Code Smell Detection and Refactoring Using Automated Genetic Algorithm Approach

Faizan Yasin and Kareem Ullah

Department of Computer Science, University of Agriculture, Faisalabad, Pakistan

ABSTRACT

Code refactoring is the method of changing a software’s internal structure without modifying its functional behavior or functionality, to be able to improve internal non-functional properties of the software, as an instance to improve code readability, to simplify code structure and to enhance maintainability to improve overall performance of the software. This leads the software to smell. Refactoring of software provides different approaches to eliminate these anomalies. It is formidable challenge to manually determining and to perform useful refactoring. Therefore, proposed the Object Oriented searched based refactoring method. From past it was observed that software refactoring has not received attention properly. This work presents approach based on genetic algorithm for automated component-based refactoring. There are four basic steps involved for automated code refactoring, at first it includes studying the literature for formulating rules for bad smell detection, then it includes list of components that are relevant to refactoring. After listing out the components the extraction of facts and construction of model from facts extracted from source code and in end it identifies the best solution for refactoring of code by refactoring the code by implementing refactoring methods.

Key words: Code smell detection tool, software, smell detection JavaScript, programmers, source code

INTRODUCTION

A software becomes harder to maintain when it continues to evolve with the passage of time. It becomes more complicated and complex and it becomes harder to find the smell and it was essential to understand code. When reorganizing code, the most important thing is making it ensure it behaves in the same way it was coded to taken place. Code refactoring is the method of changing a program’s code in such a way that it donot have any effect or any change in the behavior of the program, to be able to improve internal non-functional properties of the software. The code refactoring idea was proposed by Sir. William Opdyke in 1992. In this research a tool was developed to do the automated refactoring specially for JavaScript programmers to refactor their code by automated smell detection. Rodrigo et al., proposed an approach for the solution of problem of correcting anti-parties. In this research researcher found that different developer prefers to give
suggestions of refactoring of the files which are active in the workplace. They found that consideration for developer's was poor adaptation of automated refactoring approaches. Many of the other approaches of refactoring require the developer to inject bad code for the detection, these requirements of preforming manual refactoring slows the refactoring process and put extra work on the developer. Some researchers also worked on clustering methods which are used for refactoring of code. These methods were SLINK, CLINK and WPGMA and many others as move method refactoring\(^1\) to refactoring the software at functional level. They also introduced a new clustering method called A-KNN. Researcher conducted comparative study between newly suggested algorithm and three other clustering methods. There are three industrial systems used in the experiments. Results of experiments shows that newly suggested algorithm was competitive performance than the others because A-KNN calculates similarities among the classes only once where as other algorithms, hence it requires less computation power then other algorithms. Bartosz et al.\(^2\), analyzed that presence of design pattern for coding results in the absence of code smell. They conducted research by performing experimentation on two JAVA systems. It was found that with the presence of design patterns for the removal of the code smell it reduces chances of bad smell of code. This research was conducted for the development of tool for JavaScript refactoring because there is no such tool that is specially designed for JavaScript refactoring. When there is existence of any smell in JavaScript code it stops working without reporting any smell or error. Hence there is need of such a tool which can be used to detect smell in JavaScript code to facilitate developers to maintain their code.

This research was conducted to facilitate the developers to detect smell in their code so that code becomes easy to maintain and understand. Main focus of this research is to facilitate JavaScript developers, developing a tool for them to maintain their code more easily.

**MATERIALS AND METHODS**

**Methodology:** In this research a tool was developed to detect the JavaScript code smell and to diagnose the code smell to refactor the smell of the JavaScript code, the tool developed is for non-object oriented based JavaScript code smell detection and refactoring, this by combining the dynamic and static analysis onto the JavaScript code to gather required information about the code and to diagnose the smell in the JavaScript code. In the Code as discussed earlier that the code smell is the potential and the maintenance issue in the code that needs to be removed by applying refactoring techniques the same thing applies for the all languages. Code smell detection is very much time taking process and this may cause errors in the code may be logical or syntax errors in the code because of lacking specific code smell detection tools for the JavaScript, this research will provide a tool which will overcome the dynamic nature of the JavaScript program.

