in OSS projects.**

Keywords— **software engineering; open source software; software maintenance; code smell; software quality.**

I. Introduction

At present, open-source software (OSS) has been developed to meet a variety of needs and is likely to be applied in many areas. In the software development process, OSS is co-developed by collaborators with diverse experiences from around the world. OSS solves problems in different ways, resulting in the successful operation of OSS [1]–[5]. However, restrictions on software maintenance may exist, as most collaborators focus on software mechanics, resulting in the development of source code such as difficulty of source code comprehension, increasingly complex source code, or poor system design. Some of these problems are caused by code smells, which could cause problems during operation or maintenance, such as time and high development cost [6]–[9].

The existence of code smells in software is a severe issue because it impacts the software in terms of deployment, maintenance, and user confidence towards the software. In software engineering, there are several ways to address code smells. One of the approaches is refactoring, i.e., changing the internal structure of the software without altering its original functionality. The purpose of refactoring is to make the software easy to understand, enhancing software maintenance. Before refactoring, it is necessary to identify which source code will be refactored. Generally, the developers perform the refactoring to fix the section with code smells, which is a part of the code that is likely to cause errors or bugs due to poor coding or mistakes committed by the developer. For instance, if the duplicated code (code clone) exists in the module, the developers will employ some refactoring methods (e.g., extract method, pull up method) to minimize such duplicated code.

Based on these problems, we propose Zsmell, which helps an OSS collaborator team find code smells in OSS by working with the GitHub system that is a famous online software version management system from OSS collaborators around the world [10]. Besides, the GitHub API service provides a variety of data services available in the system, such as user data, and OSS data systems. Thus, there are advantages from using these GitHub APIs to develop the Zsmell system, which can detect code smells in OSS based on software metrics. The system can display code smells in source code, the statistical data on the number of code smells generated by the collaborators within a team, and code smell locations for software collaborators.

The code smell information obtained from the proposed system would help developers to improve source code quality, reduce future inconsistencies, and perform software maintenance tasks. Additionally, software engineering researchers better understand the cause of code smells and bugs in OSS projects.

The remainder of this paper is organized as follows. Section II provides an overview of related work. Section III describes the Zsmell system and provides system evaluation. Finally, conclusions are drawn, and future work is presented in Section IV.

II. Material and Method

This section explains the details of related literature and technology. Also, the Zsmell system is described.

A. Related Literature

A code smell is the feature of software code that can cause software malfunction or quality degradation, which increases the risk of future problems [11]–[13]. In software engineering practices, software collaborators should detect code smells before they become defects, which can cause enormous losses. In the past, researchers had specified the number of code smell types; however, for the current version of Zsmell, there are seven types, including the following:

- A large class is a class consisting of many variables, methods, objects, and functions [14].
- Long method is a method that can be understood or modified with difficulty [15].
- Lazy class is a low function class that consumes excessive memory space [14].
- Long Parameter List is a code that has excessively sized parameters, so it difficult to understand [16].
- Cyclomatic complexity is a code consisting of several conditions that make it difficult to modify or edit [17].
- Multiple Returns is a code that uses a command to send a value back unnecessarily, so it is difficult to understand the code [18].
- Message Chains is an object that requests another object that objects request yet another one, and so on. Any changes in these relationships require modifying the object [19].

Yamashita et al. [20] detected whether researchers and collaborators are concerned with or aware of code smells. In the past, there has been much research on how to prevent and remove code smells, but research is insufficient, and code smells still cause damage. Researchers completed an online survey of 85 professional collaborators. The results of the research show that only 4% of collaborators understand code smells when most collaborators should be required to be versed in a tool that helps to identify code smells to show real-time results.

A study related to code smells by Menzies et al. investigated whether researchers and developers were concerned with or value the importance of code smells differently [21]. Many studies have addressed this issue, yet code smells continue to cause adverse effects. Yoshida et al. studied the relationship between refactoring and code smells to explore which refactoring model is used by developers to fix their code smells [22]. The study aimed to support code writing by finding a suitable refactoring model. Silva et al. examined the motivation for developers to employ refactoring with the objective of exploring the real reason behind refactoring decisions [23].

