Optimized Unit Testing for Antipattern Detection

Harvinder Kaur
University Institute of Engineering & Technology
Panjab University, Chandigarh
+919876264815
binny.mavi18@gmail.com

Puneet Jai Kaur
University Institute of Engineering & Technology
Panjab University, Chandigarh
+919914257047
puneetkaur79@yahoo.co.in

ABSTRACT
Antipatterns are poor design choices that are conjectured to make object oriented systems harder to maintain. We investigate the impact of antipatterns on classes in object-oriented systems by studying the relation between the presence of antipatterns and the change- and fault-proneness of the classes. Due to increased complexities in the software development, there is huge need of testing process to be carried on in better way. Also as the computer systems are significant to our society in everyday life and are performing an increasing number of critical tasks, so more work in software testing and analysis has become of great importance. Anti pattern testing is type of testing which is used to cut off the directly price associated with testing of different modules. This paper discusses various anti pattern detection techniques and proposes a new testing technique based on GUI for detection of anti patterns during software development.

Keywords
Maintenance Development, Antipattern, Software Development Techniques, Fault Loop, Loose Connectors

1. INTRODUCTION
TESTING is any type of software testing that seeks to uncover software errors by retesting a modified program. Testing is used to show that failures occur at an acceptable rate. Software technologies produces software based systems to grab and involve fresh industry needs and to provide quality and for bug fixing which is an important process in development. Due to the time-to-market, lack of understanding, and the developers' experience, developers cannot always follow standard designing and coding techniques, i.e., design patterns [1]. Test suites are often simply test cases that software engineers have previously developed, and that have been saved so that they can be used later to perform regression testing [2]. Re executing all the test cases require enormous amount of time and thus make the testing process inefficient. For example, one industrial collaborator reports that for one of its products of about 20,000 lines of code, the entire test suite requires seven weeks to run [3].So, in regression testing if the test cases that reveal the faults of output module execute first and test cases that reveals faults of input module executes later, then it will be delayed and in many cases will take long time to detect the original cause of output faults [4].

The main focus of this research is automation. Automated checking of testing criteria allows a high level of testing to be performed with the detection of a large number of bugs. Automation can help to manage the repetitive tasks.

2. PROPOSED WORK
Anti-pattern checking is an essential process in any software development procedure. Normally it is defined as recurring, bad designing in linking which could rise to the loose effects in software systems because it is a big difficulty in understandability and maintainability of whole development. Due to these issues, testing of the systems is increased and hence the cost increase too. In related research the testing cost is considered as the number of test cases that satisfy the minimal data member usage matrix (MaDUM) and studied four Java programs, Ant 1.8.3, ArgoUML 0.20, CheckStyle 4.0, and JFreeChart 1.0.13 which shows that unit testing is increased due to availability of anti-patterns. Previous research also introduced some refactoring actions which when applied to the classes participating in anti-patterns reduce cost of testing. In the proposed work, the test cases are enhanced for the similar work under MaDUM [2] with eclipse, Web browser as additional Java programs.

Effective testing of software is necessary to produce reliable systems. This is true in practice since static verification techniques have their own limitations. The automation testing for the Graphical User Interface is performed. The first phase includes connecting the path of the given GUI and the XMI based on ArgoUML tool. Unified modeling language is used for creating XMI files. The GUI contains a set of events, if any of the events is clicked than the flag value will be set. Now the GUI file is converted to the XMI code with all the flag set value for finding of anti-pattern credentials. Eclipse and web browsers will be act as input set for Matrix data member for finding the changes which affects classes participating in anti-patterns. Automation in testing for finding anti-pattern and automation of refactoring are the primary concerns which could be fulfill by judging the parameters which are responsible for anti-pattern and by providing refactoring of this, anti-patterns could be avoided.

This research is focused on developing a language code (probably in java) and then to test it for errors. Total testing process will be done on it. Once done, the next step is to find the test cases for

ICTCS '14, November 14 - 16 2014, Udaipur, Rajasthan, India
Copyright 2014 ACM 978-1-4503-3216-3/14/11…$15.00
http://dx.doi.org/10.1145/2677855.2677909

complete code which will be used as test case in further testing. The testing resources for anti-pattern detection can be reduced by reducing the test cases to get reduced test suites for testing.

The concept is based on the automatic detection of anti-patterns in the given code. Anti-patterns are wrong design patterns that deteriorate the quality of the software. Detection and removal of anti-patterns can enhance the software quality and will decrease the chances of the bugs in the software. If we may automate the process of anti-patterns detection in the code, it will reduce the time and cost of anti-pattern detection.

So the goal is to automate the process of anti-pattern detection. The technique used for the proposed model is shown in figure 1.