Process of detection of code smell is completely depending on the objects and classes and the methods and statements used in the code, here in this work only discuss about the JavaScript’s non-object-oriented nature. As for Java or C++, it is not that easy to identify elements in the JavaScript language. Along with the theoretical contribution this research will provide a web-based tool to detect bad smell within the JavaScript. The tool provided was open source and the basic goal of the tool is to diagnose the smell in the code. It will also help in reduction of the refactoring cost as it will scan the code and automatically suggests for the refactoring’s with the help of which it will increase the maintainability of the program and facilitates the developers by avoiding complete analyzing of the code manually. There are many types of code smells in the languages whereas in this study chose few of the types to implement on the JavaScript code to detect specific types of smells in the code of JavaScript.

Work done in this research indicates the analysis of the JavaScript code was not simple and easier as that of other object-oriented languages. JavaScript provides few of the features of object-oriented programming languages, but its behaviors was little different from the originally object-oriented languages. This difference in the prototype of the inheritance which somehow resembles with object-oriented languages and the exception existing in the JavaScript which is because of programmers can modify existing functions and can create new functions at runtime which shows redeclaration of hierarchy of objects during the program execution.

If this approach was not followed then there is need to rely on the information by analyzing the code manually. This tool is designed to facilitate the developers to ease them in finding smells in their code of JavaScript, The characteristics and purpose of the development of this tool was to make the smell detection with the help of it easier for the programmers who do web-development using JavaScript coding and are lack in the depth of the knowledge of JavaScript so that the maintainability of code can be make easier for them. With this tool code written by the developers who do JavaScript based web application development it will tend to decrease the smell in the code. As described earlier that code smells are the
poorly coded source code and the design patterns that depicts the potential of maintenance issues present in the code. When code smell is detected then it is required to perform refactoring onto the code manually and the code quality is maintained then. This tool reports the smells in the code that is scanned to refactor and refactoring needs to be implemented manually.

**Components of CSD:** Components of the code smell detector are categorized into three modules that works together to get the desired goals of detecting code smell and suggesting for the refactoring onto the code where bad smell is detected. These three components or modules are browser handler the code analyzer and the report generator all the modules work together to perform code analysis for the smell detection from within the code.

These components includes frontend manager which manages the code to parse and send it to the analyzer where it is being checked for smells and responding to the UI of the browser then the second component of the tool is analyzer which parses the JavaScript code and then analyses the code for the standards and find the smells in the code then it submits its results to the reporting module of the tool which further passes these results to the browser handler module to display the reports of the code that is scanned to let the developer know about the violations and smells in the code he scanned and then the code is refactored manually to remove the smells from the code.

- Editor, User Interface (UI) Handler which includes browser handling
- Analyzer In between browser and the frontend for parsing and analyzing code
- Reporter, that generates the report based on the analysis performed

In this tool process of code smell detection starts when developer or programmer put his code for the analysis in the editor on the browser window and request to scan the code for the possible smells in which may be invoked un intentionally while the development process of the project based on JavaScript. When a programmer puts its code for the analysis it first requests the browser for allowing the code to pass to the analyzer of the tool where the code is parsed and checked for the possible smells which may be caused during the development of the project. It is very handy for the developers to detect the smell and badly coded items in the program because manually checking of the code for smell is very hard task and requires a lot of attention and code for the deep analysis of the code manually, where as this tool will facilitate to remove the smell easily because it provides the point of errors in the codes such as the function or statement coded poorly it will prompt that this is causing smell in the program.

