3 Dimensional Visualization of Code Smells

P A C Hasantha

198753G

Faculty of Information Technology

University of Moratuwa

2022

3 Dimensional Visualization of Code Smells

P A C Hasantha
198753G

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfillment of the requirements of the Honors Degree of Bachelor of Science in Information Technology.

July 2022

Declaration

We declare that this is our own work and has not been submitted to another degree or diploma at a university or other higher education institution. Information obtained from published or unpublished work by third parties is acknowledged in the text and a list of references is provided.

Name of Student	Signature of Student
P.A. Chathuranga Hasantha	.....
	Date: .....

Supervised by

Name of Supervisor	Signature of Supervisor
Chaman Wijesiriwardana	.....
	Date: .....

Acknowledgements

I would sincerely like to thank the my supervisor Dr.Chaman Wijesiriwardana, a senior lecturer at the Faculty of Information Technology for providing me with the information and continuous support that was given and passed down to me during the research. This research and thesis were possible only because of the continuous support and guidance of my supervisor.

Thanks for letting me make a more practical approach to the field of formal methods. I would also thank all the colleagues who gave great support to improve the results. Last, but not least, thanks to my parents for your endless support and love throughout my studies.

Abstract

Bad code smells are symptoms of design flaws in source code. Several tools and approaches have been proposed for detecting and visualizing code smells. To maintain the software quality, prioritizing the identification and removal of code smells are required. Identifying the code smells using visualization will helpful developers to understand and refactor the code. This study proposes a novel 3D metaphor to detect and visualize code smells by using a combination of the code city and island metaphor visualization techniques. Proposed model identifies and visualizes the code smell at different abstraction levels in a proper understandable aspect. This model evaluates by using several open source software projects and visualizing the detected code smells in abstraction levels such as classes, methods.

The proposed model will allow for more research into code smell visualization and it will keep better focus on the needs of developers.

Index Terms—code smells, code smells detection, software visualization, code smells visualization

Table of Contents

List of Figures

List of Tables

CHAPTER 1

1. INTRODUCTION

Bad code smells introduced by the bad design and implementation decisions. As a result of that software quality suffers [1]. Code smells that violate the code practice principles will affect the software, changeability, and comprehensibility then increase the maintenance cost [2]. Developers cannot avoid the code smell occurrence in software due to many reasons such as lack of experience, deadlines. Consequently, several approaches [1], developed tools have been found to find the bad smells and refactor the source code.

Manual detection of code smells is time and resource-consuming. Due to the nature of the knowledge involved, automatic detection gives many false positives [3]. In order to that there are additional difficulties belongs to bad detection, such as context-dependence, search space size, unclear definitions, and the problem of defining value of metric threshold [4].

Visualization allows users to observe information [5]. It enables the visualization of the characteristics concealed in the data. Many visualization tools are now available but most of their available facilities have not met the needs of developers. This will effect while improving and guaranteeing the evaluating the quality of software. Graphical approaches are already proposed for code smell visualization [6] with the limitation of being unable to incorporate the visualization with large software projects and it can evaluate very few from the defined code smells [7].

This report presents a novel model for code smell visualization that reveals code-smell evolutionary information in an easy to understandable manner. This proposed model is capable of providing a visualization for identify code smells in a meaningful manner in the source code.

CHAPTER 2

2. LITERATURE REVIEW

2.1 Overview of Code smell Detection

Today's software systems are increasingly large and complex, and many people collaborate in their development and maintenance. This makes it more and more difficult to program, understand and modify the software tasks, especially when working on the code of other people. Therefore, tools for supporting these tasks have become essential [8].

Eva and Leon [9] describe an automatic code smell detection approach and code smells visualization and they discuss about how a software inspection tool may be designed using this method. There is an illustration of feasibility of their approach including the development of jCOSMO. It is a prototype apply to code smells detection and visualization in JAVA source code.

Emerson Murphy [10] emphasizes the significance of scalability, expressivity, and context-sensitivity when displaying code smells, and also proposed a prototype model of code smells detection tool using above properties. Many software tools have been suggested to help developers to evaluating the source code quality. A code smell detector that uses an interactive ambient visualization to assist make programmers becoming aware of smells and making educated, confident refactoring decisions. This study contends that such tools have a place in the software developer’s toolkit [11].

Lanza and Marinescu [12] contend that manual code smell detection has several drawbacks such as time-consuming, not repeatable, not scalable and approach that detecting code smells by humans has not been properly explored yet [13]. Furthermore, detection bad smells in large systems with many packages, classes, methods is a very time and resource consuming and faulty operation [14] because bad smells cut across classes and methods, leaving much room for interpretation in their descriptions.

According to Eclipse project three versions, Li and Shatnawi explored the relationship between code smells and occurrence of class errors. The results showed that infected classes with the code smells shotgun surgery, god class, or god methods include a higher error occurrence in class than uninfected classes [15].

2.2 Survey on Code Smells Visualization

Karim & Houari*, [16] presented a detection approach that combines automatic pre-processing with visual data representation. It is a semi-automatic method and complementary to automated approaches to code smell detection, which require knowledge that unable to extracted from the code directly. The detection is viewed as an inspection activity that is aided by a visualization tool that shows large programs that include thousands of classes. It allows analyst to navigate different abstract code levels. In this approach, they deliberately target to detect bad smells that are difficult to find automatically with this method.

Although code visualization tools are increasingly being used to aid with code smell detection, their module structures are limited to methods, classes, and packages. Glauco de F.[17] used multiple aspects of visualizations to detect bad code smells. They gives four views and it include relevant properties such as package, class and method structure, inheritance layout, dependency graph, and dependencies weighted graph and examine how well visual views enable code smell detection.

Some studies advocated using software visualization tools and approaches to detect code smells. Dhambri et al [18], proposed a model with 3D software visualization to discover design flaws. It employs both qualitative and quantitative data based on metrics and structural data based on module interactions. There is an improvement need in performance variability to reduce this variability, this approach is a manual process and needs to combine with an automatic one in another direction in their investigations.

A software visualization system called sv3D use 3D metaphors to display source code and associated attributes. Only poly cylinders are represented by sv3D, which also

shows 4 edges and a uniform fill. Different textures and a variable number of edges will be available. For 3D renderings, efficiency is typically the limiting factor. The performance of the rendering and user interaction is reduced in the large software systems. Here, controlled user studies were not performed in order to more accurately evaluate the benefits and drawbacks of the sv3D [19].

An interactive ambient visualization novel smell detector implemented for programmers. This model is used to help programmers to identify the code smells and refactoring judgments [20]. This is discussed, tested this model with limited code smells.

Multiple views approach based on concern-driven software visualization tools to quickly identify code smells. To facilitate a concern-sensitive analysis based on various viewpoints, the techniques or approaches relies on interactive visual abstractions of source code [21]. Evaluation of this approach is made for small criteria and it is not sufficient for large projects. The visual analysis might be a more thorough and productive review of the proposed approach depending on the proper assignment of issues to source code.

Murphy-Hill and Black [22] present Stench Blossom, a unique code bad smells detector provides with interactive environmental visualization. It provides programmers with a high-level overview of the code smells in their source code and it aids in the understanding of the sources of those smells. This model evaluates only for 8 code smells, and also they use only feature envy code smell for the explanations. And also their subject is unfamiliar with the source code. Therefore the novel model needs to validate the results for different code smells and the code needs to be familiar with source code.

VISMELLS is an interactive visualization created by Isaac de Jesus Silva et al. [23] to assist developers in identifying Code Smells. According to the survey results, VISMELLS can help in the reveal of bad smells. VISMELLS also supports in detect Code Smells by visualizing values from software metrics.

Nabilah and Sunindyo [24] proposed a visualization technique in 2D space to examine the code smells evaluation. Rectangles are utilized to represent software versions in their proposed methodology, while colored sub-rectangles are used to represent packages, classes, and methods. In the presence of code smell, a particular character appears. This is evaluated only with Java OpenGL programming language, and it has limited code smell detectors used to evaluate this approach, it needs to integrate with more code smell detectors to detect more code smell and refactor.

Abdulkarim [25] presented a method that focuses on extracting code bad smells from several iterations of the same piece of software in order to visualize the evolution of bad code smells. The proposed approach aims to explain in detail how code smells are identified, classified, and displayed. They only focused on the few Code smell categories. And also they evaluate the approach with only one case study.

An approach to analyzing multivariate object-oriented software was presented by Haris Mumtaz [26] and the software metrics to identify outliers that can be associated with bad smells in the context of software quality. Because of this, the data elements are more easily recognized visually as bad code smells, which are also seen as outliers in the associated visualizations. The given detection guidelines must be followed to automated code smell detection, and users other than the authors have not yet tested this method.

Classes and packages are shown as the city metaphor and its districts, respectively. By reducing the amount of different sizes exhibited at once, they look at mapping methods for software metrics to visual objects and discover mapping approaches for an easily visualized system [27].

Software visualization adopted to visualize bad smells in the program. Hammad et al[28] proposed a visualization approach based on the initials of the names of bad smells and depicted classes as buildings and bad smells as avatars. These avatars are displayed on the buildings as cautionary signs. A framework is suggested to automatically examine code to find bad smells and provide the proposed visual model. The study of

proposed visualizations revealed that they shorten the amount of time needed for understanding bad code smells.

While inspecting large software systems, the island metaphor visualization technique was utilized to highlight the modular features of OSGi and built an interaction strategy to maintain user comfort. Utilizing this method, OSGi-based software systems are explored in virtual reality. For the feasibility of the method to assist in software comprehension tasks, an application must be made [29].

Evaluating the effect of media on the 3D software visualization effectiveness is the most important point. They experimented with 3D city visualization technique and it proved to be effective for software realization tasks [26]. They are now using this with a regular computer display, a virtual 3D environment, and a real 3 Dimensional printed object. They also wish to look at the effects of collaborative visualization tools like multiple touch tables and display in wall.

2.3 Summery

The scope and coverage of these code smell detection and visualization techniques evaluate that they identified a lack of code smells through their proposed models or tools. In addition, the above studies do not visualize the code smells in different abstract levels of the software project. As a result of that, it has an issue with understanding the visualization of code smell in large software projects. Previous studies used a lack of test case studies while evaluating the model or techniques and also these processes are not fully automatic.

