IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-6, Issue-7, July 2018 E-ISSN: 2347-2693

Testing Refactoring Implementations of Object-Oriented Systems

B. Nagaveni[1*], A. Ananda Rao[2], P. Radhika Raju [3]

[1*]Department of CSE, JNTUA college of engineering, Ananthapuramu, Andhra Pradesh, India.
[2]Department of CSE, JNTUA college of engineering, Ananthapuramu, Andhra Pradesh, India.
[3]Department of CSE, JNTUA college of engineering, Ananthapuramu, Andhra Pradesh, India.

Corresponding Author: nagaveni.buruga@gmail.com, Tel.: 9985234692

Available online at: www.ijcseonline.org

Accepted: 21/Jul/2018, Published: 31/July/2018

Abstract— Testing the refactoring as for formal semantics is dealt with as a test. Refactoring engines like Eclipse, Netbeans and many other contains various kinds of refactoring techniques like move, inline, copy, extract method etc. Usually, developers used to write the test cases to control their refactoring implementations. Few automated testing techniques are used for testing the refactoring implementations of object-oriented systems. In existing, the pre-conditions are recognized and stated that they are extremely stable. In earlier, the testing is done in JRRT (jastAdd refactoring tool) using the process of Alloy Analyzer and the JDolly technique. Using the similar process and techniques, the proposed work makes, testing on the refactoring implementations in the Netbeans Refactoring Engine. Creating the meta-models using the Alloy Analyzer and generating the programs from the model by using the JDolly program generator for applying the testing concept on it. Test Oracles are involved to retain the nature of the programs after implementing the refactoring concept.

Keywords— Alloy meta-models, Automated testing, Program generation, Refactoring implementations.

I. INTRODUCTION

Refactoring is the foremost programming tool of the recent object-oriented techniques that can be formalized and used to develop the program. Refactoring is an exact control approach to upgrade the plan of existing code. With the help of a refactoring concept, the total design and structure of an existing program are improved, at the same time its functionality remains unchanged. Once the outline has been overhauled, it will be simpler to proceed [1, 2, 3].

For example, to Pull-up the method from the sub class to a super class, there is a need to contain at least two classes in a program.

Generally, few Refactoring Engines are used to automate the application of refactoring. The testing process of refactoring techniques can be done through the usage of Alloy meta-models and generated programs from the JDolly.

The Refactoring Tests are tested with the JUnit testing tool for checking whether the tests are passed or not. So by using that JUnit tool the research is done and the tests are successfully passed. Each test execution or passed time will also be notified in the project.

II. RELATED WORK

Alloy is a declarative specification language, and is used in the Alloy Analyzer tool [4]. The JDolly produces a massive representation of programs according to the specified scope of a program. The additional rules can also be considered for guiding the program generation. The testing can be concluded through the oracles. By using the unit testing in terms of object – oriented systems, with this kind of process the refactoring functionality is tested highly.

III. PROPOSED MODEL

A. Alloy and Alloy Analyzer

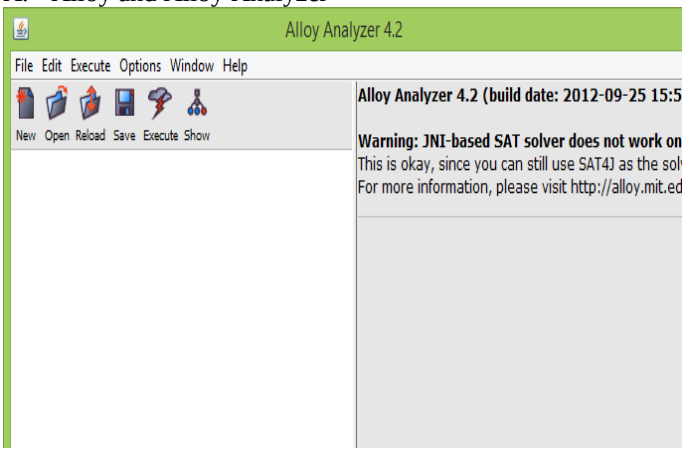

Figure 1. Alloy Analyzer

An alloy is addressed due to the creation of micro-models that can be used to check automatically for correctness. The Alloy Analyzer device allows us to do examination on Alloy determinations [4].