All the analysis of the code is done in the analyzer of the tool where code is passed through the checks for the basic coding conventions to check that if it is according to the standard where as if any of the code section is not according it rapidly reports to the analyzer of the program for the smell and violation which further sends the request to the reporting module of the program. Which is in between the analyzer and the browser handler just as parser in between the browser and the analyzer. After the results are forwarded to the report publisher module, where the report is further interpreted to be able to display it onto the browser window so that it can be easily understandable by the developer or programmer to let know about the smelly parts of the code and then he can perform refactoring to fix those smells and make his code more maintainable and light but efficient in the performance. The last module of the tool is for the developer to show that what are the smelly components of the code coded and what are the problems that are needed to be refactored and then the developer can easily get where the program smells badly and what are the basic parts of the code that are needed to be analyzed and to be refactored to make the code more reliable, efficient and more easily maintainable. After interpreting the smells developer performs refactoring on the code and made the code up to the coding conventions. Refactoring is the most time consuming process and it required a lot of attention to check for the smells and to detect what kind of the smell is produced in the code and what are the refactoring needs to be applied to the smell and violations of the code quality to make it more efficient but with the help of this tool it will be not that much time consuming to perform refactoring of the code once developer or a programmer scans his JavaScript code from this tool he will not needs to perform the analyses of the code manually or statically.

**Smell detection manager:** In Code smell detection tool, the code smell detection starts once developer or computer programmer places his code for the analysis onto it as shown in the Fig. 1. Place the code within the editor on the browser window and request to scan the code for the potential smells within the code which can be invoked in the project or code of JavaScript. Once a computer programmer puts its code for the analysis it initial requests the browser for permitting the
Fig. 1: Performance of code smell detection tool

code to pass to the parser of the tool wherever the code is parsed to the analyzer and checked for the potential smells which can be caused throughout the process of the project development, it’s very useful and helpful for the developers to find out the smell and badly coded things within the program as a result of manually checking of the code for smell is incredibly arduous task and needs a great deal of attention and code for the deep analysis of the code manually, wherever as this tool can facilitate to get rid of the smell simply as a result of it provides the purpose of errors within the codes like the perform or statement coded poorly it’ll prompt that this is often inflicting smell within the program.

All the analysis of the code is completed within the instrument of the tool wherever code is well-versed the checks for the fundamental writing conventions to ascertain that if it’s in line with the quality wherever as if any of the code section isn’t according it chop-chop reports to the instrument of the program for the smell and violation that additional sends the request to the reportage module of the program, that is in between the instrument and therefore the browser handler even as computer program in between the browser and therefore the instrument. When the results are forwarded to the report publisher module wherever the report is additional understood to be ready to show it onto the browser window, so it is simply grasurable by the developer or computer programmer to let realize the foul elements of the code then he will perform refactoring to repair those smells and create his code additional rectifiable and lightweight however economical within the performance.

The last module of the tool is for the developer to indicate that what are the foul elements of the code coded and what are the issues that are required to be refactored then the developer will simply get wherever the program smells badly and what are the fundamental elements of the code that are required to be analyzed and to be refactored to create the code additional reliable, economical and additional simply rectifiable. When decoding the smells developer performs refactoring on the code and created the code up to the writing conventions. Refactoring is that the most time overwhelming method and it needed a great deal of attention to ascertain for the smells and to discover what quite the smell is made within the code and what are the refactoring must be applied to the smell and violations of the code quality to create it additional economical however with the assistance of this tool it’ll be not that abundant time overwhelming to perform refactoring of the code once developer or a computer programmer scans his JavaScript code from this tool he won’t must perform the analyses of the code manually or statically. After refactoring it may impact on the behavior of the code so in that case if refactoring impacts on to the output of the code then it is needed to be reversed. Reverse refactoring can be simple done by un doing the refactoring’s from the code, but this happens in that case when smell’s nature is not concentrated carefully after reversing the refactoring’s it is required that to find the possible refactoring to remove the diagnosed smell in the code. Just like Extraction of method and Inlining of a method can be reversed.