CHAPTER 3

3. PROPOSED APPROCH

3.1 Introduction

This chapter outlined the proposed approach to addressing the problem of code smells visualization as well as the technology to be applied to solve the research problem. Code smells visualization is the main module of this research according to the collected data from sonarqube selected project. We present our approach by emphasizing input, output, process, end users, and features of the code smells visualizations solution.

The proposed approach is based on the two visualization techniques. There are several metaphors to visualize the source code.

3.1.1 Island Metaphor

In the island metaphor, the entire class in the package is represented as an island in the sea. [25]. Individual classes are represented by islands. This metaphor is an effective and workable visual metaphor, which maps all top hierarchies including classes, packages of a software architecture. It is focused on 3D visualization of module wise software that forms the basis of many large software systems. The volume of the island depends on the code lines included in the class.

3.1.2 City Metaphor

The city metaphor uses the analogy between programs and cities to represent programs, e.g., buildings represent types (i.e. methods) and cities represent classes. Researchers have introduced the city metaphor differently [27]. In the city metaphor, "squares" represent building types, methods. That is, buildings are drawn as parallelograms with square bases. The building base size reflects the number of attributes (NOA) of a given method and the height of the building reflects the number of methods (NOM). Therefore, the larger and taller the building, the higher the number of attributes and methods, respectively.

3.2 Proposed Approach

There are two metaphors, island and city metaphor used in this model. Figure 1, shows the first phase of the proposed code smell visualization model. It uses the Island metaphor view to visualizations.

The island metaphor expresses the aspect of disjointed entities [23]. It shows that more information was better in single view than in multiple views. The software system package is represented as an ocean including islands. An island clearly illustrate a class of software system, which makes them a successful model for representing software modules.

The circle depicted in the figure denotes the classes. The blue color surrounding is the sea and it represents the packages of the software project. Each different highlighted color (red color) set to point out the code smell classes. It evaluates the fire on the island when code smell is identified from the source code. Global variables pointed to the boats in the sea.

The input to this approach is the data set prepared from the software project analyze in the sonarqube. Next, JSON format data set upload to this model and generate the visualization into abstract levels. This objects visualization corresponds to the method, classes and inside method that has the identified bad smells.

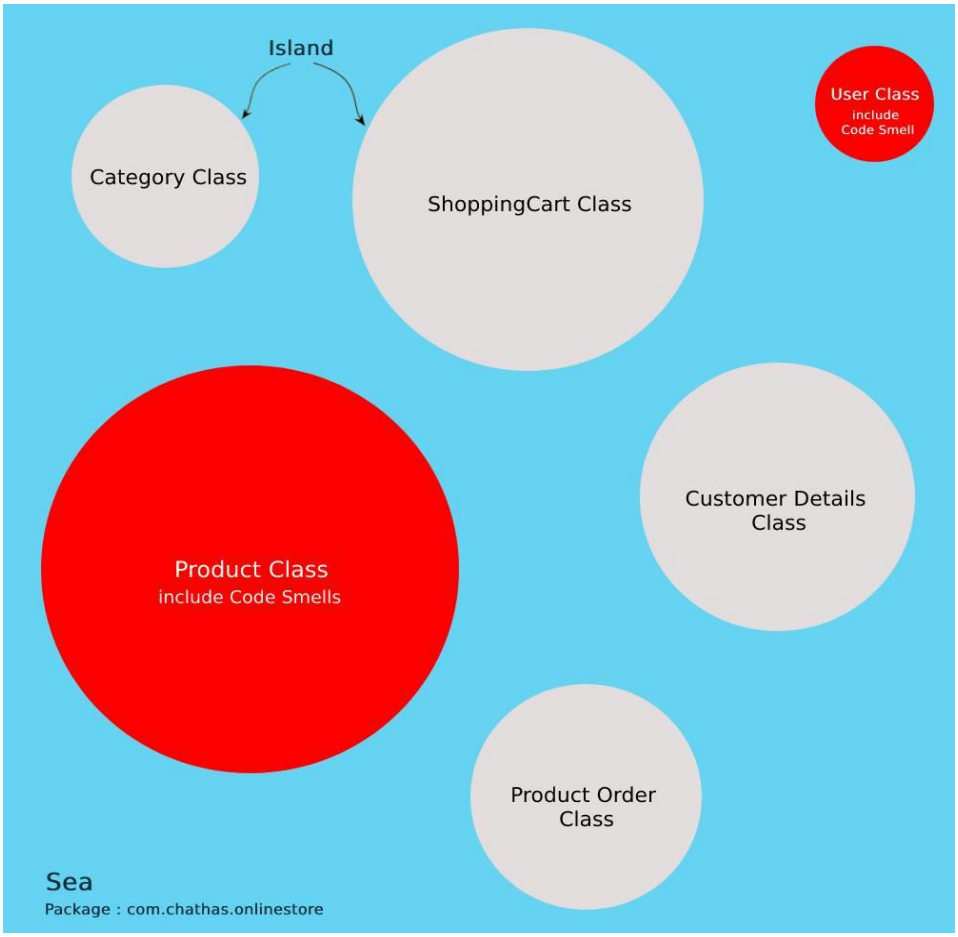


Figure 1 - Island Metaphor

In the proposed approach, the first step is island metaphor and to describe the island metaphor image, we use an online shopping store as an example. It contains Category class, Product class, ShoppingCart class etc. According to the above graphics objects of classes and packages are mapped with the below code snippets.

```
import com.chathas.onlinestore;   //package

public class Product{
        //product_class content
}
public class Category{
        //category_class content
}
```

```
public class CustomerDetails{
        //cuctomerdetails_class content
}
public class ShoppingCart{
        //shoppingcart_class content
}
public class ProductOrder{
        //productorder_class content
}
```

Figure 2. Shows the inside view of each island. This step concerns providing more insights regarding inside classes. The city metaphor visualization technique used to describe the methods and attributes. The blocks show the methods and humans are the variables in the class. The squares in the block depicted in this figure show the input parameters of the suitable method.

In addition, the number of the buildings and the building height are decided by the content inside the methods. However, the code smell in the city view is also highlighted in red as well. Thus, the proposed model produces a visual representation of the overall evolutionary characteristics of methods including package, class, and code smell.

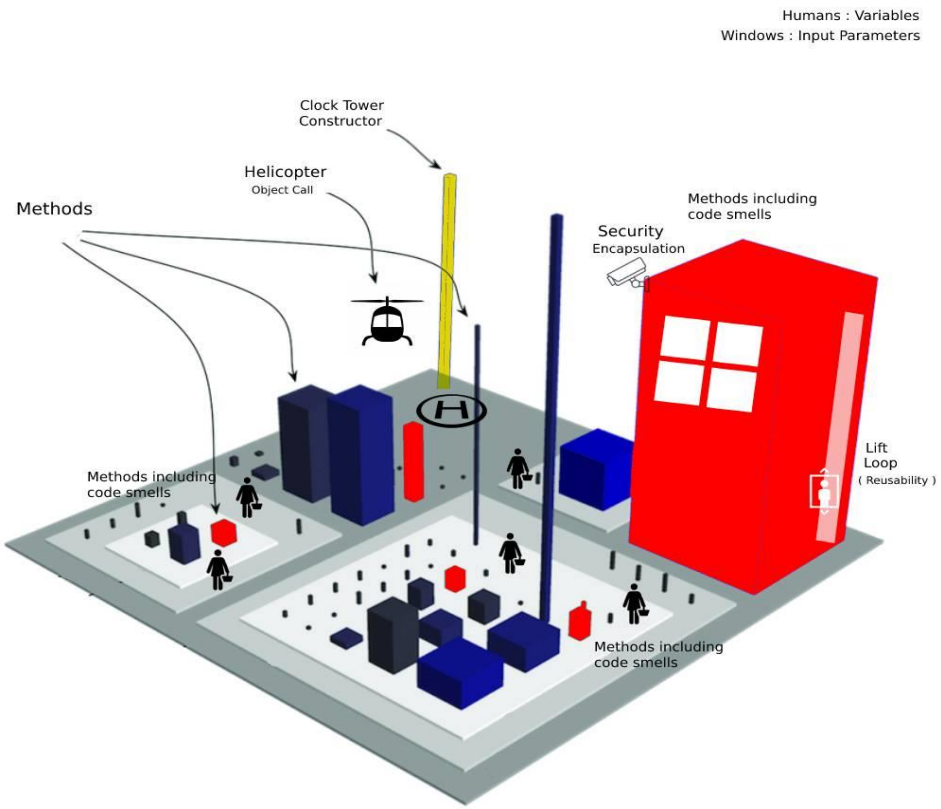


Figure 2 - City Metaphor

The second step of the approach is city metaphor and to describe the city metaphor illustration, we select the Product Class in the online ecommerce store. These city metaphor visualizations are interactive and navigable using the keyboard. The large number of methods within a single class is easy to abstract. These isolated elements (building blocks) are useful when we want to highlight each part of the system.

We define a few methods to describe the above illustration and according to the above graphics objects of class, methods, variables and input parameters are mapped with the below code snippets.

public class Product{ //class **- total space**

String productId; //variables **- humans**

```
String productName;
Int quantity;
String productDesc;

        public static void main(String[] args) { //methods - building
        }

        Public void saveProduct(String productId, String productName, int quantity, Sting productDesc- windows){
                //save_product_code
        }

        Public void updateProduct(String productId, String productName, int quantity, Sting productDesc){
                //update_product_code
        }

        Public void deleteProduct(String productId){
                //delete_product_code
        }

        Public String listProducts(){
                String[] products;
                //list all product code
                return products;
        }
}
```

Figure 3 shows the message window of each class and the method is used to point out a summary of the relevant view. As an example if developer click on the each class or method object in abstraction level can determine more details about the object features and its code smells.

