ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 2, March 2016

# INSTANT REFACTORING IN SOFTWARE TESTING (IRIS)

**N.Karthigavani**
Assistant Professor
Department of Information Technology
AVS Engineering College, Salem
vaneeamba@gmail.com

**M.Preethi**
Assistant Professor
Department of Information Technology
AVS Engineering College, Salem
preethyit88@gmail.com

**Abstract –**

**Software refactoring is a transformational process that improves the internal behavior of a software program by without affecting the external behavior. It promotes constant improvement as well as greater extendibility through emphasizing the concept of Component Based Development (CBD). The current refactoring methodologies are quite hypothetical. Since there are no strategies followed to evaluate the accuracy of a refactoring implementation. Even, most of the mainstream refactoring engines such as Eclipse JDT, Net Beans etc., possibly will contain critical bugs in their refactored codes. The Instant Refactoring in Software Testing (IRIS) is proposed to instantly monitor and evaluate the appropriateness of refactoring in an automated approach.**

***KEYWORDS-automated-software-testing; software refactoring; instant monitoring.***

## I. Introduction

Refactoring is a transformational process that changes the internal structure of the program without affecting its external behavior. We refactor because we understand getting the design right the first time is hard and you get many benefits from refactoring:

- Code size is often reduced.
- Confusing code is restructured into simpler code.
- Both of these greatly improve maintainability which is required because requirements always change.

The preconditions are used while refactoring. Most of the IDE's such as Eclipse,

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 2, March 2016

Net-Beans, IntelliJ and JBuilder includes refactoring which automates precondition checking and transformation of programs.

The tasks of refactoring processes are mostly non-trivial and also the proof of correctness for the refactoring process is the major challenge of this task. Generally, the refactoring engine developers may uses informal preconditions for implementing the refactoring. As a result, the compilation errors have been detected but the behavioral changes have been left unnoticed. Even though the utilization of the test suites may be considered as trustworthy resources for preventing such issue but the tests may be inappropriate for the outcomes of the refactoring. Also, some of the refactoring tools have been reported as passive as well as human driven. They also have been developed under developer's spontaneity. This may leads to the production of poor software quality and costs huge for the delayed refactorings.

Here, a new technique has been proposed to test the java refactoring engines. It includes two main components: JDOLLY for generation of the java programs

and SAFEREFACTOR for detecting the behavioral changes in those generated java programs.

JDOLLY generates the java programs exhaustively according to the given java declaration scope (packages, classes, fields and methods). JDOLLY utilizes the ALLOY, specification language that constructs the java meta-model. This uses the ALLOY Analyzer that analyses the ALLOY models and generates the solution for the ALLOY models.

These generated ALLOY java models are subjected to the refactoring process. This technique uses SAFEREFACTOR in order to evaluate the correctness of the transformation. It promotes the use of an instant monitor that invokes the smell detection tool while at the instant of refactoring process. It analyses the changes brought by the SAFEREFACTOR and reports them through SMELL VIEW. The SMELL-VIEW delivers the information regarding the type of smell that has been introduced. By rectifying the smell at that instant of detection the total defects have been reduced by a huge amount. Also the lifespan of smells have been reduced drastically. The process of resolving the code smells have also been improved.

## II. Technique

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 2, March 2016

The basic refactoring process consists of a number of distinct activities:

1. Identifying where should be the refactor must be done in the software;

2. For the identified places, determine which refactoring(s) should be applied;

3. Ensure behavior is preserved by the applied refactoring;

4. Apply the refactoring;

5. Analyze the quality characteristics of the software for the applied refactoring;

6. Stabilize the consistency between the refactored program code and other software artifacts.

There are 4 main steps have been adopted in this process. (Step-1) An automated generator that generates programs as a yield of test inputs using JDOLLY. (Step-2) These programs have been subjected to the refactoring process in which the transformation process has been adopted using SAFEREFACTOR. (Step-3) These transformations are analyzed instantly using INSREFACTOR and detected for any behavioral modifications. These modifications are corrected and evaluated against the existing refactored codes in order to verify the preservation of the correctness of the code.