**Code analyzer:** The analyzer component of the Code smells detector was somewhat like the sniffing dog, this component of the tool just has the key importance for detection of bad smell in the code and for the coding convention violations occurring in the code. By analyzing the code this component finds that which section of the code was most smelly. It sends the reports to the report parser for each of the code smell when it was detected and the areas that were problematic were reported to the report manager of the code smell detection tool. Which after interpreting the reported issues
passes to the browser window to show the smelly areas of the code onto the browser window and highlights the issues that were needed to fixed according to the coding conventions.

**Code smell resolution:** In this tool developed in such a way that it just displays the smells in the code and shows the responding smells in the code which needs to be refactored manually by implementing refactoring code smell can easily be eliminated, to every code smell there are rules and methods that are already discussed previously in this study. Which are general rules and are easily implementable to all the programming languages. Code smell detection process was one of the parts of code resolution because for the removal of smell in the code it was necessary to know about the smells in the code and this smell can be easily removed by the process of refactoring. This may include strategies and techniques that were used for removing smells in the code according to the nature of the smell just like re-declaration of function can be removed by changing the name of the function and if there were irrelevant methods name used inside of the other class then it was needed to be moved to its appropriate class to be renamed it according to its behavior performing function in the program.

**Annotation suggester:** In code smell detection tool annotations were suggested regarding the smells of the code which are helpful for the developer[8] to perform refactoring according to the annotation suggested. It reduces the work load of the programmer by giving suggestions to the programmer that what type of refactoring are needed to implement in the code. It suggests for the refactoring which was needed to be implemented on the specific piece of code, such as if programmer writes two methods with the same name annotation suggested will prompt that function was defined already and it will override the previously defined function which makes it easier to the developer to rename or to change the name of method. It only suggests the solution but donot automatically implements because if program automatically changes the name or implements the refactoring it may become troubling specially at the end of the project if it happens because the output of the function or program will be changed with the change in the name of method. Therefore it only suggests the refactoring’s to the developer to let him know about the issues exists in the code and refactor it manually to fix the smell in the code.

**RESULTS AND DISCUSSION**

Previous work has introduced many approaches to find the refactoring’s. Some work mentioned the thoughts of combining code smells detection with refactoring tool but none has actually implemented it. This work proposed and developed a framework that combines the processes of identifying the smell in the code and suggesting to apply refactoring’s. There was no such framework especially for the JavaScript programs which helps the programmers to analyses their code. Such a framework allows information from analysis to be reused and avoid re-computing any information that was already known. Therefore, it helps improve the efficiency of the overall system.

The tool was developed for the detection of JavaScript code smell. It automatically scans the code for the smell and reports for the possible refactoring’s and the smells in the code to refactor the smell of the JavaScript code, the tool developed was for non-object oriented based JavaScript code smell detection and refactoring, this by combining the dynamic and static analysis onto the JavaScript code to gather required information about the code and to diagnose the smell in the JavaScript code. In the Code as discussed earlier that the code smell was the potential and the maintenance issue in the code that needs to be removed by applying refactoring techniques the same thing applies for the all languages.

In this study discussed in detail about the refactoring and the code smell, how code smell was produced and what may be the possible solutions to remove or to fix these smells in the code. It is also discussed in detail that even if smell is found in the code it needs to refactor by keeping its behavior preserved which mean that program must behave the same way it works before implementation of the refactoring. Refactoring have impact on both logical and the syntax of the program. Hence it is necessary to make sure that the logic of the program should remains the same, but its syntax can be changed. With the implementation of refactoring it becomes easier to understand and to maintain the code of the program. About different techniques to code refactoring and smell detection are also discussed in the detail.

The most general type of refactoring includes to instantly check for the duplication of entity when new entity is just created which helps in reducing risk of code smell to be induced in the code and also prevents from any conflicts which may occurs in the program. This also helps to make the code maintainable and efficient and flawless to both syntax or logical errors which induces smells in the code and it also prevents from method overriding issues which can change the behavior of the program.