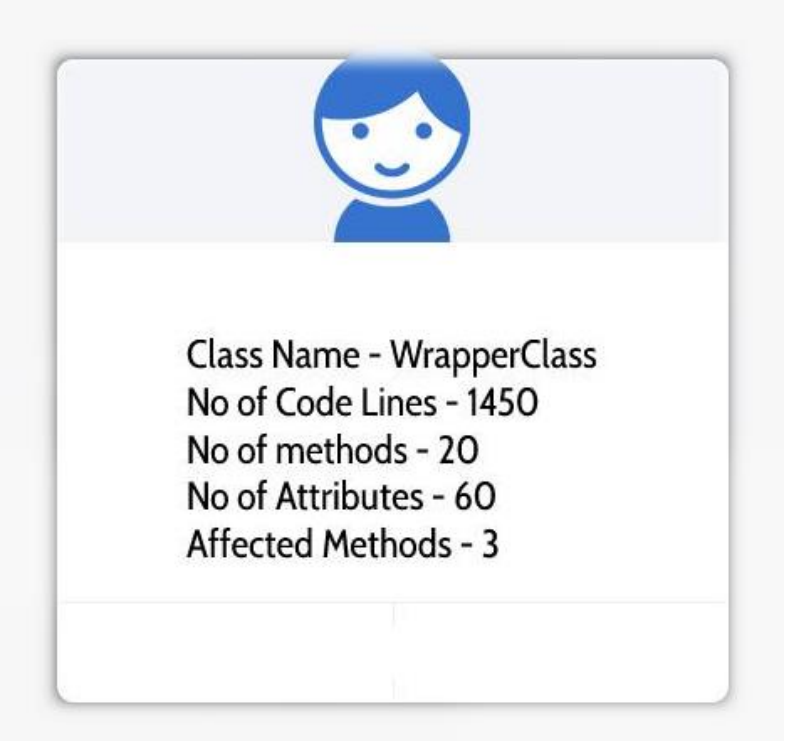


Figure 3 - Summary of Each Class or Method

The visual display as shown in Figure 4, is include attributes, analytical data of the code smell with respect to source code, a summary of suspicious code snippets. However, this visual output is generated when the user clicks on the building block (method). Children in the classroom represent the variables inside the method. Next, Pie charts, bar charts, and line charts on the wall visualize the analysis of the collected data throughout the process.

Additionally, the whiteboard (Figure 4) visualizes and lists code smells identified from the proposed visualization model. The Whiteboard on the wall includes code smell type in the method. Bad smells can be a long parameter list, feature envy, duplicate method etc.

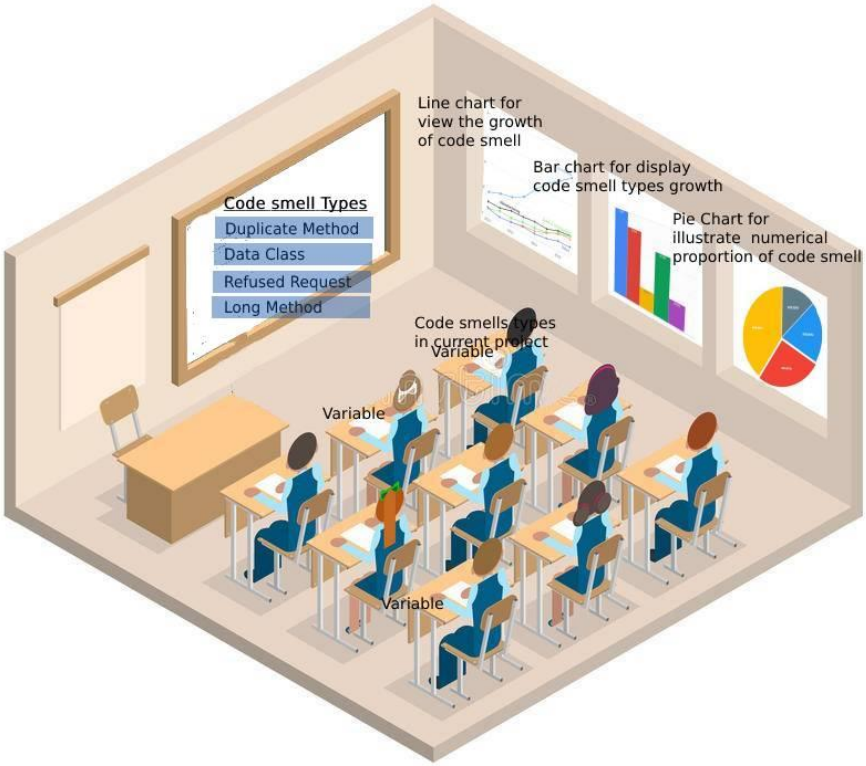


Figure 4 - Inside Building Block

Figure 5 illustrates the actual code snippet of the code smell types shown in Figure 4. Whiteboard describes the identified code smell [14] name and code snippet. According to the proposed visualization model Figure 5 is the final output.

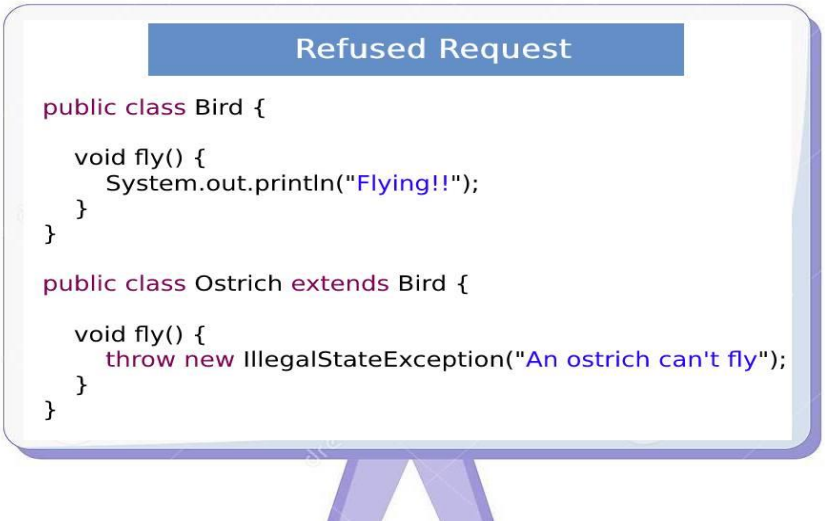


Figure 5 - White Board

Metrics regarding source code has to be generated in order to predict the size of visual elements. As for now the following metrics are to be generated. Table 1 summarizes the object mapping of the corresponding visualization techniques, island view and city view.

3.4 Object Mapping

Attribute		**Mapping Object**
Classes	:	Island
Lines of Code in Class	:	Perimeter of cylinder
Packages	:	Sea
Pie Chart View in Island metaphor	:	Numerical proportion of code smell with considering source code
Left Sidebar Navigation	:	Classes and methods including in the generate model
Right Sidebar	:	Details of Number of issues, severity, time effort need to fix the issue
Methods / Functions	:	Building Block
Attributes / Variables	:	Left Whiteboard Inside building
Input Parameters	:	Blocks on the Floor, Classroom
Lines of code	:	Building Height
Inside of Method	:	Classroom
Pie Chart	:	Illustrate the numerical proportion of code smell with considering source code

Bar Chart	:	Type of the code smells found in the source code
Main Whiteboard in the classroom	:	Available code smell types and smell snippets in the source code
Whiteboard in Right side wall	:	Suggestions to fix code smells
View Graph button, Left wall in Classroom	:	Illustrate the numerical proportion of code smell with considering source code in method

Table 1 - Object Mapping 3D Visualization Model

CHAPTER 4

4. IMPLEMENTATION

4.1 Introduction

This chapter describes the general high-level implementation of the proposed novel model for 3 dimensional visualization.

The tool is built on the knowledge of Visualization techniques, code smell detection tools and approaches. This knowledge is used to visualize the code smells that are include in source code. To automatically visualize the semantically imported JSON request generated from SonarQube, rules are defined on how syntax compositions should behave to reveal violations when scanning request.

4.2 Architecture

The main contribution in this thesis is the implementation of an automatic 3 dimensional code smell visualization tool for selected SonarQube project result set; the tool takes as input a set of data to be checked and outputs warnings about suspicious code constructs as code smells identified in the code.

The tool is rule driven. This means that there are clearly defined rules about how given code constructs in the code should be handled. The tool checks whether these rules are broken. Breaches are flagged as code smells.

The tool is meant for guidance and it will be helpful to identify code smells that programmers may be unaware of, on the other side the tool might warn about code smells that the programmer does intentionally.
The tool is implemented entirely in PHP Framework and JavaScript library call Babylon.js, the tool can therefore be deployed to an web hosting for every platform, without any third party dependencies. The tool provides a 3D model interface with the

ability to specify each level visualization in class, method and inside method. This visualization result generates according to the prepared set of parameters (JSON request) collected from SonarQube static analyze tool. The result is outputted to web browser window view.

The module has to be expressive enough to provide a total solution to visualize the code smells. Due to that reason we need to develop novel model for generate visualization mode of give extracted JSON request from SonarQube.

Developed novel 3 dimensional visualization model,

- Extract a data set from a project analyze in SonarQube.
- Visualization models for three abstraction levels (Class, Method, Inside Method)
- Algorithm for avoid the overlapping objects

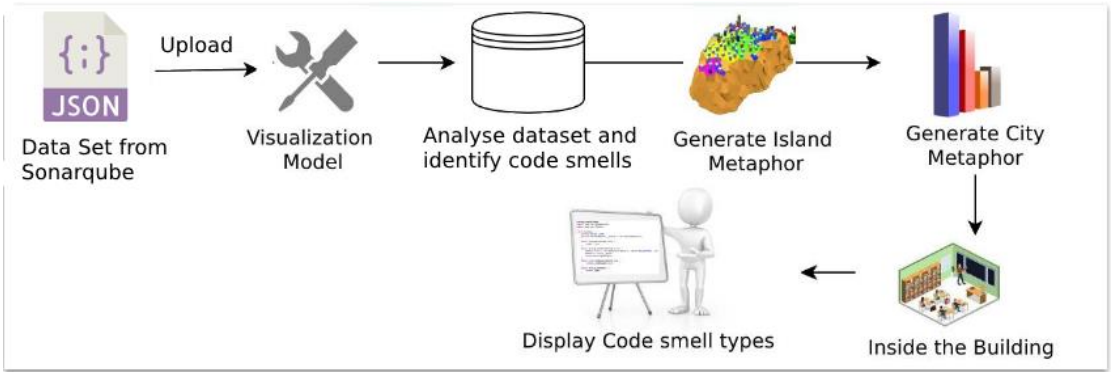


Figure 6 - Implementation Cycle

4.3 Extract a data set from SonarQube

The process of prepare a systematic format of input parameters from plain source code consist of two steps. First step is to import or select a project in SonarQube and perform a static analysis on it. The second step consists of analyze and find the raw data to apply

the predefined JSON format. Figure 7 is a sample view of a code quality inspection using static analysis call sonarqube to find bugs and code smells.