B. *Alloy Meta-model*

Alloy meta-model is encoded in the Alloy. The large numbers of well-formed rules are mentioned in java as shown in Figure 2. These can be used for creation of an instance. It can allow a various kinds of requirements for associating in to a program.

```
sig Type{}
sig Classs extends Type{
extend: lone Class,
methods: set Method,
fields: set Field
}
Sig Method{}
Sig Field {}
```

Figure.2 Alloy Rules

```
One sig C1,C2 extends Class {}
One sig m1,m2 extends Method {}
Pred generate [ ] {
C1 in C2.extend
M1 in C2.methods
}
```

Figure.3 Alloy program

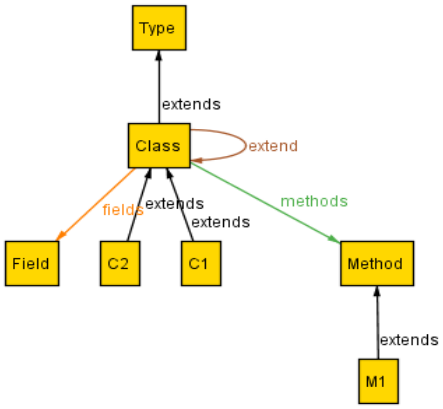

Figure 4. Alloy Meta-model

Figure 3 shows the particulars of a Alloy model respectively. This information is most important to achieve the restructuring of a code.

For the above mentioned program in Figure 3, the Alloy Meta-model will be as shown in Figure 4.

C. *JDolly*

JDolly is a program generator, it intensely allows programs, for the inclined scope [6]. According to the scope and rules JDolly translated the Alloy models in to java programs. It discovers a proper occurrence from the Alloy specifications, to satisfy the rules with in a specified limit. It configures few necessary parameters for generating java programs. Using skips, developers can identify the bugs. The extension is about the quantity of bundles, classes, techniques and fields in the projects. The additional rule parameter is used to develop the particular type of action to improve the structure.

D. *Program Generation*

In the Figure 4, to utilize the enclose reusable formulas and determine operations the predicates (pred) are used. It uses at most three objects as default scope for each signature [7]. The user can also use various scopes for individual signatures. The below Alloy fragment represents the run and predicate using the scope of 3.

```
pred generate [] {}
run generate for 3
```

Figure.5 Alloy Fragment

A Java Alloy Analyzer API is used to find every possible solution. For each generated Alloy solution, a java program is mapped with their equivalent java abstract syntax. The reuse of Java abstract syntax tree is used for program generation.

E. *Assertions*

In the context of object-oriented software systems the assertions are the formal rules. Related to software systems source program, behaviours of assertions are inserted as an annotation. Assertions can be clearly explained through JUnit. Assertion method is used to regulate the test case status, whether it is right or wrong.

A class named Assert was provided by the JUnit to use a group of assertion methods, which are used in writing the test cases and detecting the test failures [8] . The annotations are @Test, @Before, @After, @Before Class, @After Class, @Ignore. These can be used according to their specifications respectively [9]. The assert Equals method is required to

generate the number of programs in terms of specifying the programs number.

F. *Oracles*

Oracles are used to figure out the correctness of the refactoring transformations. Based on this functionality, the implementations are tested for the correctness of the program and also evaluated the output of each transformation [10].

Framework

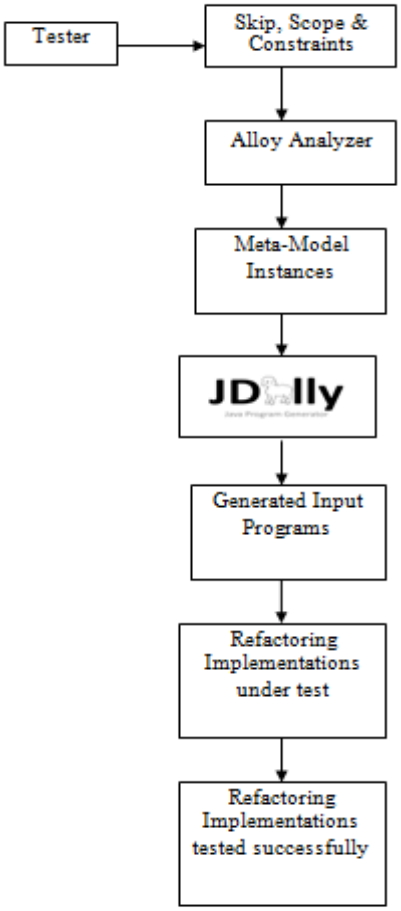

Figure 6. Process to test refactoring implementations

The framework represented in Figure 6 explains the process of testing refactoring implementations in Netbeans. The tester tests the entire process according to the above represented framework. After the usage of all rules the programs are generated and these generated programs are tested according to the Junit test process.

IV. IMPLEMENTATION

In this section, different types of refactoring implementations are tested by using the JDolly [11]. A few Refactoring Implementations are tested in the Netbeans Refactoring engine, by considering the process shown in Figure 6.

A. *Rename*

To evaluate this, refactoring can apply on package, class, method, and field. Once the change is happened, it will be updated in the entire source code of a project.

B. *Pull-up*

The programs must declare a Method/ Field in a sub class in order to test the Pull-up Method/ Field.