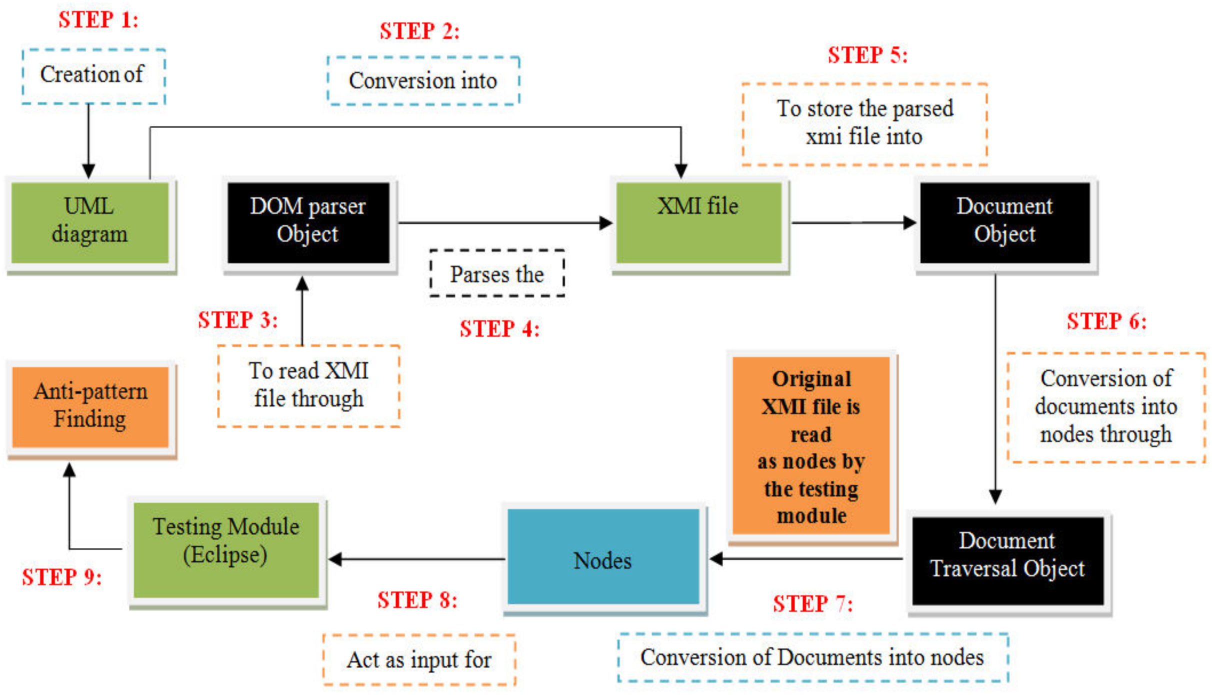


Figure 1. Technique used for the proposed model

3. OBJECTIVES

The research starts with study of anti pattern detection mechanisms like SMURF [5] and DÉCOR [6]. The processing of detection systems is done to find techniques with better accuracy and quality.

Some of the objectives which need to be fulfilled are given below:

- Find optimized testing technique for detection of anti patterns.
- Providing solution for anti-patterns by testing approaches.

For fulfilling the above mentioned objectives, simple testing strategies such as testing of whole code without use of any test case were used. After that, simple test case scenarios were found which save resources and improve efficiency of bug finding and for detection of anti-patterns also. Finally for providing better test case selection process, the set of test cases created so far have been processed with solutions for anti patterns. In this paper, the refactoring of detected anti patterns has been accomplished for providing solution through testing.

4. RELATED WORK

This section reviews related literature pertaining to Anti-patterns and their detection, testability of classes and unit testing approaches in Object-oriented systems.

Travassos et al. [7] established a manual technique in which design smells were not specified. This technique makes provision for manual evaluation and reading strategies to carry out the process of identification of design smells. Automation is not possible with this approach and therefore, applicability to extensive systems is not supported.

Dhambri et al. [8] (Manual approach) and Simon et al. [9] proposed some visualisation techniques to determine a trade-off between manual inspections (time-consuming and non-representative) and fully automated approaches (systematic and productive).

Munro [10] used metric-based heuristics for the identification of anti-patterns. A template is also proposed by him for the systematic characterization of code smells, so that limitation of text-based descriptions can be avoided. The template is composed of three things: name of code smell, text-based descriptions of its attributes, and heuristics to detect them.

Lanza and Marinescu [11] and van Emden and Moonen [12] performed fully automated detection of anti-patterns but they do not tackle uncertainty and long lists. Their methodologies represented the results of detection through visualisation techniques. The quality analysts’ interpretation and thresholds are not taken care of while implementing this approach.

Moha et al. [6] developed a plan for the conversion of rule cards into detection algorithms automatically [13]. They also proposed some algorithms and brought in a DSL to describe the behavior of each anti-pattern on the basis of set of rules.

Khomh et al. [14] formulated an anti-pattern detection technique that relies upon Bayesian Belief Networks (BBNs).This technique is a Goal Question Metric (GQM) based BDTEX (Bayesian Detection Expert) approach. This approach is considered to be highly unreliable to uncover the existing anti-patterns. In this type of approach, BBNs are formed by taking into account the definition of anti-patterns.

Maiga et.al [5] proposed SMURF for anti-pattern detection. This approach is based on the polynomial kernel which uses a machine learning technique. The main objective of their research is to model a technique that is suitable for both inter and intra system framework. This approach allows for practitioners' feedback. This incremental methodology culminated in more desirable precision and recall for both type of configuration- inter and intra. The technique also reported that the practitioners' feedback plays an important role in improving the efficacy of this approach.