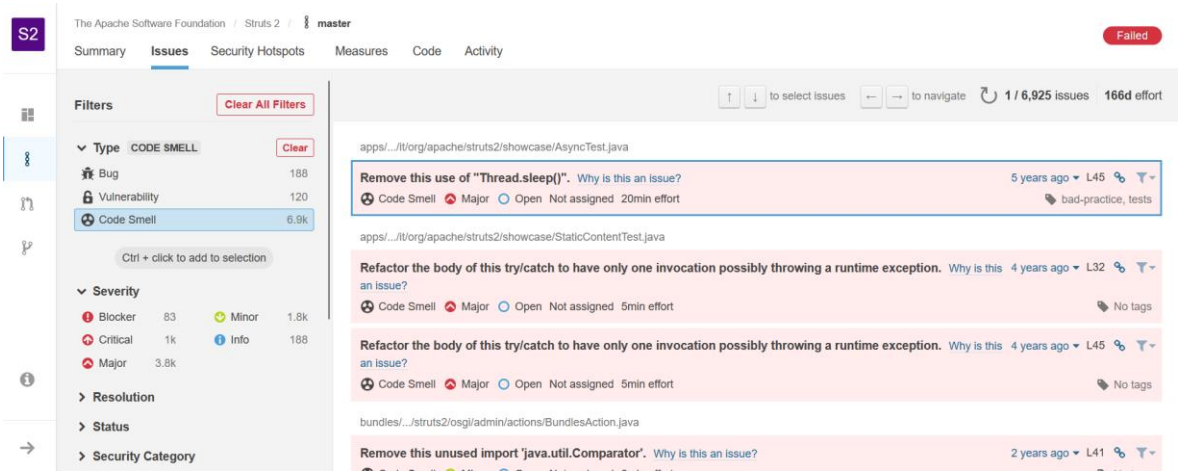


Figure 7 - Project analysis result in SonarQube.

We were created a novel model for code smells visualization to produce views from the previously defined data model. The code snippet below gives a sample view of JSON format of input data set. The specifications are saved as a data meta-model in a JSON format. The purpose of the visualization tool is to make the rendering process easier.

```
{
   "classes" : [
         {
         "class_name": "Product Class",
         "class_id" : 1,
         "no_of_lines" : 123,
         "position": [
                     {
                        "x": 3,
                        "y": 1,
                        "z": 4
                     }
               ],
         "is_code_smell" : "yes",
         "smell_type" : "Empty Class",
         "color_code" : "1, 0, 0",
         "code_snippet" : "final class DEFAULT { }",
         "methods" : [
```

```
            {
               "method_id" : 1,
               "name" : "getTransferData",
               "no_of_lines" : "12",
               "no_of_attributes" : "int status = 0;",
               "is_code_smell" : "no",
               "smell_type" : "",
               "code_snippet": "",
               "color_code" : "1, 1, 1",
               "priority" : "",
               "class_id" : "1",
               "effort" : "9",
               "suggested_code" : "",
               "suggested_links" : "",
               "position": [
                  {
                     "x": -5,
                     "y": 1,
                     "z": 34
                  }
               ],
               "parameters": [
                  {
                     "pname" : "String[] args"
                  },
                  {
                     "pname" : "Properties additionalUserProperties"
                  },
                  {
                     "pname" : "ClassLoader coreLoader"
                  }
               ]
            }
         ]
      }
   ]
}
```

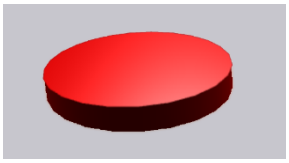
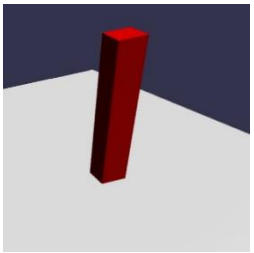
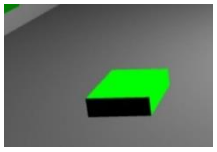

In this regard, creating a visual representation based on the generated JSON format imported into the program. The overall evolutional characteristics of code smells visualization, other visual objects representations, and object location calculations depends on the parameters include in the JSON format.

4.4 Visualization models for three abstraction levels

In addition to find code smells in source code, we implemented a visual approach [9], 3 dimensional visualization tool. This is a novel visualization model and made it using open-source technologies such as PHP and Babylon.js etc. With this respect, the related work on Chapter 2 utilizes to build this novel model. Additionally, this model employs one-to-one mapping and uses unit visualizations at each level of abstraction. It delivers a 360-degree view of visual objects including zooming and rotating features [24].

The following attributes inside in the source code are mapping as visual objects:

- Classes
- Methods
- Parameters
- Inside method
- Local Variables

Attribute	Visual Object
Class	 Cylinder
Method	 Block
Parameters	 Box

Inside method	 Inside Building

Table 2 - Object mapping

Mainly this visual representation consists of islands, buildings, and inside building. Islands have emerged from the sea. Islands are represented with gray color surfaces. A blue color background represents a package. Building blocks are the ones that represent methods and gray color ground surface represent the base class. Parameter are map to small boxes on the floor and white board represent code smells types, snippets, suggestions and local variables inside the method.

The perimeter of the cylinder and building block height corresponds to number of code lines. The colors of each object, dark red, light red, yellow, and green, represent the severity of the bad smells. The small boxes in green color on the floor inside the building represent the parameters declared in the method and the whiteboard on the wall in-building represent the code snippet, code smell types, local variables and suggestions.

Metrics extract from the source code mentioned in the JSON format use to predict the size of the visual objects.

- **Code Lines in a class** – Each Cylinder size is determined by the number of code lines in a class
- **Code Lines of method** – Height of each building block
- **The number of local variables** – This is represented in white board in left side wall inside the building.
- **The number of parameters** – This is represented by green color boxes on the floor. This is important when representing the long parameter list code smell. It

show with red color boxes when detect the long parameters list and visible to the user clearly.

4.5 Algorithm for avoid the overlapping objects

In this visualization model, objects placement on the canvas is an important factor while generate visualization model. This process concerns providing more convenient solution for place the object on the canvas without overlapping it. We applied an algorithm to avoid the overlapping of the visual object in all abstraction levels.

First get the class array from the JSON request and pick the first class and place it on the Canvas. After detecting first x, y coordinates, and sizes of that class, then calculate next coordinates by using first class object width and constant value. Iterate this process with considering the canvas width. After end of the first row in canvas then the visual object coordinate need to move next row.

According to this process take the first method / class block and check whether there is enough space to place that block/class in the root. Here in this case it has enough space then object will be placed in top left corner. Then the space in root will be divided into 2 as right node and left node. Then when it comes to the 2nd block it starts from root to check for spaces. Here the upper left corner is utilized. Then check spaces in its right and down. If that block is placed again space is split into 2. This happens until every block is placed inside the canvas.

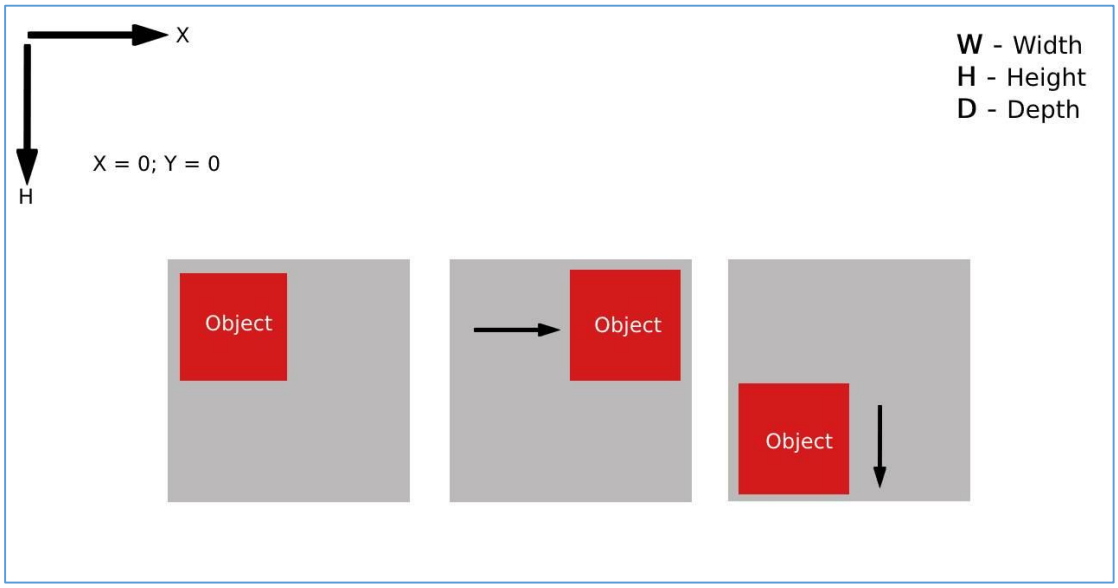


Figure 8 - Object moving to avoid overlapping

Root canvas has to be grown based on the method / class size. First we equalize root size into the size of the largest method. Then start the algorithm as earlier. Here the first block will be placed without issue since the root is equal to its size. From the 2nd block onwards root has to be grown.

4.6 Code smells visualization using Novel 3D model

The tool will be capable of producing 3D visualizations for buildings, islands, inside building views according to this dataset. Zooming, localization, and browsing are all important aspects being considered. Developers will be able to search for and discover specific buildings that corresponds to a specific classes. In addition, they will be able to zoom in or out the building visual objects.

The tips, navigations, summary graphs feature assists developers to navigate among buildings in 3D environment. These features assists users to understand and manipulate large number of visualization in a large scale systems with many classes.