The JDOLLY generates the java program according to the ALLOY specifications prescribed by the developers. These java programs are served as the test inputs for the process of the refactoring. The translation of the test inputs (Java programs) were done by Alloy Analyzer. The Alloy specifications are the sequence of paragraphs containing *Signatures* (set of objects and relations along with the type) and *Constraints* (value of the objects). The Classes and associations are termed to be as the *Signatures and Constraints.*

The SAFEREFACTOR proposes the refactoring strategy for the test inputs. It utilizes the *Pre-Conditions* to evaluate the correctness of the refactorings. These *Pre-Conditions* guarantees the behavioral preservation of the refactored code. In previous methodologies the test cases only compares the program for any behavioral changes. In case of SAFEREFACTOR, it generates the tests that can compare the program's behavior. In case of varying results, the behavioral change has been reported instantly by the tool and displayed as an unsuccessful test.

The INSREFACTOR is made up of a monitor, set of smell detectors associated along with the refactoring tools, and a smell view. Normally, the monitor monitors for the changes made in the source code. Typically, the monitor has been set to be run in the background of the IDE in order to avoid the blocking of main threads. The smell detectors are invoked by the monitor once if the change has been introduced. It analyzes the change and reports to the smell view.

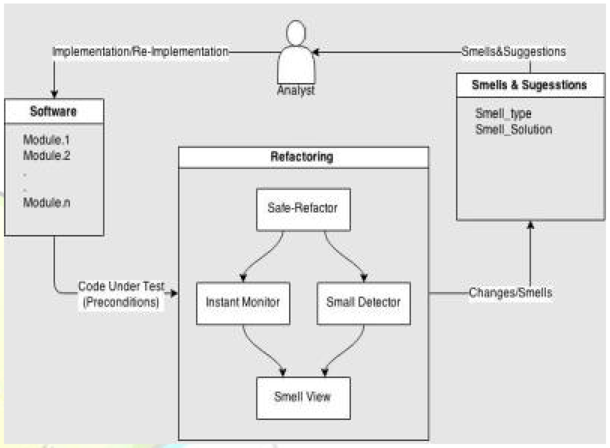
The smell view reports the change's type, explanation and suggestions that would be more helpful for the developer's to rectify the smells that have been introduced. The result of the evaluation will be made based on the amount of smells that has been detected, acknowledged and rectified. Besides, the quality of the source codes has been gradually increased by rectifying the bad smells. A threshold value has been maintained for the performance analysis.

$$Precision\,rate\ P = S_D \div (S_A + S_R)$$

$S_D$ - Number of smells detected,

$S_A$ - Number of smells Acknowledged,

$S_R$ - Number of smells Resolved.



## III. Framework

In general, many of the software refactoring works on the basis of the smell detection algorithms like feedback control algorithm for optimization of smell detection threshold. Sometimes a feedback controller is also set to work parallel with refactoring engines. This may also be utilized to optimize the process of detection of the bad smells that are introduced into the program source codes.

It is an approach that automates testing of refactoring engines. The core of this approach is a framework for automated generation of input programs. It allows developers to write imperative generators whose executions produce

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 2, March 2016

input programs. More precisely, it offers a library of generic, reusable, and compassable generators that produce abstract syntax trees. With this system, a developer can generate more tests and focus more on the creative aspects of testing than when with manual generation. Instead of manually writing input programs, a developer writes a generator whose execution produces thousands of programs with structural properties that are relevant for the specific refactoring being tested.

Fig.1 Instant Refactoring System- Working Framework

For example, to test the Rename Field refactoring, the generated program should contain a class that declares a field. This generator systematically produces a large number of programs that satisfy such constraints. It follows the bounded-exhaustive approach for exhaustively testing all inputs within the given bound. This approach covers all "corner cases" within the given bound. In contrast, manual testing requires knowledge of the corner cases and manually-written tests covering each. Bounded-exhaustive testing has never been used for test inputs as complex as Java programs, and the approach of imperative generators introduced differs by using declarative generators.

The developers can be enabled to implement the source codes via Eclipse IDE. Once the program has been implemented, the monitor which runs on the background monitors the implementation of the source code. The behavioral changes of the source codes have been monitored instantly while under the refactoring process undergoes. The Refactoring tools have been utilized for the process of refactoring.

The SAFEREFACTOR uses pre-condition during the process of refactoring in order to maintain the behavioral consistency. Once the changes have been introduced, the Monitor invokes the SMELL-DETECTOR to analyze the changes. The changes have been analyzed and reported instantly. The bug report includes information such as type, explanation and suggestions.

The bug information will be reported through SMELL-VIEW. Using the SMELL-VIEW detail, the bugs have been rectified and reported immediately to the developer. The suggestions made by the SMELL-VIEW will be useful in rectifying the changes that have been introduced into the code. This process continues in order to obtain the better quality code with a

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 2, March 2016

minimal amount of changes/Bugs.