5. EXPERIMENTATION

The proposed work is started and implemented with the implementation of anti-pattern detection mechanism. The working of the program can be divided into two phases:

- **In first phase**, XMI file is read and the required information is collected from it.
- **In second phase**, the collected information is made use of and a java based tool will use this information to test the anti-patterns in the given code.

The application used in the study is related to conversion of numerals from feet to inches, inches to feet and centimeter to inches. An additional feature added to the program is uploading a text file and converting the values in it. For easy interpretation of proposed work, graphical user interface is created which is shown in figure 2 below.

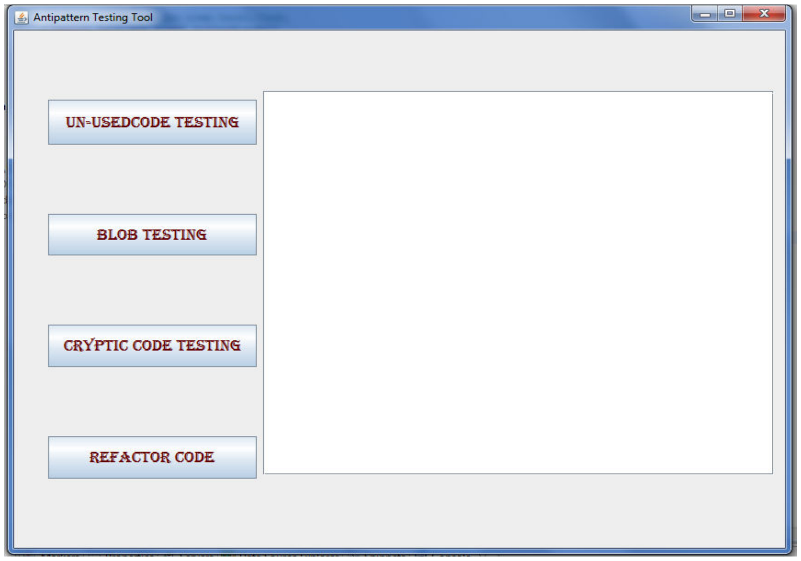


Figure 2. Graphical User Interface for automated anti pattern detection

First, UML diagram is created for the program using ArgoUML. Generated UML is then converted into XMI, this XMI is used as an input for eclipse testing module.xmi file path should be given for exact result fetching. The next step is to find the nodes that contain class name and to get the information about the field of these classes. Once the required input has been provided, the program can be run. The program then reads the xmi file from the specified location. To read xmi, a DOM Parser object is created that is used to parse the file. After parsing, the file is stored in a Document object in the form of document. A Document Traversal object is used to convert Document (xmi file is converted into document) into nodes. Each node has a name and some attributes. These node names and attributes are used to get required information.

The GUI for fetching input file and processing is shown in figure 3.

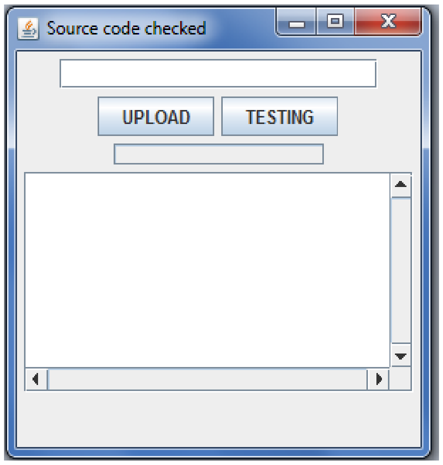


Figure 3. GUI for getting input file for testing

Now a node is picked one by one i.e. one node at a time is picked till all nodes are done. From the previous step, we will get Node name and node attributes. After that it will be checked if a node is specifying a class, then all the following nodes that specifies class attributes will be stored in a list corresponding to that class. These lists of attributes will be used to search the attribute which is specifying the information about a field in the program.

After getting the required information from the XMI, the java based eclipse testing module will analyze the main code and find the antipatterns in it. In the developed GUI application, the input for the conversion system can also be taken manually. The GUI for manual input is shown in figure 4.