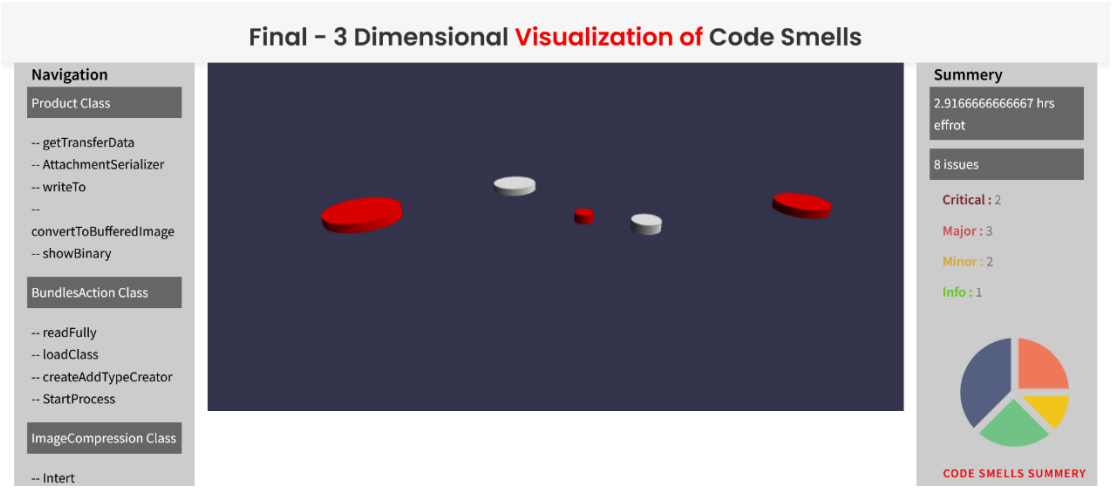


Figure 9 - 3D Model for Island Metaphor

The proposed approach is flexible enough to be extended to higher levels of abstraction. Individual island groups can form archipelagos, which will provide a first level of abstraction and the user interface of island view prototype was developed using the PHP framework (Code Igniter) and JavaScript library that help to create rich GUIs call Babylon.js.

It is mapping the dynamic parameters. As an example, this model filter the classes from the imported JSON request that generate using SonarQube result. Figure 8 illustrate the automatically generate class-level mode in the visualization tool. This perimeter of the cylinder island view vary from the dynamic values in the database and this perimeter calculation depends on the number of code lines include in class.

The dark red color used to highlight the code smells including classes. If any class or any other lower level member include code smells it will alerts from the class level.

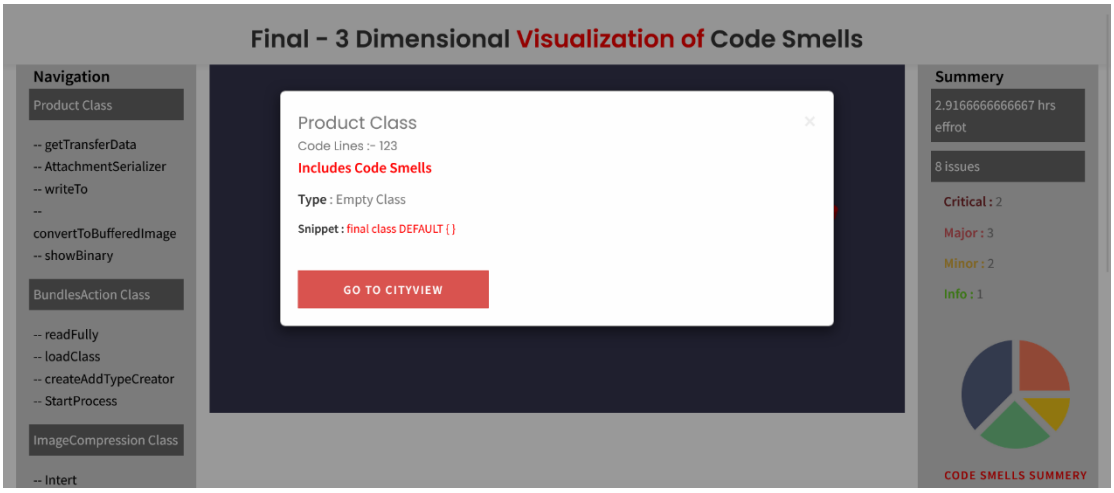


Figure 10 - Message box with details of the Class in Island Metaphor

It is important to mention that move to next abstract level need to click on a class. Then it shows a message box including details such as Class Name, number of Code Lines include in the class, is there code smell include in class or lower member level and link to next abstract level etc.

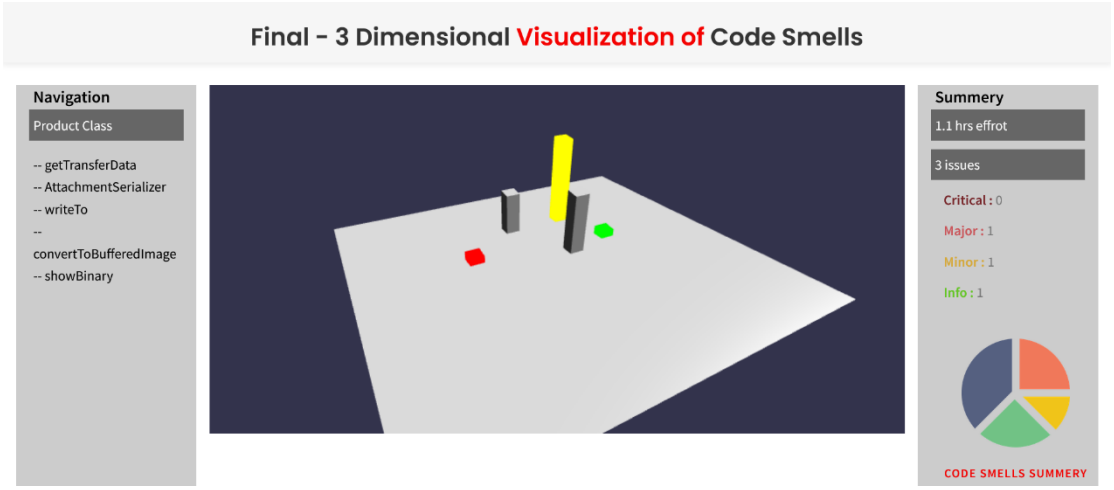


Figure 11 - 3D Model for City Metaphor

This section provides specification on the proposed automated three-dimensional approach for producing the visualization at the method level. The proposed framework depicted in Fig. 11 can be used to realize this automated approach. The block view represent the methods inside in class is displayed as the main component of the method level in proposed framework. The JSON request is used as the initial input of the

framework to begin the procedure. Then, the visualization model generated for three levels and it is visualize the identified bad smells. The second abstraction level is city metaphor illustration was designed using Babylon.js (JavaScript Library), Code Igniter (PHP Framework) and MySQL. This height of the building blocks vary from the dynamic values updated in the database. This height calculate based on the number of code lines include in the method.

The method block is shown on the different dark colors of the ground represent the class since it is a method related bad smell. These different colors based on their severity. The severity levels are critical, major, minor and info.

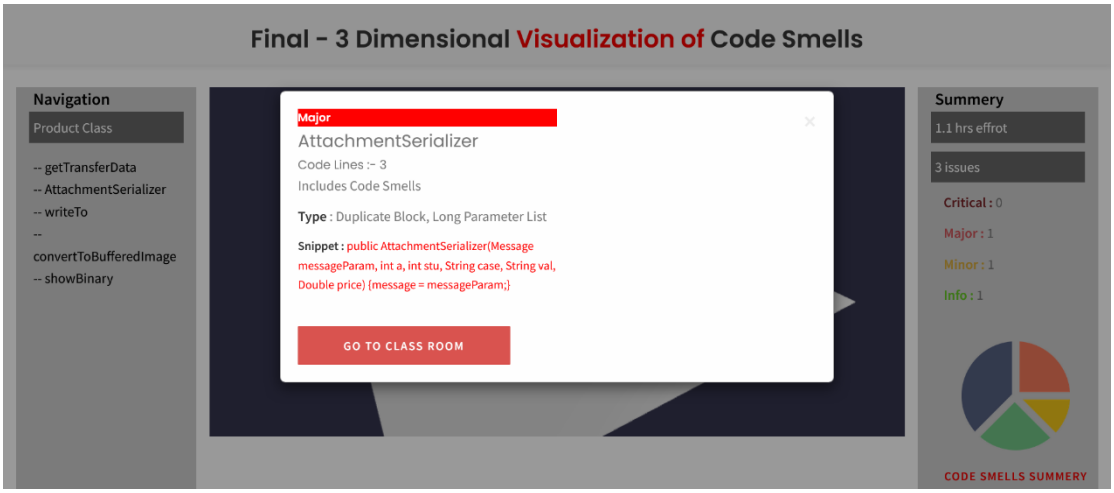


Figure 12 - Message box with details of the Method in City Metaphor

Fig. 12 shows message box with more details in the method. When a method includes code smells, this method listed details of code smell type, snippet and it severity in detail. A method that has large number of code lines, user can identify referring this message box. Finally, user can move to the inside method view by click on the Got to Classroom button.

The visualized visual object, building shown in Fig. 13 represents the inside method account. It comprises of comprises of three walls in detailed. The number of boxes in the floor represents the number of parameters for each method. The code smell type and error code is shown on the whiteboard in the main wall of the building. The smell

founded from the process provide the suggestion and tip to solve this code smell issues in right side wall whiteboard.