For OSS projects, several studies have focused on processes and procedures for maintaining OSS [24]–[26]. Nevertheless, from the survey of related literature, there are no tools for finding code smells in OSS projects that can work with GitHub.

B. Related Technology

1) GitHub API: GitHub API provides data services from the GitHub website via the HTTPS protocol and is accessible from https://api.github.com. All data are exported in JSON (JavaScript Object Notation) format. GitHub API technology is used to retrieve data from GitHub, for example, to include website user data, user OSS data, and OSS source code revision.

2) Java Parser: Java Parser is a library for analyzing source code in Java to be an abstract syntax tree for data structure in code smells searching.

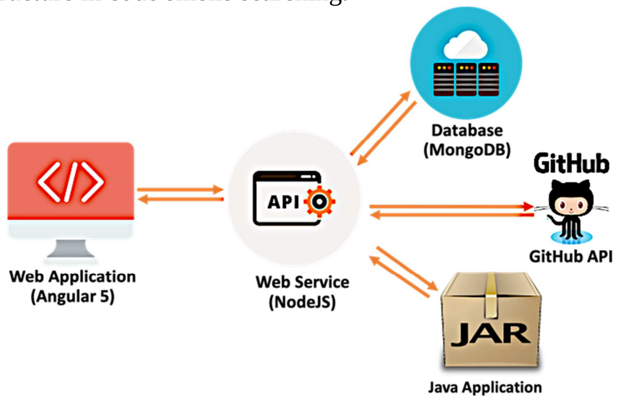


Fig. 1 Overview of the Zsmell System

To search code smells existed in the project, the program has to read the data structure of the given source code. If any structure matches the conditions specified in the program, the program collects code smell information into the database.

C. Zsmell System

The main functions of the Zsmell system include:

- Finding code smells existed in OSS projects stored in GitHub;
- Using the search results to summarize the number of code smells;
- Calculating statistics on code smells generated by collaborators for each revision;
- Visualizing the evolution of code smells; and
- Generating the reports

The overview of the process is shown in Figure 1. The collaborators must log in with a GitHub account to authorize and to select the user OSS to search for code smells via the web application. The developed Zsmell system can detect code smells only projects implemented by using the Java programming language. After that, the system can save the code smell data in the MongoDB database via web services and display data in graphical form via a website. Details for each part are presented in the sections below.

1) Web Application: We have developed a web application with an Angular Framework for data processing on web browsers. Users can select OSS software for searching code smells via a web browser. Users can view and summarize the results of code smells through the display of statistical data in graphical format for software quality improvement and empirical evidence to encourage collaborators to be more concerned with the issue of code smells.

2) Web Services: ExpressJS technology is used in this work for web services development. ExpressJS is a framework for NodeJS that has main functions to exchange data as follows.

- GitHub API - web services are communication intermediaries between the web application and the GitHub API, which are responsible for identifying the user and accessing their project data from the GitHub service. By default, all requests to https://api.github.com receive the v3 version of the REST API. All API access is over HTTPS. All data is sent and received as JSON. The system authenticates through GitHub API by using OAuth2.
- Java Application - web services send all project data that the user selects to the Java application to find code smells in the project and send them to the web services.
- MongoDB – the processor of the data when the web service receives the data before saving the code smells to the database.
- Web Application - web services provide various data services for visualizing the results on the application website.

3) Java application: Java applications are primarily responsible for identifying code smells from OSS derived from web services. In each project submitted, one project can be divided into two types.

- The Master Project is the current state of the source code, which is all code in the project that is on the main branch. More specifically, the initial project in GitHub is a master project, which can have branches in the future.
- The Commit Project is a source code revision of each collaborator in the team with details of the person who recorded the changes, including which files and lines were edited.

When both projects are entirely found, the data of code smells are gathered by identifying the project files and lines of code that have code smells. This Java application can find the following previously described seven types of code smells: Large class, Long Method, Lazy class, Long Parameter List, Cyclomatic Complexity, Multiple Returns, and Message Chains. To reduce the processing time, the application has been designed to utilize the multi-thread concepts in which each thread simultaneously detects each code smell type.