**Smell detection using the developed tool:** The tool developed in this research is basically for the detection of the code smell in the JavaScript code which facilitated the
developers to make their code smell free and to make it more efficient and maintainable for long term use. If the code is free from smells, then it also reduces the risk of pulling down the system and makes it more reliable for using. For the testing of the tool developed in the research have tested its performance by taking smelly sample of a code and analyzed it through the tool and after the analysis it prompts with the areas that contain smell in the code have analyzed. It runs on the browser and scans the code for developer with reporting the issues in the code which causes bad smell in the code. In the below code, scanned it through the tool which was developed for the detection of the code smell after passing the smelly code it erodes errors in the report of analysis on the code.

```javascript
(function () {
  var anObject = {};
  var anElem = document.getElementById("wrap");
  var aString = "This is a string"
  if (aString == "This is a string") {
    anotherString = aString;
  }
  function Person(name, age) {
    this.name = name;
    this.age = age;
  }
  aPerson = new Person("John", 25);
}());
```

After scanning the above code these below smells in the code are diagnosed, which are needed to be eliminated to make the code maintainable and efficient.

<table>
<thead>
<tr>
<th>Smell</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function needs a &quot;use strict&quot; pragma.</td>
<td>(function () {</td>
</tr>
</tbody>
</table>
| Expected ';' and instead saw 'if'. | var aString = "This is a string"
| Undeclared 'anotherString'. | anotherString = aString;                                                |
| Unexpected 'new'.              | var aPerson = new Person("John", 25);                                   |

After the removal of smell from the code as in the reports of the scanning of code, it becomes smell free and maintainable code with the help of developed tool.

```javascript
0 (function () {
1   var anObject = {};
2   var anElem = document.getElementById("wrap");
3   var aString = "This is a string"
4   if (aString === "This is a string") {
5     anotherString = aString;
6   }
7   function Person(name, age) {
8     this.name = name;
9     this.age = age;
10   }
11   aPerson = new Person("John", 25);
12 })();
```

Also implemented analysis on another piece of code of JavaScript as shown below, scanned it through the tool developed by us for the detection of the code smell after passing the smelly code it erodes errors in the report of analysis on the code.

```javascript
0 var p1, p2;
1 function myFunction(p1, p2) {
2   "use strict";
3   return p1 * p2;
4 }
5 function myFunction(p1, p2) {
6   "use strict";
7   return p1 * p2;
8 }
```

After scanning the above code these below smells in the code are diagnosed, which are needed to be eliminated to make the code maintainable and efficient.

```javascript
(function () {
  var p1, p2;
  function myFunction(p1, p2) {
    "use strict";
    return p1 * p2;
  }
})()
```

After scanning the above code it diagnosed that function/method is redefined which may cause overriding and the change in the behavior of the program. This system is simple and shabby amid programming advancement, however it is viewed as awful practice. It makes programming support more difficult from numerous points of view: First in case of existing bugs or blunders in already written code, they will be proliferated each time it gets duplicated. Secondly For the most part, when a case of copied code should be changed, all other copied cases should likewise be adjusted. Third onwards, code that is copied and pasted makes it harder to examine. Examining the code basically is examining the source code to eliminate vulnerabilities of the code it makes hard to find every time code is copied to the program.
After the removal of smell from the code as in the reports of the scanning of code, it becomes smell free and maintainable code with the help of this developed tool.

```javascript
0 var p1, p2;
1 function myFunction(p1, p2) {
2 const strict;  
3 return p1 * p2;
4 }
5 function myFunction(p1, p2) {
6 const strict;
7 return p1 * p2;
8 }
```

CONCLUSION
In this research a tool is developed to analyze the code and to suggest the refactoring to be finished in the code. This is required to make the code smell free and mainly developed for the JavaScript programmers because to find smell in JavaScript code is just by analyzing the code manually. This tool will facilitate the developer or programmer to do the maintainability of the code easily.

REFERENCES