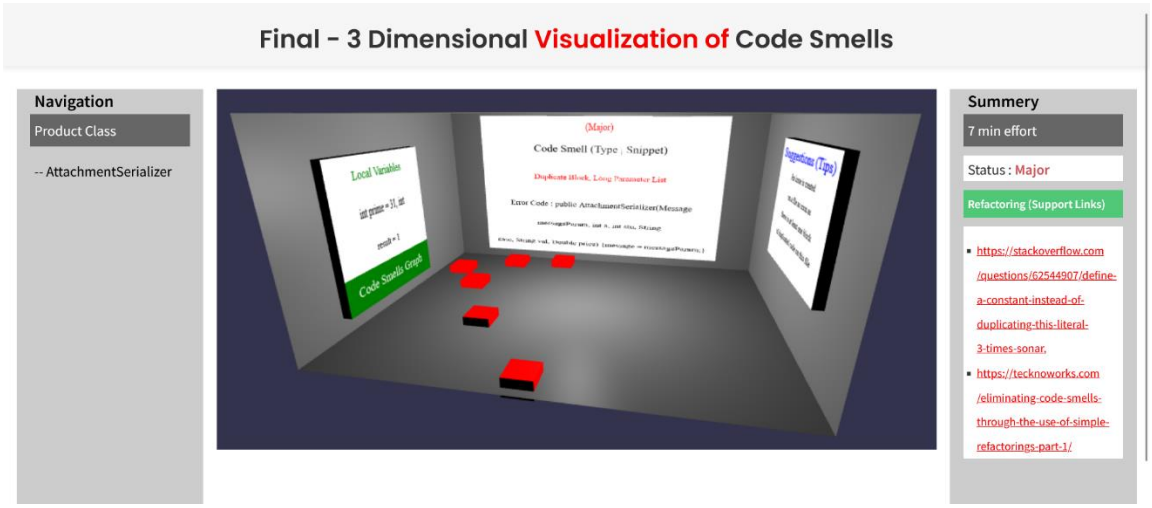


Figure 13 - 3D Model for Inside Building

This abstraction level (Figure 13), inside the building view user interface was designed by using Babylon.js (JavaScript Library), Code Igniter (PHP Framework) and MySQL. It displays if source code changed significantly or stayed unchanged with respect to code smells from the results.

Figure 14 - Parameter Names of Inside Building

Fig. 14 shows more visualizations about the parameters include in the method. The box on the floor represents the parameters and each popup box contains the parameter name.

Long parameter list is visualized as boxes on the floor of the building. This buildings include green color boxes indicate that it has minimal parameters and not contain any bad code smells. This floor consists multiple red color boxes that indicate the method identified as a code smells method.

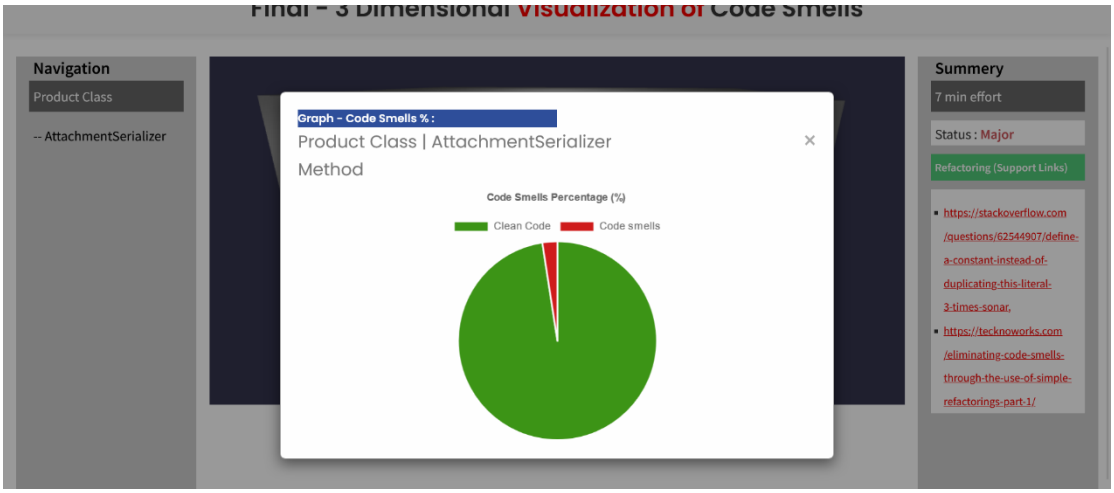


Figure 15 - Code smells % include in the Method

We applied the Pie chart visual model to represent code smells and clean code using the source code portion and color affected by a particular concern. Figure 15 displays how data are represented in the Pie chart. The portion colored in dark red corresponds to the percentage of smell code in the method affected by a particular concern.

In this figure, we can see supportive links (red color links), issue status (critical, major, minor or info), method you selected and its parent class (left sidebar in Figure 15).

Currently, we visualize the module of the software tool that is developed using Babylon.js (JavaScript Library), Code Igniter (PHP Framework) and MySQL. Visualizations in development allow us to visually explore the entire architecture at different levels of abstraction.

CHAPTER 5

5. EVALUATION

5.1 Introduction

We need to analyze the helpfulness and comprehension of the proposed three-dimensional visualization model in supporting comprehension tasks. As a result, we conducted a controlled pilot experiment on experts in software relevant field. The purpose of experiment is to see how the proposed visualization model helps in quickly understanding and identifying bad code smells in source code.

So, we performed a controlled pilot experiment on experts in software relevant field. The goal of the experiment is to check how the proposed visualizations model help in quickly understand and identify bad smells in source code.

The proposed model is planning to evaluate with sample software projects from the sonar cloud. It needs to select several software projects including code smells from SonalCloud. In this context, pick the GitHub selected project results and make the input request in JSON format. Then, upload the prepared JSON input to the proposed model and generate the visualization of the relevant project.

With regard to the proposed model, first, it will generate the island metaphor view including classes with code smells, navigation with classes and method hierarchy, code smells summery charts and other important data related to the project that applied.
After clicking on the class, the model illustrates the content of the class such as methods, variables, input parameters, etc.

In defining the proposed model we set up several objectives,

- Ability to identify and visualize the software project in proper understandable and four abstraction levels i.e Island metaphor view, city metaphor view, classroom view, and whiteboard illustration with code smells.

- Ability to identify, categorize and visualize the code smells in the software project

Eleven software specialists and researchers were chosen to participate in the pilot experiment. The following steps will be taken to familiarize students with code smells, detection approaches, and evolutional characteristics. First, all participants are given an overview of code smells and code smell visualizations.

Second, participants are given a detail explanation of the proposed visualization model. The attendees were given time to look over the proposed visualization tool. Following that, all participants were asked if they had any questions before beginning with the actual evaluation. The real experiment phase begins after verifying that all participants are familiar with the notion of code smell and its visualization.

The experiment was organized as follows:

(i) Google Form prepared with several questions for all level with covering process of the code smell visualization.

(ii) The targeted code smells were duplicate block, long parameter list, replace all(), Extract Try/Catch Block etc. that extract from project analyze in sonarqube. The case study has already completed this process.

(iii) Ask participants to identify the objects using software solutions and answer the question and time consumed for each level.

(iv) Get the feedback from the participants and analyze that visualization helps them and whether they easily use this model in tracing and investigating the detecting code smells and visualization objects.

(v) The duration that each participant requires to answer questions in each level and whole process of tour is also scree recorded.

5.2 Evaluation of Precision

According to the response of all participants, we evaluated the usefulness of proposed model visualization island view, enhanced building visuals, inside building visuals in code smell detection. A controlled pilot experiment is carried out to validate the notion that the suggested visualization model can improve the usability, efficiency, learnability, memorability, errors of code-smell evolution.

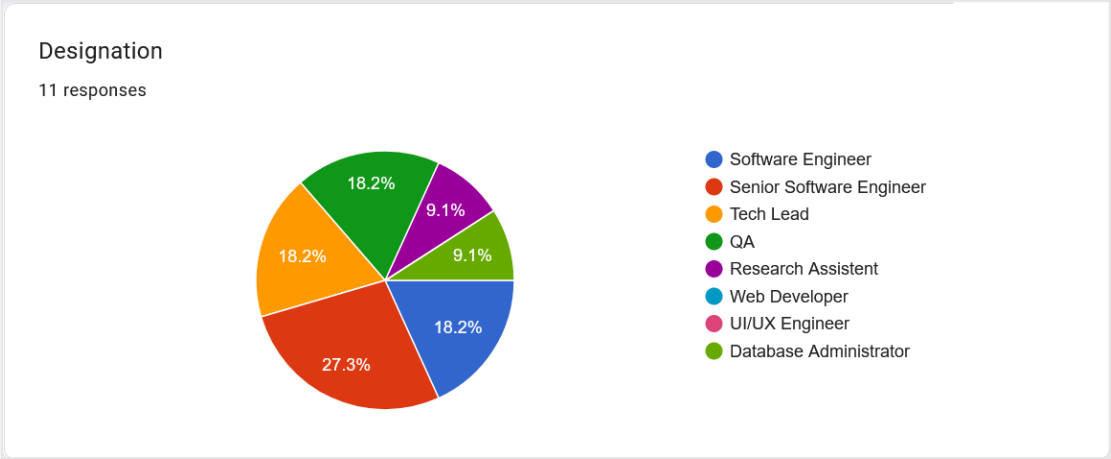


Figure 16 - Designation of Participants

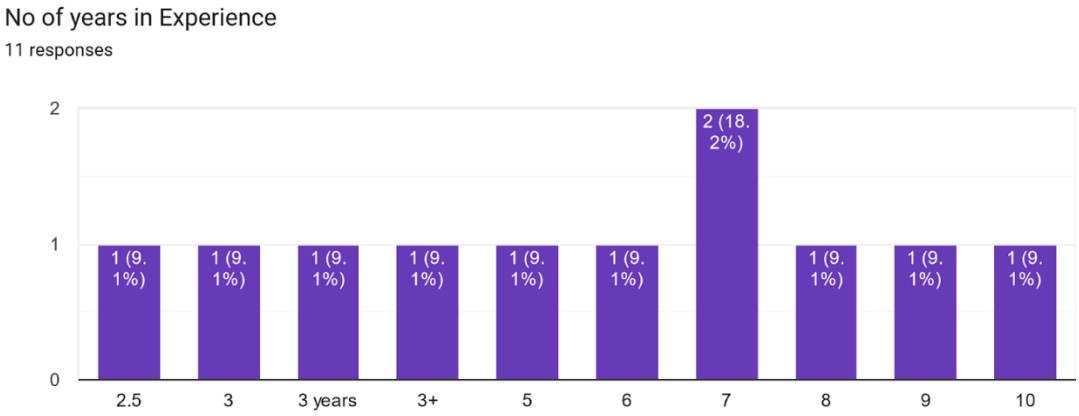


Figure 17 - Number of Years' Experience of Participants

Figure 18,19,20,21 shows the number of correct answers for questions submitted by eleven participants in the pilot experiment. The average correct answers for the

participant is 5 and more with visualizations. The first participant is a QA and he submitted 14 correct answers out of 17 total questions. 16 correct answers provided by the second participant.