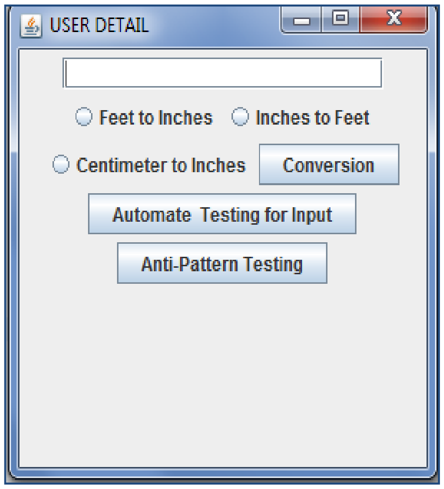


Figure 4. **GUI for getting manual input for testing**

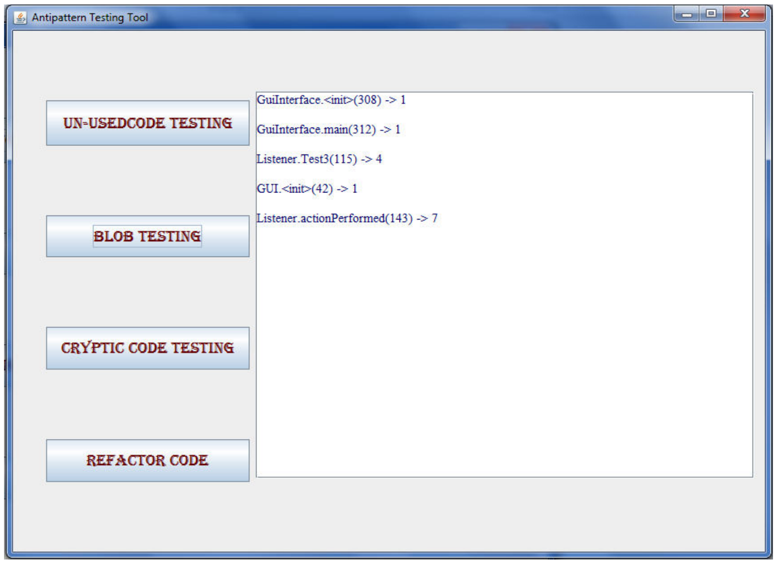
There are different types of anti-pattern, and in this research, the most common anti-patterns are detected in the code i.e. unused data, blob and cryptic code anti-pattern that affects the quality of the code.

- **BLOB:** Blob is a type of anti-pattern in which a single function defined in an application performs multiple functions. It is also known as God object. To find blob anti-pattern, the tool use to print a method named 'call stack'. Method 'call stack' will display the calling scenario in the program. Mostly it is expected that one method should perform one job/function. If some method is called over and over again, it may be a blob design. So by inspecting the call stack the occurrence of blob can be detected in the program to be tested.
- **CRYPTIC CODE:** Cryptic code is another type of anti-pattern which defines the abbreviations used for fields instead of proper naming. These cryptic fields make it hard to understand what type of values the field is simulating. This makes it difficult and cumbersome to reuse the code or modify the code in future. To find cryptic code, the tool will define the minimum length required for a perfect name of field. The fields having character less than the specified length is considered as cryptic code and is added to the list which prints them after searching through the whole program.
- **UNUSED CODE**: Unused code is similar to dead code. It is also popular from the name of 'Lava Flow'. The systems which emerged as research but later on turned out as production comprises these types of anti-patterns. This type of anti-pattern is an outcome mainly when the developers are in research stage of practicing various approaches to carry out things. It can also be called as 'abandoned design information' which is not responsible for any final outcome of the system but degrades the performance. Or we can say that any information which is involved in design but not contributing to performance in a positive way. So there is need to re-factor such kind of anti-pattern. In the proposed technique, a management process identifies the unused code and wipe it out to simplify the design and for better performance of the system.

A tool is created that automate the process of finding above mentioned anti-patterns.

The information which has been obtained from XMI file is used for the detection of unused code and cryptic code. Field defined but not used during the execution of the code can be considered as unused code. Figure 5, showing the Blob testing module.

These type of field's increases program execution time and consume unnecessary memory space which may reduce the quality of the code. Data gathered from XMI file provided us with the information about the fields of the class. The defined program use to search these fields to find their values, if field is not used during the execution of program, that field is marked as unused.

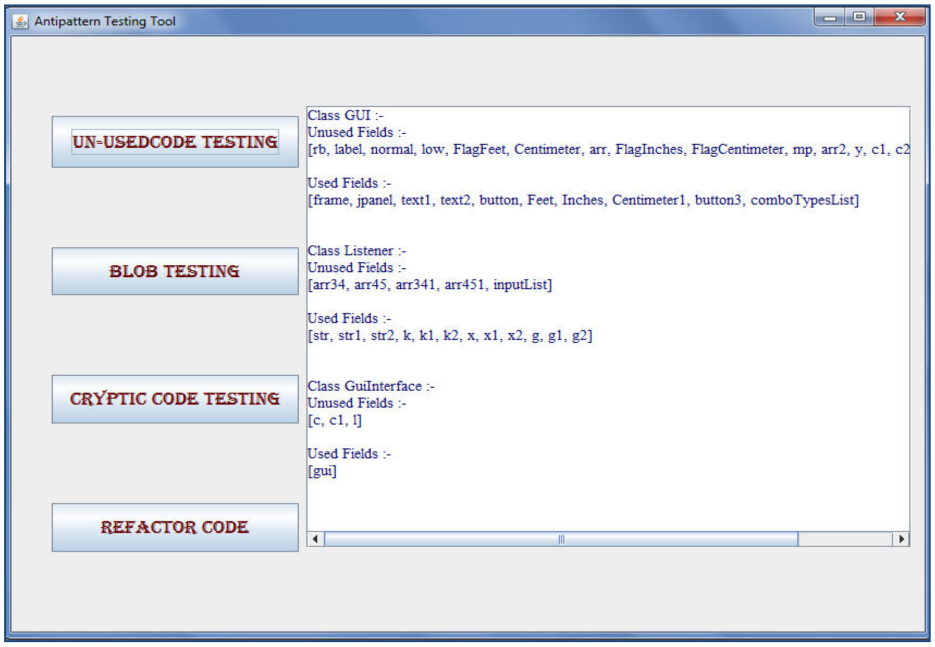


```
GuiInterface.<init>(308) -> 1

GuiInterface.main(312) -> 1

Listener.Test3(115) -> 4

GUI.<init>(42) -> 1

Listener.actionPerformed(143) -> 7
```