The system reads the submitted source code and all the source code for identifying the code smell format. If any part of the source code is found with the conditions shown in Table 1, the source code is the code smell and returned to the web services. The conditions shown in Table 1 are based on the definition of the code smell types that were previously described. However, the conditions in Table 1 are the only preliminary criterion for determining code smells. Users can edit these considerations of criterion via the web application. For example, the user can adjust the threshold of Large class by either increasing or decreasing the number of lines of code. However, the system does not allow the user to reduce the value for some code smell types, including Multiple Returns and Message Chains.

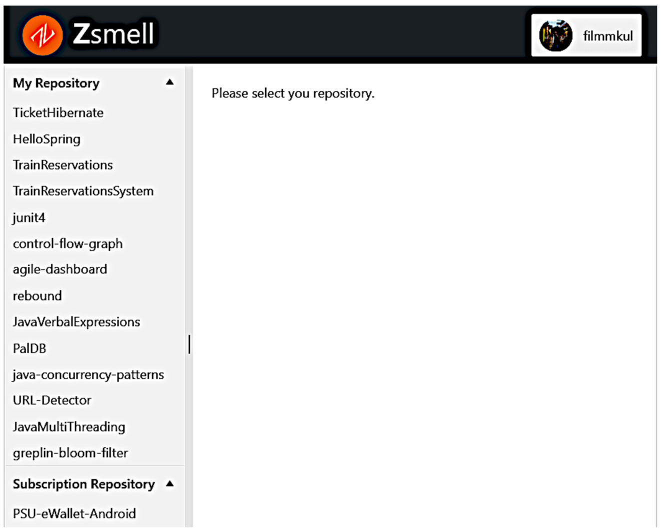
TABLE I
CODE SMELL TYPES AND CRITERION

Code Smell	Criterion
Long Method	• Number of lines of code without comments in the source file < 50 or • Algorithm complexity > 5
Large class	• Number of methods > 5 or • Number of lines of code without comments in the source file > 300
Lazy class	• Number of methods = 0 or • Number of lines of code without comments in the source file < 100 and the complexity of the class algorithm per number of methods < or = 2
Long Parameter List	The method has a few parameters > 4
Cyclomatic Complexity	Algorithm complexity, including the number of loops and control statements > 10
Multiple Returns	The number of methods that have the return command > 1
Message Chains	The number of calls in the source file to other methods > 2

III. RESULTS AND DISCUSSION

The method for using the proposed web application is described along with examples in this section.

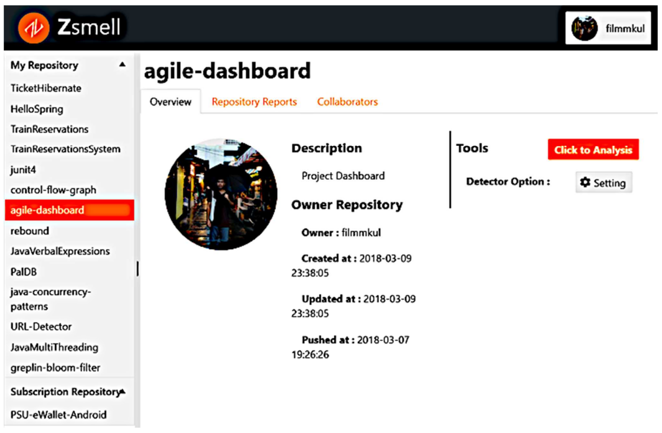
First, the user must have a GitHub user account that contains an OSS developed in Java. When the user account is ready, the user needs to log in via GitHub and accept permission to access the user data and projects from the Zsmell application in GitHub. After that, the user has been brought to the system homepage. The user can select his/her projects that need to analyze the code smell from the left menu bar, as shown in Figure 2. Only projects, which are under the “My Repository” menu can be analyzed to search for code smells.



Please select you repository.

Fig. 2 System Homepage

When a user has selected the projects from which to find code smells, the project details are shown in Figure 3. The user presses “Click to Analyze” to search for the code smells of the project. The user then must wait for the results. The waiting time depends on the size of the selected projects. After the analysis is completed, the system notifies the user through the browser. Users can view statistical results in graphical forms.