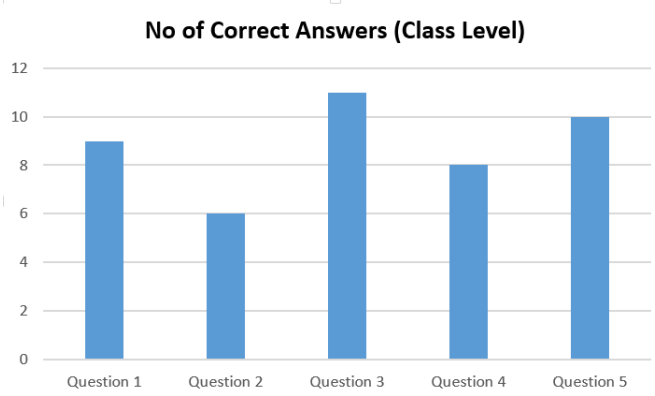


Figure 19 - Number of Correct Answers submitted in class level

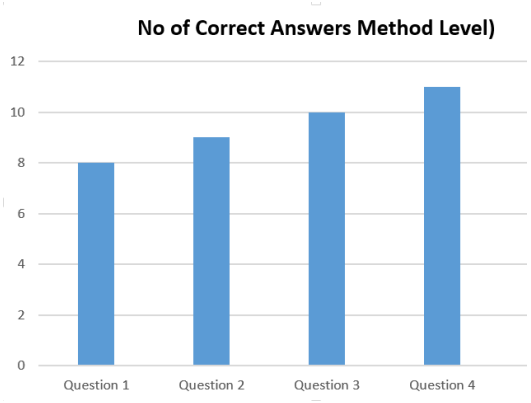


Figure 18 - Number of Correct Answers submitted in Method level

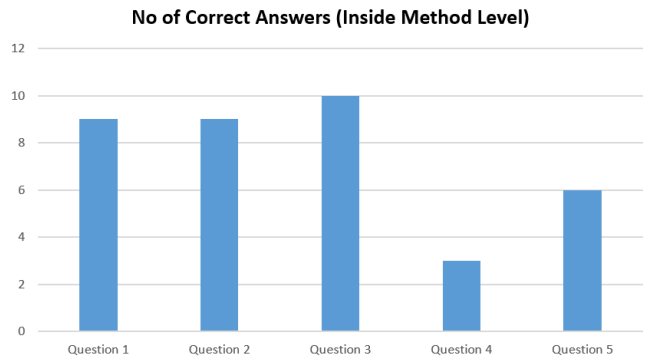


Figure 21 - Number of Correct Answers submitted in Inside Method level

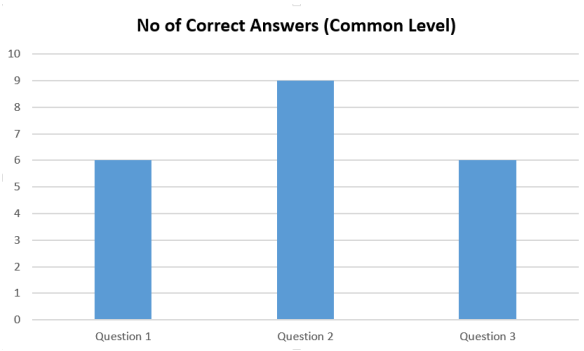


Figure 21: Number of Correct Answers submitted in Common level

The comparison results between the participant's submitted answers showed that most of them understand the visualized process and the visual objects.

5.3 Evaluation of Recall

Figure: 22 shows the completion time of the common level questions. These questions were designed to assess participants' recall (Memorability) of the novel model.

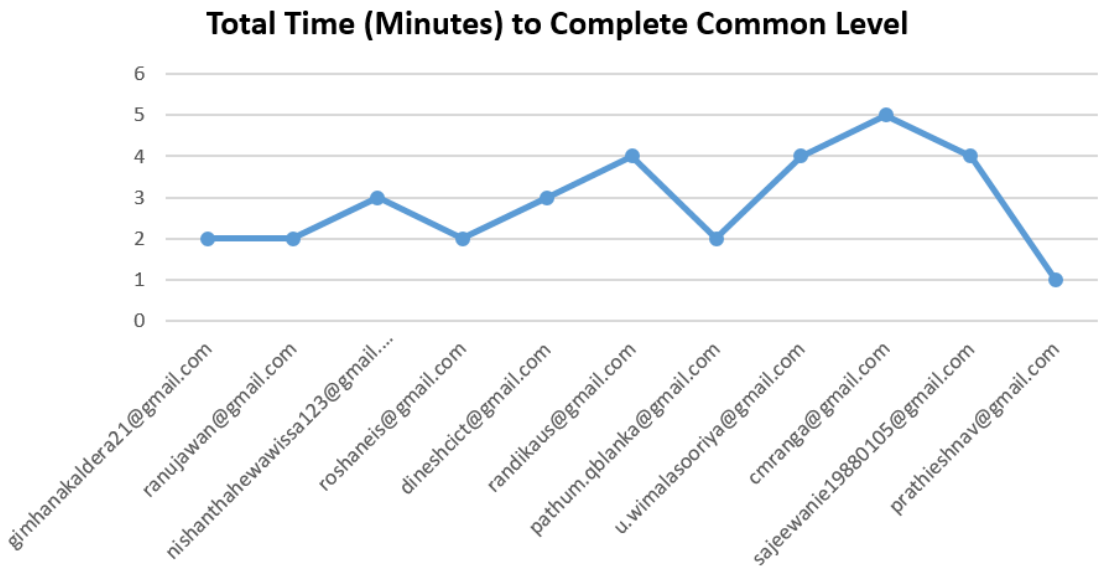


Figure 22 - Number of Correct Answers submitted in Inside Method level

This assesses users' ability to recall various functions, objects, and processes after learning the system's functionality. This group used 2.9 minutes average time to complete the task in 4th level in the visualization novel model.

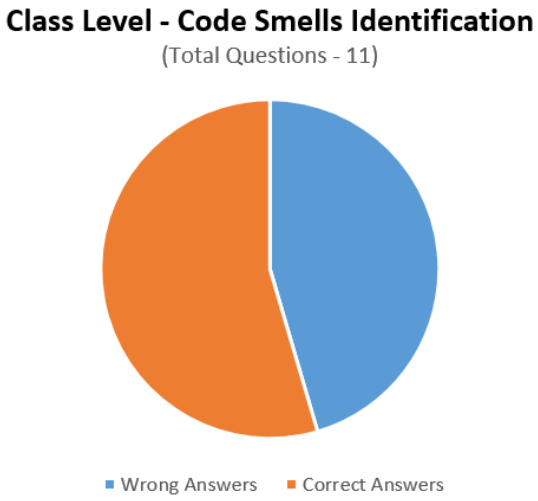


Figure 23 - Class Level Code smell visualization identification

After completing the experiment, we checked participants' answers related to code smells from the questions presented.

We used a different questions to verify the identification of the code smells visualizations in abstract levels. The participants were asked to answer the question and we filtered out the specific code smell visually identification questions. These questions should be answered using the novel model.

Figure 22 depicts the correct responses to the class-level questions submitted by the eleven participants. The group that used code smells visualization questions at the class level completed the task with 55% of all questions answered correctly.

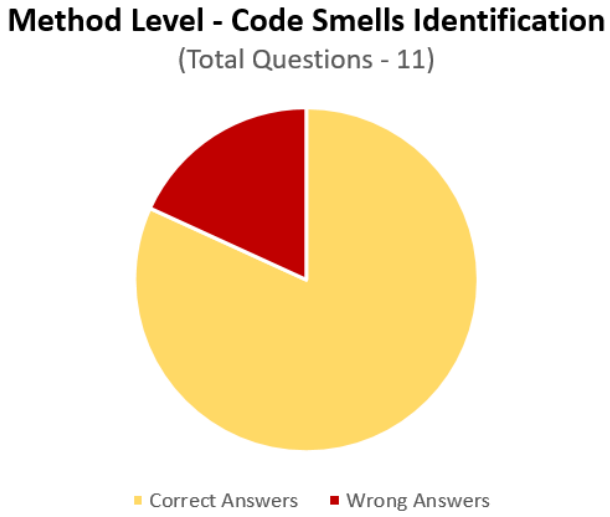


Figure 24 - Method Level Code smell visualization identification

Figure 24 depicts the eleven participants' correct answers to the method-level code smells visualization questions. The participants that used visualization questions at the method level completed the task with 82% of all questions answered correctly.

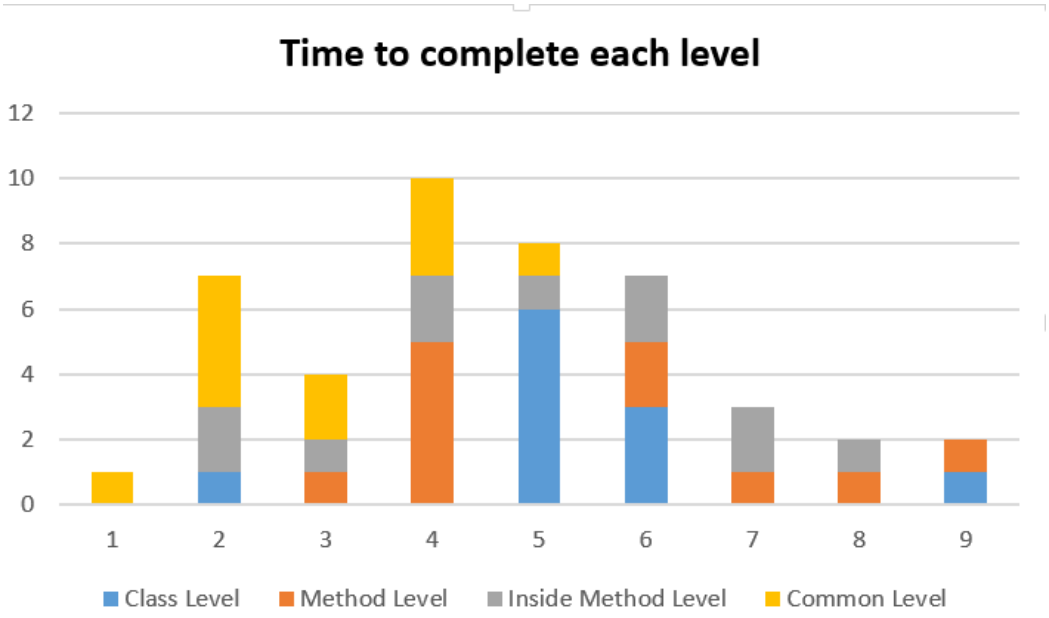


Figure 25 - Time to take complete each level by number of participants

This study relies on three consecutive abstract levels of process that class level, method level, inside method level. Participant were answered for the prepared questions by identifying the object and bad smells visualization in this tool. In this context, Figure 24 depicts the analysis of the time participants took to complete each level.