C. *Push-down*

The programs must declare a Method/ Field in a sub class in order to test the Pull-up Method/ Field.

D. *Encapsulation*

To test the Encapsulate Field refactoring, the programs must maintain at least one public method and field respectively.

E. *Move*

Move Method refactoring can be tested by the program containing with two classes. One of the classes must have a field and a method of the equivalent type of the other class.

Few other refactoring implementations are tested according to their specifications respectively.

Programs shown in Figure 7 and Figure 8, performs the re-designing concept for one of the implementation. Programs mentioned in the below are generated from the JDolly program generator.

```
package Package_0;
public class ClassId_1 {
}
public class ClassId_0 extends ClassId_1 {
 public long m_0(){
  return new ClassId_0().methodid_1(2);
}
}
```

Figure 7. Before applying Pull-up method

```
package Package_0;
public class ClassId_1 {
 public long m_0(){
  return newClassId_0().methodid_1(2);
 }
}
public class ClassId_0 extends ClassId_1 {
}
```

Figure 8. After applying Pull-up method

For these programs the transformation is applied and declared that the Pull-up method preserves the program

behavior and converted the refactoring transformation clearly [12].

V. EXPERIMENTAL RESULTS

The Table 1 contains the clear information i.e., the input given to the peculiar refactoring implementation and how it generate the programs. According to their specifications scope is also represented for all refactoring implementations. The count of generated programs is mentioned.

GP = Generated programs;

Skip = Reduces the testing time;

Working with the JDolly, the programs are generated as, for the Rename class 15916, Rename method 11263 and Rename field 19424 programs.

The Add parameter and Encapsulation field generated the 30186 and 2000 programs respectively. The Pull-up method and field generated programs are 11709 and 10927.

The 26348 and 11936 programs are generated by the other techniques. A Move method generated the 22905 number of programs according to its requirements. The skip number 25 is considered to reduce the testing time of the refactoring implementations [13, 14].

The other different methods accomplished with the 20.062s and 11.672s. The another method tests done with the 27.206s and 13.625s respectively. All the refactoring implementation tests are passed and executed in Netbeans refactoring engine successfully.

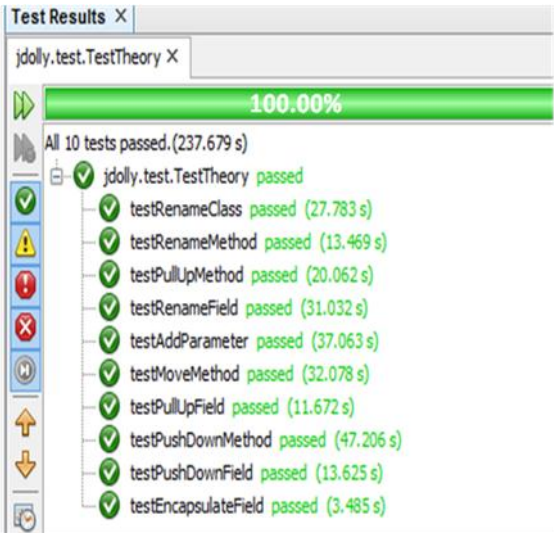

Figure 9. Test Results.

In the Figure 9, the Test Results are shown as per the specified requirements. By considering the JDolly program generator the programs are generated and applying those generated programs for testing the Refactoring Implementations.

Table 1. Evaluation of Refactoring Implementations.

Refactoring Implementations	Skip	Scope				GP	Time(sec)
		P	C	M	F		
Rename Class	**25**		**2**	**3**	**3**	**15916**	**28**
Rename Method	**25**		**2**	**3**	**3**	**11263**	**13**
Rename Field	**25**	**2**	**3**	**1**	**2**	**19424**	**31**
Add Parameter	**25**		**2**	**3**	**3**	**30186**	**37**
Pull-up Method	**25**		**2**	**3**	**4**	**11709**	**20**
Pull-up Field	**25**	**2**	**3**	**2**	**1**	**10927**	**11**
Encapsulation Field	**25**	**2**	**3**	**3**	**1**	**2000**	**3**
Move method	**25**	**2**	**3**	**3**	**1**	**22905**	**32**
Pushdown Method	**25**		**2**	**3**	**4**	**26348**	**41**
Pushdown Field	**25**	**2**	**3**	**1**	**2**	**11936**	**13**

VI. CONCLUSION and Future Scope

This proposed work is a way to test Refactoring Implementations in Netbeans refactoring engine. With the help of Alloy Analyzer and the JDolly program generator, 19580 java programs are generated. Few refactoring implementations such as rename, pull-up, pushdown, encapsulation, add parameter and move method are tested successfully and their behaviour transformation is checked with oracles. Using oracles the refactoring transformations are tested correctly.