Zsmell
filmmkul

My Repository
TicketHibernate
HelloSpring
TrainReservations
TrainReservationsSystem
junit4
control-flow-graph
agile-dashboard
rebound
JavaVerbalExpressions
PalDB
java-concurrency-patterns
URL-Detector
JavaMultiThreading
greplin-bloom-filter
Subscription Repository
PSU-eWallet-Android

agile-dashboard
Overview | Repository Reports | Collaborators

Description
Project Dashboard
Owner Repository
Owner : filmmkul
Created at : 2018-03-09 23:38:05
Updated at : 2018-03-09 23:38:05
Pushed at : 2018-03-07 19:26:26

Tools
Click to Analysis
Detector Option : Setting

Fig. 3 Details of the Project Selected by the User

In terms of search results, the researchers developed the Zsmell system to display graphics in the application with the following details.

A. Repository Reports Menu

The repository reports menu is a summary menu of code smell search results, consisting of the following three sub-menus:

1) Repository Reports Menu: The graphically results of code smells in graph format are divided into the following three types:

- a graph that shows the total number of code smells by code smell types,
- a graph that shows code smells by collaborators, and
- a graph that shows the evolution of code smells in a project.

All the graphs are interactive, and many are pannable and zoomable. These charts are based on pure HTML5 technology.

Figure 4 shows the total number of code smells divided by type in a bar graph. The vertical axis is the number of code smells, and the horizontal axis is the code smell types found within the project.

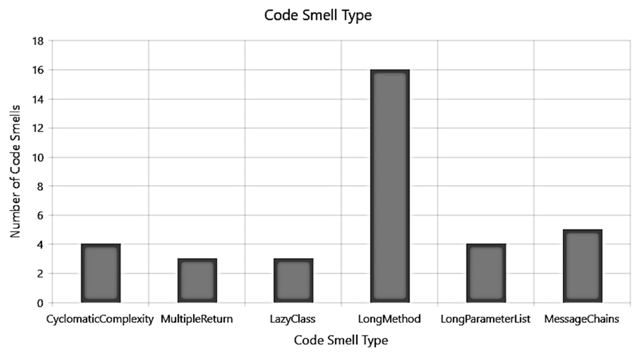


Fig. 4 Number of Code Smells Divided by Type

Figure 5 shows the total number of all code smells produced by software collaborators in a bar graph. The vertical axis is the number of code smells, and the horizontal axis is the collaborators' code in the project. Figure 6 shows the evolution of the code smells generated by each collaborator. The horizontal axis represents the Git commit IDs performed by the collaborator. Each Git commit ID is automatically generated to identify the commits uniquely. The vertical axis is the number of existing code smells.

2) Commits Menu: Displaying the amount of code smells in the menu by showing the number of code smells in each source code revision (Figure 7). Users can select to view each revision by choosing Git ID. The vertical axis shows the number of code smells, and the horizontal axis shows code smell types.