Hence the analysis clearly provides sufficient information about the total time taken for each level and time taken to complete all the levels. Therefore the average time to complete whole process is 17.7 minutes for each user. Table 3 shows the average time taken by all participants to complete all levels and each level separately.

	Class Level	**Method Level**	**Inside Method Level**	**Common Level**
Avg Time (Minutes)	17.2	5.4	4.6	4.8

Table 3 - Average time to complete the whole process and each levels

CHAPTER 6

6. DISCUSSION

This section presents precision and recall outcomes based on participant and code smell visualization oriented evaluation. We gathered direct data from participants via questionnaires filled out on Google forms.

The questionnaires available described the 3 dimensional visualization model main functionalities, visualization of code smells, priority levels of code smells, suggestions to solve. Participants were asked to identify the classes they suspected each code smells exhibited as well as the methods they used to identify them. They were also asked to explain the visual object and its role without utilizing the tool, using resources like as views, concerns, objects, and colors. Based on their responses on Google forms and screen recorded activities, the objective is to gain a better understanding of the participants' strategies.

Precision and recall are two metrics to analyze to which extent the 3D novel model of code smells visualization supported the code smell identification. These metrics were derived from the evolution of information retrieval processes. The precision metric estimates the rate of accurately recognized visualized code smells by number of participants. The rate of successfully detected code smells visualization by the total number of actual code smells is quantified by recall (memorability).

In a manual study, the Google form questionnaires and screen recording were utilized to reveal the strategies applied by the participants to complete each activity. We analyzed how distinct abstract level views were typically used together by each participant to complete each experimental task. The evaluation results in a set of propositions supported by both qualitative and quantitative findings for the proposed approach.

Efficiency is defined as how quickly users complete their tasks without wasting time or resources. Also included in this visualization model is learnability, which is the ability of users to learn how to use a system as quickly and simply as possible the first time they engage with it. Here we analyzed how many errors do participants make and how severe are these errors and how easily can they recover from the errors through this evaluation.

This analysis is discussed about precision and recall. Figure 22 presents the average time taken to complete recall section (common level) of all participant in the code smell visualization. According to the Figure 22, the average time of the all participant to complete that section is 2.9 minutes.

Here first 3 levels (class, method and inside method level) are based on the precision and common level is based on the recall. In precision of the novel model figure 23 and 24 show correct answers percentage of all participant. This data confirmed that the percentage of correct answers was more than 50%. Therefore, it determined that the identification of code smell visualization using this tool is more accurate with considering correct answer percentage.

CHAPTER 7

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

Code smells visualization support program comprehension tasks for developers. We have presented a model for identity and visualize code smells as a novel mechanism for formulating software projects in different visualization aspects to increase the understandability and maintain the quality of the software project.

A useful visualization approach has been proposed to assist developers in identifying code smells. These visuals were mapped to objects based on the types of bad smells recognized by analyzing data extracted from sonarqube. It offers a clear illustration of the abstraction levels of a software project. It uses a combination of the code city and island metaphor visualization techniques. This framework is presented to increase understanding and efficiency to identify and refactor the code smells in different abstraction levels in software projects. The evaluation of the novel visualization model revealed a beneficial influence on easily and clearly comprehending code bad smells and their causes at each abstract level.

7.2 Future Work

We are focusing on visualizing more visualization objects like as connections / relationships between classes, methods, data types, and attribute invocations. Our future work will focus on completely implementing the framework and integrating it with a code version control solution.

Most of the software companies doing code maintaining using the version controlling tools like GitHub, GitLab, BitBucket etc. Currently, when a feature is developed and the merge request is sent to the senior manager, he needs to review the entire code to

identify bugs and if it needs a refactoring. But before going to source code analysis if we can provide visual analysis of source code using our new visualization model then higher management can easily make decision about software projects.

REFERNCES

1. Chathuranga Hasantha, A Systematic review of code smell detection approaches, 2021
2. Marcel Steinbeck , An Arc-Based Approach for Visualization of Code Smells
3. Nabilah - Controlling Software Evolution Process Using Code Smell Visualization – 2019
4. Karim, Visual Detection of Design Anomalies - 2008
5. M. Fowler and K. Beck. Refactoring: improving the design of existing code. Addison-Wesley Professional, 1999.
6. Haris Mumtaz- Detecting Bad Smells in Software Systems with Linked Multivariate Visualizations - 2018
7. Lanza, M., & Marinescu, R. (2007). Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media.
8. Hammad, Maen & Alsofriya, Sabah. (2019). Visualizing Code Bad Smells. International Journal of Advanced Computer Science and Applications. 10. 10.14569/IJACSA.2019.0100536.
9. Emden, Eva & Moonen, Leon. (2002). Java Quality Assurance by Detecting Code Smells.f
10. Emerson Murphy-Hill and Andrew P. Black. 2010. An interactive ambient visualization for code smells. In Proceedings of the 5th international symposium on Software visualization (SOFTVIS '10). Association for Computing Machinery, New York, NY, USA, 5–14. https://doi.org/10.1145/1879211.1879216
11. Yamashita, Aiko. (2012). Assessing the capability of code smells to support software maintainability assessments: Empirical inquiry and methodological approach.
12. M.Lanza, R.Marinescu, Object-Oriented Metrics in Practice – Using Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems,

Springer – Verlag, Berlin, Heidelberg, New York, Germany, 2006, ISBN 978-3-540-24429-5

13. Schumacher, Jan & Zazworka, Nico & Shull, Forrest & Seaman, Carolyn & Shaw, Michele. (2010). Building empirical support for automated code smell detection. ESEM 2010 - Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. 10.1145/1852786.1852797.
14. Mäntylä, Mika & Vanhanen, Jari & Lassenius, Casper. (2004). Bad smells - Humans as code critics. IEEE International Conference on Software Maintenance, ICSM. 399 - 408. 10.1109/ICSM.2004.1357825.
15. Li, Wei & Shatnawi, Raed. (2007). An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. Journal of Systems and Software. 80. 1120-1128. 10.1016/j.jss.2006.10.018.
16. Dhambri, Karim & Sahraoui, Houari & Poulin, Pierre. (2008). Visual Detection of Design Anomalies. Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR. 279-283. 10.1109/CSMR.2008.4493326.
17. Carneiro, Glauco & Silva, Marcos & Mara, Leandra & Figueiredo, Eduardo & Sant'Anna, Claudio & Garcia, Alessandro & Mendonça, Manoel. (2010). Identifying Code Smells with Multiple Concern Views. 128 - 137. 10.1109/SBES.2010.21.
18. Schumacher, J., Zazworka, N., Shull, F., Seaman, C., & Shaw, M. (2010, September). Building empirical support for automated code smell detection. In Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement (pp. 1-10).
19. Ghoniem, Mohammad & Fekete, Jean-Daniel & Castagliola, Philippe. (2005). On the Readability of Graphs Using Node-Link and Matrix-Based Representations: A Controlled Experiment and Statistical Analysis. Information Visualization Journal. 4. 10.1057/palgrave.ivs.9500092.
20. A. R. Teyseyre and M. R. Campo, "An Overview of 3D Software Visualization," in IEEE Transactions on Visualization and Computer Graphics, vol. 15, no. 1, pp. 87-105, Jan.-Feb. 2009, doi: 10.1109/TVCG.2008.86.

21. Graham, Hamish, Hong Yul Yang and Rebecca Berrigan. “A Solar System Metaphor for 3D Visualisation of Object Oriented Software Metrics.” InVis.au (2004).
22. Emerson Murphy-Hill and Andrew P. Black. 2010. An interactive ambient visualization for code smells. In Proceedings of the 5th international symposium on Software visualization (SOFTVIS '10). Association for Computing Machinery, New York, NY, USA, 5–14. DOI:https://doi.org/10.1145/1879211.1879216
23. I. d. J. Silva, M. S. R. Santos, L. L. Ramos and L. P. d. S. Carvalho, "VISMELLS: An Interactive Visualization for Identifying and Evaluating the Effects of Code Smells on Software Projects," 2018 XLIV Latin American Computer Conference (CLEI), 2018, pp. 40-49, doi: 10.1109/CLEI.2018.00015.
24. Nabilah and Wikan Danar Sunindyo. 2019. Controlling Software Evolution Process Using Code Smell Visualization. In Proceedings of the 2nd International Conference on Control and Computer Vision (ICCCV 2019). Association for Computing Machinery, New York, NY, USA, 51–54. https://doi.org/10.1145/3341016.3341026
25. Katbi, A., Hammad, M. and Elmedany, W. (2020), Multi-view city-based approach for code-smell evolution visualisation. IET Softw., 14: 506-516. https://doi.org/10.1049/iet-sen.2020.0010
26. Mumtaz, Haris, Fabian Beck, and Daniel Weiskopf. "Detecting bad smells in software systems with linked multivariate visualizations." 2018 IEEE Working Conference on Software Visualization (VISSOFT). IEEE, 2018.
27. Wettel, Richard, and Michele Lanza. "Visualizing software systems as cities." 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis. IEEE, 2007.
28. Hammad, Maen, and Sabah Alsofriya. "Visualizing code bad smells." International Journal of Advanced Computer Science and Applications 10.5 (2019).
29. Misiak, Martin & Schreiber, Andreas & Fuhrmann, Arnulph & Zur, Sascha & Seider, Doreen & Nafeie, Lisa. (2018). IslandViz: A Tool for Visualizing

Modular Software Systems in Virtual Reality. 112-116. 10.1109/VISSOFT.2018.00020.

30. Merino, Leonel & Fuchs, Johannes & Blumenschein, Michael & Anslow, Craig & Ghafari, Mohammad & Nierstrasz, Oscar & Behrisch, Michael & Keim, Daniel. (2018). On the Impact of the Medium in the Effectiveness of 3D Software Visualizations. 10.1109/VISSOFT.2017.17.