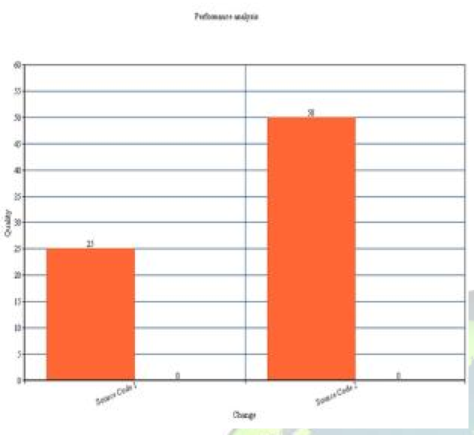


Fig.2 Performance analysis

The performance graph shows that the process of instant refactoring system has been proved at a greater extent of quality and performance than the existing normal refactoring system.

## V. Conclusion

The Automated testing for the refactored codes proves that the quality of the program codes have been increased mush more than using the normal refactoring and testing system. Due to the instant monitoring and analysis, the source code changes have been rectified instantly by the developer to improve the production of the quality program codes. It has been analyzed that the quality of the source code after refactoring has been gradually increased over the instant modifications done by the developer.

Future work will be carried over on other kinds of IDE's such as Net-Beans, IntelliJ etc. Also the semi-automated change modification will also be fully automated under the system that facilitates the developer for clear refactoring process. This may be useful for the people who do not much aware about the refactoring process.

## References

[1] Hui Liu, Xue Guo and Wizhong Shao, "Monitor-Based Instant Software Refactoring" , IEEE Transactions on software Engineering, Vol.39,No.8, August 2013.

[2] Gustavo Soares, Student Member,IEEE Rohit Gheyi, and Tiago Massoni, "Automated Behavioral Testing Of Refactoring Engines" ,IEEE Transactions on software Engineering, Vol.39,No.2, PP. 147-162, February 2013.

[3] T. Mens and T. Tourwe´, "A Survey of Software Refactoring," IEEE Trans. Software Eng., vol. 30, no. 2, pp. 126-139, Feb. 2004.

[4] Marico Cornelio, Ana cavalcanti, Augusto sampaio, "Refactoring by Transformation", Electronic Notes in Theoretical Computer Science 70 No. 3 (2002).

[5] Miryung kim,University of Texas,Austin, Thomas Zimmermann, Nachiappan, Nagappan"Appendix to A Field Study of Refactoring Rationale,Benefits, and Challenges at Microsoft", Microsoft Research

Technical Report. MSR-TR-2012-4.

[6] M.S.Amalan, M.S.Geethadevasena, "Code Detection and Evaluation Using Search Based Refactoring", International Journal of Computer Trends and Technology (IJCTT) – vol.4 Issue.4 – April 2013.

[7] K.V. Hanford, "Automatic Generation of Test Cases," IBM Systems J., vol. 9, pp. 242-257, Dec. 1970.

[8] Eclipse.org, "JDT Core Component," http://www.eclipse.org/jdt/core/ , 2011.

[9] W. Jin, A. Orso, and T. Xie, "Automated Behavioral Regression Testing," Proc. 23rd Int'l Conf. Software Testing, Verification and Validation, pp. 137-146, 2010.

[10] P. Borba and S. Soares, "Refactoring and code generation tools for AspectJ," in OOPSLA 2002 Workshop on Tools for Aspect-Oriented Software Development, November 2002, Lecture Notes in Computer Science.

[11] D. Dig and R. Johnson, "The Role of Refactorings in API Evolution," Proc. 21st IEEE Int'l Conf. Software Maintenance, pp. 389-398, 2005.

[12] Max Schafer, Oxford University, Andreas theis & Friedrich Steinmann, Fern university, Hagen, Frank tip, IBM Research Division, "A Comprehensive Approach To Naming and Accessibility in Refactoring Java Programs", IBM-RESEARCH-REPORT, C25201(W1108-027), August 8,2011, Computer Science.

[13] Mohan kumar, GPM & PRD Group of Infosys, Rajeev kumar agarwal, PRD Group of Infosys, "View-point: Refactoring", 2012, Infosys Limited, Bangalore, India. IleneBurnstein, "Practical Software testing", Springer professional computing, Springer - Verlag New York, Inc.Sec.15.5, PP.518, 2003.

[14] IleneBurnstein, "Practical Software testing", Springer professional computing, Springer - Verlag New York, Inc.Sec.15.5, PP.518, 2003.