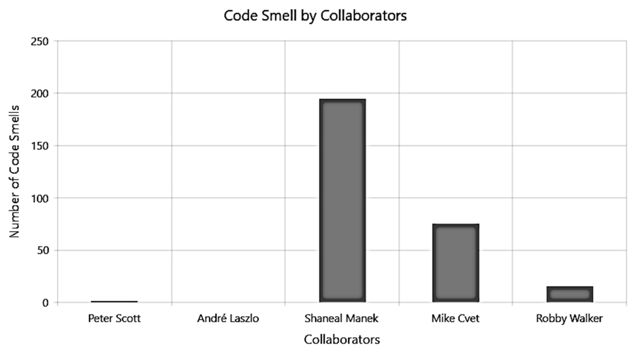


Fig. 5 Number of Code Smells Divided by Collaborators

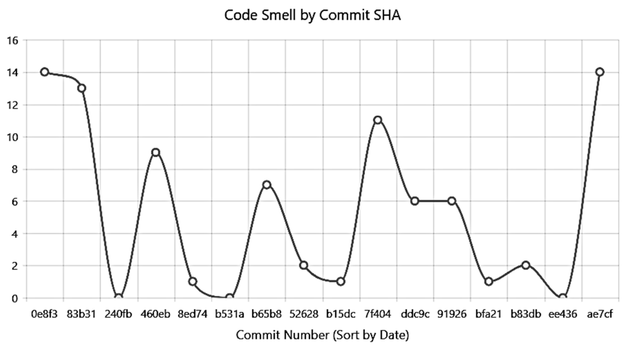


Fig. 6 Number of Code Smells Divided by Commit SHA

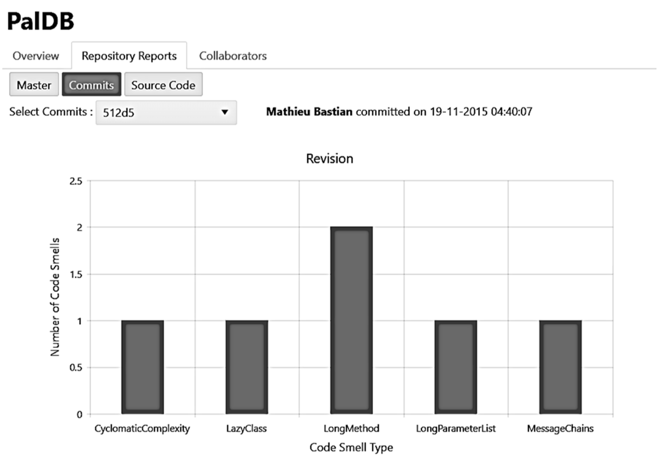

PalDB
Overview | Repository Reports | Collaborators
Master | Commits | Source Code
Select Commits : 512d5
Mathieu Bastian committed on 19-11-2015 04:40:07

Fig. 7 Number and Types of Code Smells in Commits

3) Source Code Menu: Figure 8 shows the relationship between the number of code smells and code smell types of a selected collaborator. The vertical axis indicates the number of code smells, and the horizontal axis shows code smell types that a collaborator revise.

B. *Repository Reports Menu*

Displaying the number of code smells in this menu shows some code smells by summarizing an overview from the collaborator revisions. Users can select the code of a collaborator who participates in the project and has revised source code. The code smell area is highlighted with a different color, as shown in Figure 9. For example, after the user selected the "ReaderImpl.java" and the code smell type as "Cyclomatic Complexity," the code where the algorithm complexity > 10 was highlighted with a different color. This feature helps the user identify the code smells without reading the code line by line.

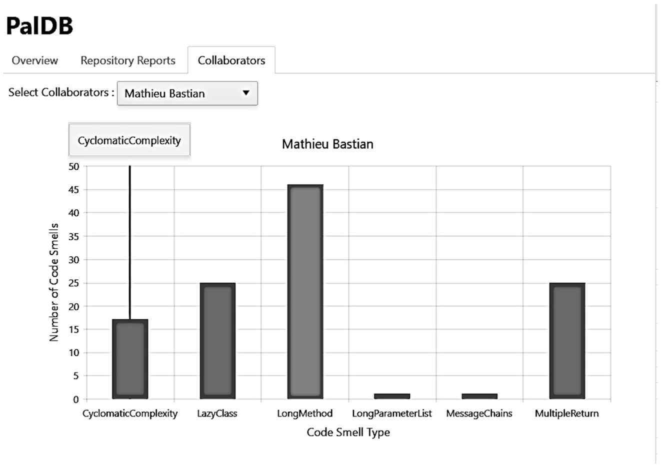
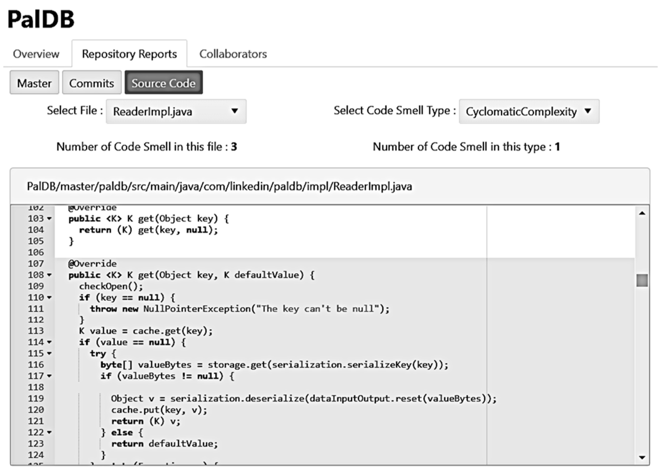