Figure 5. Testing module for BLOB detection

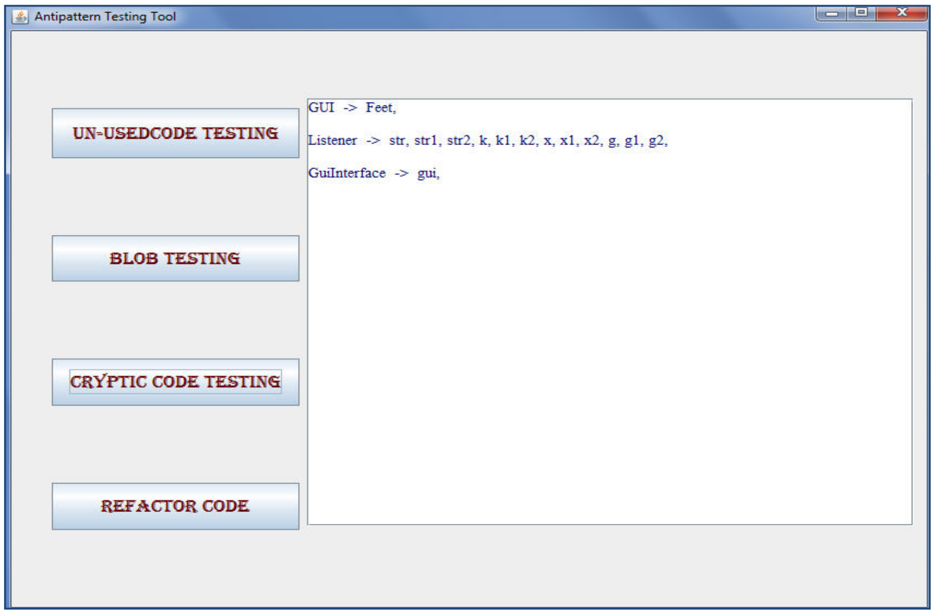
After searching through the whole program to be tested, it will provide the list of all unused fields in the last.

After finding the anti-patterns, the code is re-factored to remove the anti-patterns from the code.



```
Class GUI :-
Unused Fields :-
[rb, label, normal, low, FlagFeet, Centimeter, arr, FlagInches, FlagCentimeter, mp, arr2, y, c1, c2

Used Fields :-
[frame, jpanel, text1, text2, button, Feet, Inches, Centimeter1, button3, comboTypesList]


Class Listener :-
Unused Fields :-
[arr34, arr45, arr341, arr451, inputList]

Used Fields :-
[str, str1, str2, k, k1, k2, x, x1, x2, g, g1, g2]


Class GuiInterface :-
Unused Fields :-
[c, c1, l]

Used Fields :-
[gui]
```

Figure 6. Testing module for Unused Code detection



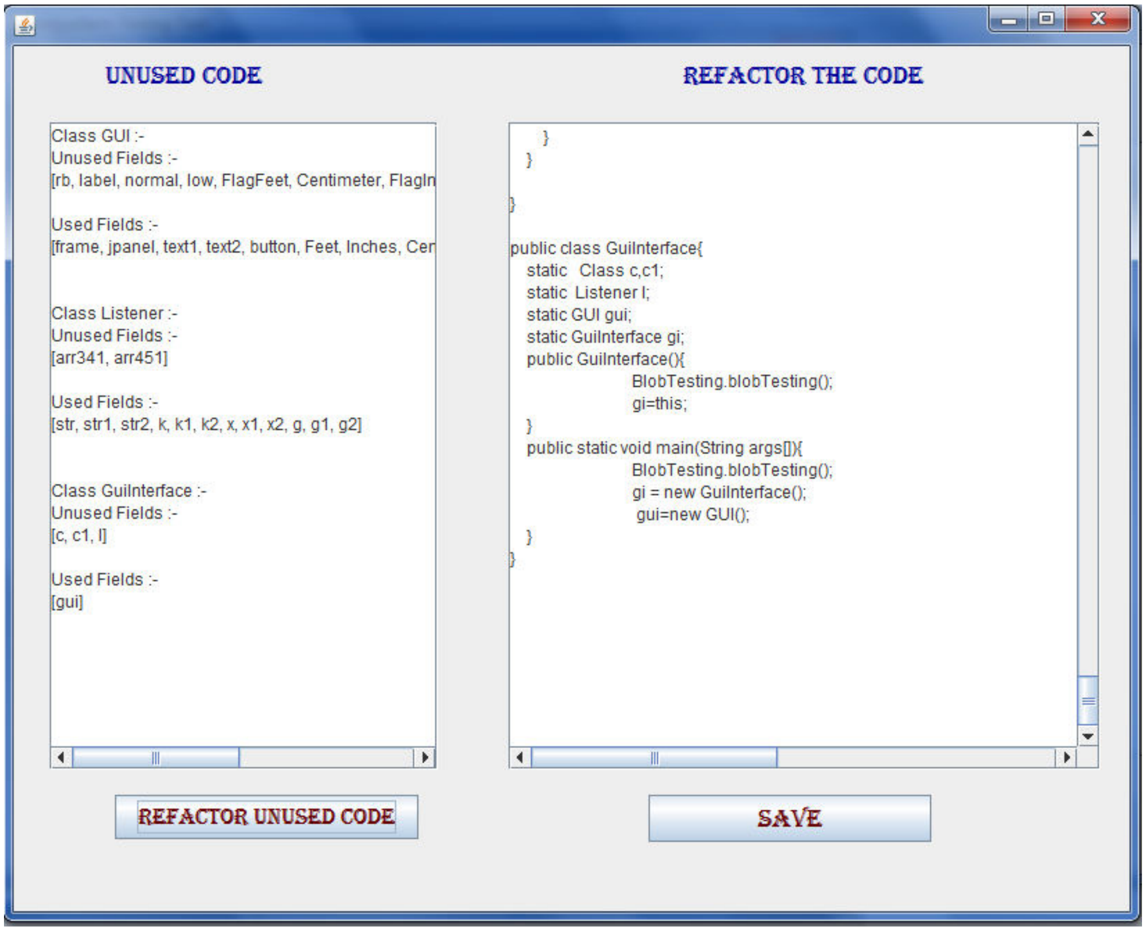
```
GUI  ->  Feet,

Listener  ->  str, str1, str2, k, k1, k2, x, x1, x2, g, g1, g2,

GuiInterface  ->  gui,
```