In future the work can be extended to test the refactoring implementations using the other program generator tools other than JDolly. And also the JDolly can be improved to increase the possibility of testing refactoring techniques in object- oriented system concepts.

REFERENCES

[1] M. Schafer and O.de Moor, "Specifying and implementing refactorings," in proceedings of the 25th ACM International Conference on Object-Oriented Programming, Systems, Languages, and Applications, ser. OOPSLA' 10. ACM, 2010, pp. 286-3-1.

[2] W.F.O pdyke, "Refactoring Object-Oriented Frameworks,"Ph.D. dissertation, Univ. of Illinois at Urbana-Champaign, 1992.

[3] Melina Mongiovi Member, IEEE, Rohit Gheyi, Gustavo Soares, Márcio Ribeiro, Paulo Borba and Leopoldo Teixeira" Detecting overly strong preconditions in refactoring engines", IEEE Transactions on Software Engineering *DOI 10.1109, 2007*.
[4] G. Soares, R. Gheyi, D. Serey, and T. Massoni, "Making program refactoring safer," IEEE Software, vol. 27, pp. 52–57, 2010.
[5] D. Jackson, I. Schechter, and H. Shlyahter, "Alcoa: the Alloy constraint analyzer," in Proceedings of the 31st International Conference on Software Engineering, ser. ICSE '00. IEEE Computer Society, 2000, pp. 730–733.
[6] G. Soares, M. Mongiovi, and R. Gheyi, "Identifying overly strong conditions in refactoring implementations," in Proceedings of the 27th IEEE International Conference on Software Maintenance, ser. ICSM '11. IEEE Computer Society, 2011, pp. 173–182.
[7] D. Soares, R. Gheyi, and T. Massoni, "Automated behavioral testing of refactoring engines," IEEE Transactions on Software Engineering, vol. 39, pp. 147–162, 2013.
[8] Liangliang kong "Essential of unit testing tool for special testing."IEEE xplore, 2008.
[9] D. S. Rosenblum, "A Practical Approach to Programming with Assertions", IEEE transactions on software engineering, vol21(1),1995,pp.1
[10] M. Mongiovi, G. Mendes, R. Gheyi, G. Soares, and M. Ribeiro, "Scaling testing of refactoring engines," in Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution, ser. ICSME '14. IEEE Computer Society, 2014, pp. 371–380.
[11] Brett Daniel Danny Dig Kely Garcia Darko Marinov "Automated Testing of Eclipse and NetBeans Refactoring Tools"
[12] Narendar Reddy kancharla, Ananda Rao Akepogu, Gopi chand merugu, Kiran kumar Jogu " A Quantitative methods to detect design defects after refactoring" in software engineering research and practice 2008.
[13] M. Monogivi, R. Gheyi, G. Soares, L. Teixeira, and P. Borba, "Making refactoring safer through impact analysis", science of ComputerProgramming, vol. 93, pp. 39-64, 2014.
[14] G. Fraser and A. Arcuri, "EvoSuite: automatic test suite generation for object-oriented software," in proceedings of the 19th European Conference on Foundations of software Engineering, ser. FSE '11 ACM, 2011, pp.416-419.

Authors Profile

MS. B. Nagaveni completed her Bachelor of Technology under the afliated college of Jawaharlal Nehru Technological University Anantapur in 2016. She is pursuing her Master of Technology in computer science and engineering in 2018 from Jawaharlal Nehru Technological University Ananthapuramu, Andhra Pradesh, India.

Prof. A. Ananda Rao received his B.Tech. Degree in computer science and engineering from university of Hyderabad, erstwhile Andhra pradesh, India and M.Tech. Degree in A.I & Robotics from university of Hyderabad, erstwhile Andhra pradesh, India. He received his Ph.D. Degree from Indian Institute of Technology Madras, chennai, India. He is a professor of Computer Science and Engineering Department and currently working as a Director, Research & Development, JNT University Anantapur, Ananthapuramu, Andhra pradesh, India.

Dr. Rao has published more than 160 publications in various national and international Journals/Conferences. He received the best research paper award for the paper titled : An approach to testcase design for cost effective software testing, "IAENG- International conference on Software Engineering, Hong Kong 2009.

P. Radhika Raju received B.Sc. (Comp. Sci) from Sri Krishnadevaraya University, Ananthapuramu; MCA degree from Indira Gandhi National Open University (IGNOU), India and M.Tech degree in Computer Science from Jawaharlal Nehru Technological University Anantapur, Ananthapuramu, A.P, India. She Completed her Ph.D degree from JNT University Anantapur. She authored a text book and has publications in National and International Journals/Conferences. Her research interest include primarily Software Engineering. She is now working as a Lecturer in department of Computer Science and Engineering, JNTUA College of Engineering, Ananthapuramu, A.P.