Fig. 8 Number and Types of Code Smells by Collaborators



Select File : ReaderImpl.java

Select Code Smell Type : CyclomaticComplexity

Number of Code Smell in this file : **3**

Number of Code Smell in this type : **1**

PalDB/master/paldb/src/main/java/com/linkedin/paldb/impl/ReaderImpl.java

```
@Override
public <K> K get(Object key) {
  return (K) get(key, null);
}

@Override
public <K> K get(Object key, K defaultValue) {
  checkOpen();
  if (key == null) {
    throw new NullPointerException("The key can't be null");
  }
  K value = cache.get(key);
  if (value == null) {
    try {
      byte[] valueBytes = storage.get(serialization.serializeKey(key));
      if (valueBytes != null) {

        Object v = serialization.deserialize(dataInputOutput.reset(valueBytes));
        cache.put(key, v);
        return (K) v;
      } else {
        return defaultValue;
      }
```

Fig. 9 Source Code that File Selected

TABLE II
OPEN SOURCE PROJECTS TESTED

OSS#	OSS Project	Number of classes	Number of Methods
1	Rebound	54	476
2	Java-concurrency-patterns	53	164
3	URL-Detector	25	225
4	JavaMultiThreading	42	104
5	Greplin-bloom-filter	14	118

C. *Evaluation*

We have empirically evaluated the system in terms of correctness and user satisfaction. The following subsections present the evaluation results.

TABLE III
TEST RESULTS

Code Smell	OSS#1	OSS#2	OSS#3	OSS#4	OSS#5
Long Method	7(7)	4(4)	21(21)	4(4)	16(16)
Large class	0(0)	0(0)	1(1)	0(0)	0(0)
Lazy class	30(30)	33(33)	5(5)	12(12)	4(4)
Long Parameter List	7(7)	0(0)	2(2)	0(0)	4(4)
Cyclomatic Complexity	9(9)	0(0)	10(10)	0(0)	4(4)
Multiple Returns	18(18)	2(2)	15(15)	0(0)	3(3)
Message Chain	20(20)	5(5)	0(0)	0(0)	5(5)

1) Correctness: To check how the system provides the correctness results, five GitHub OSS projects stored in GitHub were selected to find code smell types mentioned above. Those projects include:

- Rebound: A Java library that models spring dynamics and adds real-world physics.
- Java-concurrency-patterns: Concurrency Patterns and features found in Java, through multithreaded programming.
- URL-Detector: A Java library to detect and normalize URLs in text.
- JavaMultiThreading: Examples of Java MultiThreading concepts.
- Greplin-bloom-filter: A Bloom Filter implementation in Java, that optionally supports persistence and counting buckets.

Table 2 shows the OSS project details, including the number of classes and methods. The correctness test procedure consists of the following steps.

- Code smells were found manually or with Zsmell.
- The 2nd and 3rd authors searched for code smells in each type. When a bad code was found, the data have been saved, such as the type of code smells, file name, and line that represents the code smell. Each researcher separately performed a search.
- The results from the 2nd and 3rd authors were compared. If the results were inconsistent, the 2nd and 3rd authors discussed the results.
- Tests were conducted using Zsmell.
- The results of the two methods were compared to present the difference between such methods.

The results for code smells from both tests were compared to include the number of each code smell type, as shown in Table 3. In Table 3, each field of the table shows the following two numbers: the first is the number of code smells found by the researcher, and the numbers in parentheses are the number of code smells found by Zsmell. The table shows the accuracy of finding code smells. Zsmell could find all code smells according to the conditions listed in Table 3 (100%).