Figure 7. Testing module for Cryptic Code detection

The testing module for the detection of Unused Code anti-pattern is shown in figure 6.All the fields shown in the figure are unused fields of different classes. And figure 7 shows the cryptic code anti-pattern detected using GUI application. In this paper, three anti-patterns have been identified. These anti-patterns can be refactored to improve the quality of design and code.

Figure 8, showing anti pattern refactoring process.

This made the execution of the code faster and enhanced the quality of the code. Refactoring of the code is carried out in three different phases. In each phase, one of the anti-patterns is removed from the code.



```
Class GUI :-
Unused Fields :-
[rb, label, normal, low, FlagFeet, Centimeter, FlagIn

Used Fields :-
[frame, jpanel, text1, text2, button, Feet, Inches, Cer


Class Listener :-
Unused Fields :-
[arr341, arr451]

Used Fields :-
[str, str1, str2, k, k1, k2, x, x1, x2, g, g1, g2]


Class GuiInterface :-
Unused Fields :-
[c, c1, l]

Used Fields :-
[gui]
```

```
    }
  }

}


public class GuiInterface{
  static  Class c,c1;
  static  Listener l;
  static GUI gui;
  static GuiInterface gi;
  public GuiInterface(){
                  BlobTesting.blobTesting();
                  gi=this;
  }
  public static void main(String args[]){
                  BlobTesting.blobTesting();
                  gi = new GuiInterface();
                   gui=new GUI();
  }
}
```

Figure 8. Testing module for refactoring of detected anti patterns

After the completion of refactoring process, re-factored code is passed to the testing tool to check if still there is any anti-pattern left. At the end, the execution time and the memory space consumed by the re-factored and non-refactored code is compared.

6. RESULTS AND DISCUSSION

The main cause for the development of the proposed technique is to contribute to the automated detection of anti-patterns through testing process. The primary concern of detecting anti-patterns through testing process is to save time and effort required for this process and to find as much bugs as possible from restrained resources. A good design will have lesser execution time in comparison to a bad design i.e. anti-patterns. For the developed application, the performance of refactored and non-refactored design has been compared in terms of memory space and execution time.

6.1 Advantages of Automated Anti-pattern Identification

The GUI based technique proposed in this paper is fully automated for the detection of anti-patterns. The user needs to input values either by uploading a file or manually entering the input values. Though the input is both manual and automated, but the detection process is automated. Some of the benefits of using this technique are:

- Simple and Convenient to use
- User Friendly
- Automated
- Requires no manual effort for detection
- Can be extended in order to be applicable to other well-known anti-patterns.

The refactoring of anti-patterns is semi-automated in developed application i.e. it feels necessity for a little manual effort. Only three anti-patterns have been identified in the implemented work: Blob, Unused Code and Cryptic Code .But this work can be extended so that other well known anti-patterns can be identified.

6.2 Comparison of Refactored and Non-Refactored Application

The refactored and non-refactored code for the developed application is analyzed in terms of memory space consumed during conversion process and the execution time required to run the applications.

6.2.1 On the basis of Execution Time

6.2.1.1 For Manual Input

The execution time for the anti-pattern application (non-refactored) and the refactored application is analyzed and the comparison is shown above in figure 9. The execution time shown above in the figure 9 is measured in nanoseconds for both the mentioned applications. In this figure, the input is taken by the user in the manual form.