2) Satisfaction Results: We asked the participants, including 40 undergraduate students who enrolled in the software construction and maintenance class, to use Zsmell. We chose these students as participants because they were studying code smell and refactoring topics in the class. Thus, we believed that the Zsmell system would increase student understanding of code smells. Each participant had to use Zsmell to detect code smells that existed in the Rebound Project, and the time was limited to one hour. Once the time ended, the students completed an online questionnaire. The questionnaire consisted of 5 Likert-scale questions addressing the following topics:

- The ease of use of the software (e.g., it requires the fewest steps possible to accomplish what the user wants to do with it.)
- The satisfaction with messages which appear on the screen (the messages can be of the following types: notification, confirmation, warning, and error.)
- The satisfaction with instructions for commands or choice (the target task is completed with less effort).
- The satisfaction with the speed of the system (the response of the system meets the user's expectation).
- The satisfaction with the report (the system can generate sufficient reports).

To ensure that the survey questions were comprehensible and valid with respect to the objective, we conducted a pilot study to observe all stages of the survey process, including the administration of the questionnaire. The pilot study duplicated the final survey design on a small scale from beginning to end, including the data processing and analysis steps. The pilot study allowed us to see how well the questionnaire performs during all steps in the survey. We randomly selected 5% of the participants from the target lists. These participants were excluded from the subsequent major targets.

Finally, we received the survey responses from all participants. Table 4 presents the survey results. Overall, the participants were satisfied with the software; however, students provided the following comments:

- The system should have the capability to export the report into various formats (e.g., PDF, MS Excel, MS Word)
- Additional code smell types should be made available.
- The system should define each code smell type (e.g., help documentation).
- The system should detect more code smell types (e.g., Data Clumps, Code Clone).

3) Limitations: Based on the evaluation, the limitations of the proposed system can be divided into four-folds.

- Concerning the issue of OSS development experience, students may respond differently to questions than OSS developers working on OSS projects. Thus, the survey results may not show the real benefits of our proposed tool to the OSS developers. To survey with the OSS developers should help us better understand how Zsmell was adequately designed for developers.
- The system performance must be measured in terms of the quantitative method how the system utilizes the resources.
- The current version can only detect seven code smell types, which can be extended for more code smell types.
- The obtained evaluation results might be specific to the small projects that were used. More future evaluations, with larger projects, are needed to confirm the results further and draw more general conclusions.

TABLE IV
RESULTS OF USER SATISFACTION

Question	**Strong Agree (%)**	**Agree (%)**	**Neither Agree nor Disagree (%)**	**Disagree (%)**	**Strong Disagree (%)**
The software is easy to use	75	20	5	0	0
The messages which appear on screen are satisfied	80	15	5	0	0
The instructions for commands or choice are satisfied	85	10	5	0	0
In your opinion, the system returns provide the results quickly	70	17.5	12.5	0	0
The reports are satisfied	77.5	20	2.5	0	0

IV. CONCLUSIONS

Zsmell is a system that helps find code smells of OSS projects saved in GitHub. Zsmell is used to determine what types of code smells occurred within a project, specify whether the developers have abilities to improve code smells or not, and encourage collaborators to pay more attention to code smells. This proposed system is a part of creating software with higher quality and better ease of maintenance. In the future, the system will be improved by adding types of code smells to be found by the system, including to make the procession more efficient and faster. Additionally, the system is planned to be released in an open source system for interested users.

REFERENCES

[1] G. W. Hislop and H. J. C. Ellis, "Humanitarian Open Source Software in Computing Education," *Computer (Long. Beach. Calif).*, vol. 50, no. 10, pp. 98–101, 2017.

[2] D. L. Olson, B. Johansson, and R. A. De Carvalho, "Open source ERP business model framework," *Robot. Comput. Integr. Manuf.*, vol. 50, pp. 30–36, 2018.

[3] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "A Systematic Mapping Study of Software Development with GitHub," *IEEE Access*, vol. 5, pp. 7173–7192, 2017.

[4] E. Katsamakas and M. Xin, "Open source adoption strategy," *Electron. Commer. Res. Appl.*, vol. 36, p. 100872, 2019.

[5] A. S. Sohal, S. K. Gupta, and H. Singh, "Trust in Open Source Software Development Communities: A Comprehensive Analysis," *Int. J. Open Source Softw. Process.*, vol. 9, no. 4, pp. 1–19, 2018.