As we can see, the execution time is significantly less for the refactored application than the non-refactored one. This makes it refactoring, a fundamental necessity for the better performance of the system. Therefore, refactoring of anti-pattern application is essential.

6.2.1.2 For File Input

The input values can also be uploaded from a file. Through file uploading option, we can convert more than one value at the same time and further identify anti-patterns. In this type of input, the process again works in the same way.

The comparison of execution time for the anti-pattern application and its refactored version in case of file input is shown in figure 10.The results for the comparison of execution time for both the applications conclude in the same way as for the manual input of values i.e. refactored application is executed faster. Figure 10 shows that the execution time for refactored application is low in comparison to the non-refactored application. From these results, we conclude that the refactored application is more efficient than non- refactored one in terms of execution time.

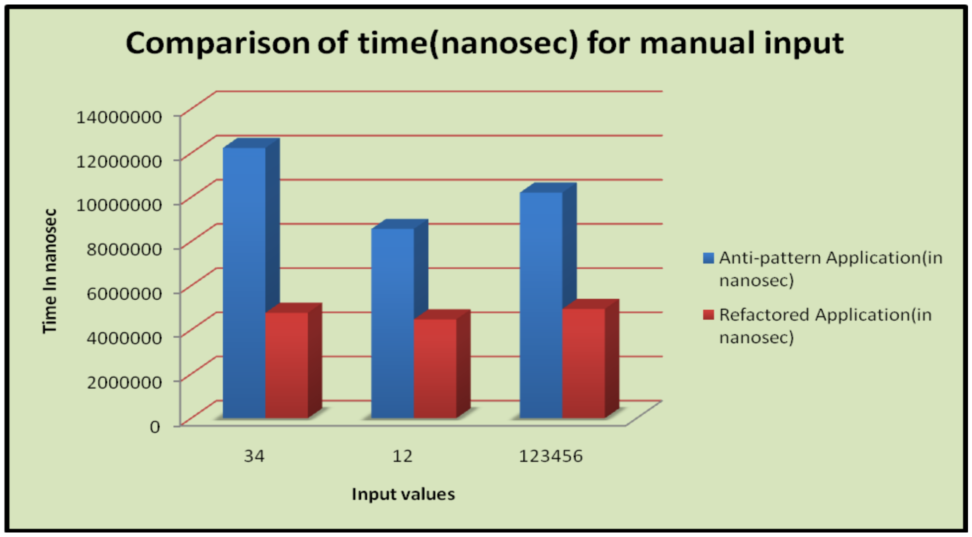


Figure 9. Time Comparison for the manual input

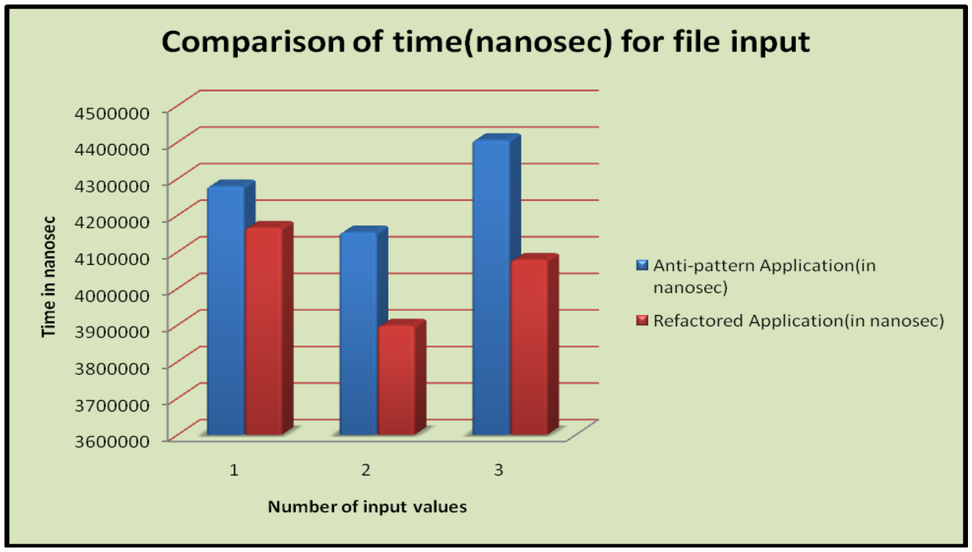


Figure 10. Time Comparison for the file input

6.2.2 *On the basis of Memory Space used*

For the developed application, the performance of re-factored and the non-refactored versions can also be compared on the basis of memory space used during conversion process.

Figure 11 shows the comparison of memory usage (in bytes) by the anti-pattern and its refactored application during the conversion process.

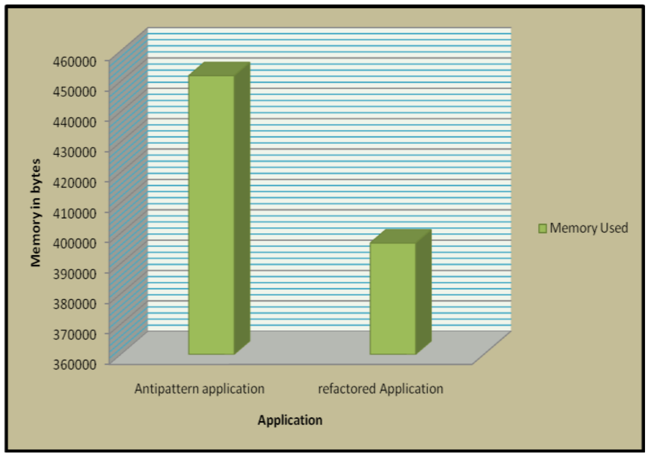


Figure 11. **Memory Usage Comparison**

From results on comparison of memory usage, we conclude that refactored version of anti-pattern application consumes much lesser memory space than the original designed anti-pattern application.

So by refactoring the object-oriented applications we can get better results in comparison to non-refactored or antipattern application. There are many advantages of refactoring:

- Better performance
- Better management of data
- Improved design
- Less errors
- More testability

7. CONCLUSION AND FUTURE SCOPE

This paper proposed a process for anti-pattern detection automatically and provided refactoring for detected anti- pattern designs. AgroUML tool have been used for testing unified modeling language diagram bugs. The target for research is to have robust anti pattern detection system with unit testing process and is shown in terms of cryptic testing and refactoring for these testing. The anti-patterns are detected using the eclipse tool with Java programs and ArgoUML.

In future work, the inputs, outputs, processes, implementations and our detection technique can be postulated to other anti-patterns. We have not analyzed the comparison of our implemented technique with other procedures available for anti-pattern detection, but will make the comparison in future while extending the work done so far. The proposed process has been practiced with one domain only i.e. Object Oriented Domain, therefore this work can be extended in future using other domains with larger sets of projects.

The test cases for testing process of anti-pattern identification will be enhanced. We will be validating the proposed method in future using higher projects. In this paper, only three types of anti-patterns have been identified which are namely: Blob, Cryptic Code and Unused Code. This work can also be used to detect other well known anti-patterns and therefore the results can be validated.

8. REFERENCES

[1] Fowler, M.1999.*Refactoring – Improving the Design of Existing Code*.1st edn. Addison-Wesley.

[2] Sabane A., Penta M. D., Antoniol G., Gueheneuc Y. –G. 2013.A Study on the Relation Between Antipatterns and the Cost of Class Unit Testing.In *Proceedings of the 17th European Conference on Software Maintenance and Reengineering.* IEEE Computer Society (July 2013)

[3] Chidamber S. R.1994.A metrics suite for object oriented design. *IEEE Trans. Softw. Eng*.

[4] Kayes M. I.2012.Test Case Prioritization for Regression Testing Based on Fault Dependency.In *Proceedings of the IEEE International Conference of Software Engineering*. IEEE Computer Society (Feb. 2012)

[5] Maiga A. et al.2012.SMURF: a SVM based incremental anti-pattern detection approach.In *Proceedings of the 19th Working Conference on Reverse Engineering (WCRE)*. IEEE Computer Society Press (Oct. 2012)

[6] Moha N., Guéhéneuc Y.-G., Duchien L., Meur A.-F.L.2010. DECOR: a method for the specification and detection of code and design smells.*IEEE Transactions on Software Engineering(2010a)*, vol. 36, no.1, pp. 20–36(Jan.-Feb. 2010)

[7] Travassos G., Shull F., Fredericks M., Basili V.R.1999. Detecting defects in object-oriented designs: using reading techniques to increase software quality.In *Proceedings of the 14th Conference on Object-Oriented Programming, Systems, Languages, and Applications*.ACM Press, pp. 47–56 (Oct. 1,1999)

[8] Dhambri K., Sahraoui H., Poulin P.2008.Visual detection of design anomalies.In *Proceedings of the 12th European Conference on Software Maintenance and Reengineering*.IEEE Computer Society, pp. 279–283.

[9] Simon F., Steinbrückner F., Lewerentz C.2001.Metrics based refactoring.In *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR'01)*. IEEE Computer Society , pp. 30-38.

[10] Munro M.J.2005.Product metrics for automatic identification of "bad smell" design problems in java source-code.In *Proceedings of the 11th International Software Metrics Symposium*.IEEE Computer Society Press, pp.15 (Sept. 19-22, 2005)

[11] Lanza M., Marinescu R..2006.*Object-Oriented Metrics in Practice*. Springer Berlin Heidelberg.

[12] van Emden E., Moonen L.2002.Java quality assurance by detecting code smells.In *Proceedings of the 9th Working Conference on Reverse Engineering (WCRE'02)*.IEEE Computer Society Press.

[13] Moha N., Guéhéneuc Y.-G., Meur A.-F.L., Duchien L., Alban Tiberghien.2010.From a domain analysis to the specification and detection of code and design smells. *Formal Aspects of Computing (FAC)*, vol. 22, no. 3-4, 2010b, pp. 345-361.

[14] Khomh F., Vaucher S., Gu´eh´eneuc Y.-G., and Sahraoui H.2011.Bdtex: A gqm-based bayesian approach for the detection of antipatterns.*J. Syst. Softw.*, vol. 84, no. 4, pp. 559–572 (Apr.2011)