[6] H. Liu, B. Li, Y. Yang, W. Ma, and R. Jia, "Exploring the Impact of Code Smells on Fine-grained Structural Change-proneness," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 28, no. 1487–1516, Apr. 2018.

[7] A. Cairo, G. Carneiro, and M. Monteiro, "The Impact of Code Smells on Software Bugs: A Systematic Literature Review," *Information*, vol. 9, p. 273, Nov. 2018.

[8] T. Sharma and D. Spinellis, “A survey on software smells,” *J. Syst. Softw.*, vol. 138, pp. 158–173, 2018.

[9] M. Paixao, J. Krinke, D. Han, C. Ragkhitwetsagul, and M. Harman, “The Impact of Code Review on Architectural Changes,” *IEEE Trans. Softw. Eng.*, p. 1, 2019.

[10] C. Liu, D. Yang, X. Zhang, B. Ray, and M. M. Rahman, “Recommending GitHub Projects for Developer Onboarding,” *IEEE Access*, vol. 6, pp. 52082–52094, 2018.

[11] S. Singh and S. Kaur, “A systematic literature review: Refactoring for disclosing code smells in object-oriented software,” *Ain Shams Eng. J.*, vol. 9, no. 4, pp. 2129–2151, 2018.

[12] M. Tufano *et al.*, “When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away),” *IEEE Trans. Softw. Eng.*, vol. 43, no. 11, pp. 1063–1088, 2017.

[13] D. Taibi and V. Lenarduzzi, “On the Definition of Microservice Bad Smells,” *IEEE Softw.*, vol. 35, no. 3, pp. 56–62, 2018.

[14] J. Dexun, M. Peijun, S. Xiaohong, and W. Tiantian, “Detection and Refactoring of Bad Smell Caused by Large Scale,” *Int. J. Softw. Eng. Appl.*, vol. 4, no. 5, pp. 1–13, 2013.

[15] F. A. Fontana, P. Braione, and M. Zanoni, “Automatic detection of bad smells in code: An experimental assessment,” *J. Object Technol.*, vol. 11, no. 2, pp. 1–38, 2012.

[16] G. Vale and E. Figueiredo, “A Method to Derive Metric Thresholds for Software Product Lines,” in *Proceedings - 29th Brazilian Symposium on Software Engineering, SBES 2015*, 2015, pp. 110–119.

[17] S. Kaur and R. Maini, “Analysis of Various Software Metrics Used to Detect Bad Smells,” *Int. J. Eng. Sci.*, vol. 5, no. 6, pp. 14–20, 2016.

[18] S. McConnell, *Code complete*. Pearson Education, 2004.

[19] M. Chains, “Message Chains,” 2019. [Online]. Available: https://refactoring.guru/. [Accessed: 10-Jul-2019].

[20] A. Yamashita and L. Moonen, “Do developers care about code smells? An exploratory survey,” in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2013, pp. 242–251.

[21] T. Menzies, L. Williams, and T. Zimmermann, "Perspectives on data science for software engineering," Morgan Kaufmann, 2016.

[22] Y. Norihiro, T. Saika, E. Choi, A. Ouni, and K. Inoue, “Revisiting the Relationship Between Code Smells and Refactoring.” *In Proceedings of the 24th International Conference on Program Comprehension*, 2016, pp. 1–4.

[23] S. Danilo, N. Tsantalis, and M. Tulio Valente, “Why We Refactor? Confessions of Github Contributors,” *In Proceedings of the 24th ACM Sigsoft International Symposium on Foundations of Software Engineering*, 2016, pp. 858–70.

[24] S. Lee, H. Baek, and J. Jahng, “Governance strategies for open collaboration: Focusing on resource allocation in open source software development organizations,” *Int. J. Inf. Manage.*, vol. 37, no. 5, pp. 431–437, 2017.

[25] A. Adewumi, S. Misra, N. Omoregbe, B. Crawford, and R. Soto, “A systematic literature review of open source software quality assessment models.,” *Springerplus*, vol. 5, no. 1, p. 1936, 2016.

[26] O. Franco-Bedoya, D. Ameller, D. Costal, and X. Franch, “Open source software ecosystems: A Systematic mapping,” *Inf. Softw. Technol.*, vol. 91, pp. 160–185, 2017.