iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** IV **Month of publication:** April 2021

DOI: https://doi.org/10.22214/ijraset.2021.33595

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue IV Apr 2021- Available at www.ijraset.com

Implementation of Randoop Automated Test Generation Tool for Novel Perspective on Software Development

Mohit Bohra
Computer Engineering Department, SVKMs NMIMS University, Shirpur, India

Abstract: ***Code testing constitutes a central method in code development to ensure the consistency and reliability of source code. Numerous generic software tools for producing test units are already proposed to support developers through this process and increase the test time.***
Most of these approaches aim to cover as many branches as possible. The aim of automated generation of test is to increase the efficiency of the product testing and to reduce the human costs involved. The automation test generation could be implemented through the test adapters at all test rates.
It has contributed to improving reporting as well as discovering new vulnerabilities which previous research activities have not identified in the projects. Automated method parameter generation is required in the benchmarking of test case random generation scenarios.
Although test adapters can also be performed effectively at the test case stage, they're functionality as well as the related efforts are growing to provide significantly higher test levels for adapter implementations. A full automation support is strongly recommended for software testing to effectively address the demands of a limited-resource testing environment. In this study we analyse the extent to which existing Randoop automated test case generation tool effectively produces test code. Generation of random tests by integrating feedback obtained from the execution of test inputs as generated.
Our methodology incrementally constructs inputs by selecting random process call to add and extract assertions from subsequently developed inputs.
Keywords: Automated Unit Test Generation, Java Unit test generator, Randoop Tool

I. INTRODUCTION

A software product is classified into two types of interests: core factor that implies to the leading function required by the system, and cross-cutting factor that implies to the functions which are frequent in many core components of the system (i.e., modules implementing core factors).

Ensuring the appropriate behaviour of any successful software product, software testing is extensively accepted as a significant method. Though object-oriented programming (OOP) offers the foundation of a system carving methodology for core factors of interests by using classes and objects, the crosscutting factor implementations are dispersed across considerable number of core modules[8].

The basic elements containing methods and fields for these Java applications are classes. The functional characteristics (example accuracy) and extra-functional characteristics (example efficiency) of the methods are subject to industrial requirements and the quality of code must be guaranteed in today's data driven environment, and the impact of method input parameters must be considered in the analysis of those properties[1].

The primary intention is to enhance code coverage and trim 1.the time of execution 2.cost 3.dynamic memory usage 4.Number of mistakenly missed goals[4].

Unit testing has been one of the leading software testing methods with the main goal of evaluating the software's individual programming element for example, several method in the source code, a list of certain of methods of the system, or classes along with suitable control over data and operating steps. Unit testing reduces the cause of bugs at the initial phase and improves designing and integration of the methods which enables high quality of software product[5],[9].

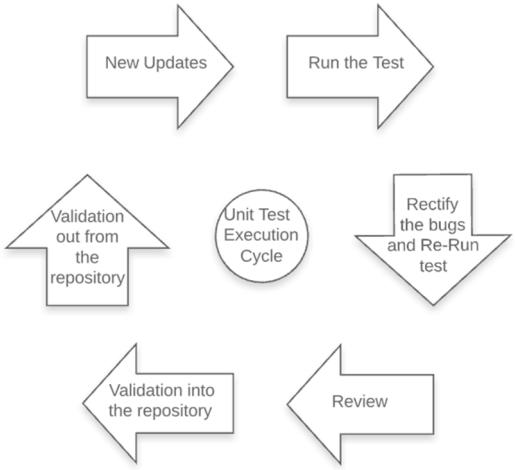


Fig. 1. Unit Test Execution Cycle[13]

Automated Test generation has been suggested as a strategy for reducing the great effort associated in manually adding test cases and speeding up test cycles. A simple test-generation tool needs to achieve and to find an input that will move execution of a (accepted, loop-free) source code class or method along all plausible path in its control flow chart [1].

Randoop is an Automated test generation tool which uses feedback-driven random testing to construct test cases by random order of method calls to the framework being tested. Randoop includes multiple ways of adapting its nature to a particular system. Randoop is totally self starting tool, needs no user feedback (other than a binary name for .NET or a Java class directory), and scales practical applications into hundreds of classes[3]. The collection of built-in relationships can be expanded with the framework which needs to be checked with unique properties checks, their occurrence may lead to error-revealing tests for different types of exceptions and additional values may be included in the test data list[6]. Randoop implies that random feedback generation retains the advantages of random testing (scalability, ease of implementation), eliminates the drawbacks of random testing (generation of inefficient or inconsequential input data), and is able to compete with systematic techniques[6].

There is still a shortage of factual analysis for the quality of the test code generated by such instruments[4]. The aim of such tests is to address complicated faults, however there are no autonomous oracles and which have to be inspected manually after the creation process. To achieve effective coverage, errors must be reduced to the greatest extent possible. The purpose of these tests is to identify complicated faults for which no autonomous oracles are readily accessible and need to be verified manually after the production system. Moreover, recent studies have already shown that badly written test scripts (i) causes quite-called flakiness, i.e. the non - trivial actions of a test (ii) restricts the ability of test cases to detect faults[4].

II. RANDOOP

An object-oriented unit test comprises of a sequence of state-setting method calls (such as object creation and mutation), and declaration of end call tests. This segment presents a methodology for randomised, feedback-driven unit testing[3].

A deterministic feedback-driven execution tool called Randoop, identifying and removing test inputs that either create exceptions or similar artefacts. These sequences are generated progressively by generating random a calling method and acceptable assertions, maybe from a collection of previously defined testing data or even from data generated by subsequently implemented sequences over the call method. Every new sequence is implemented at formation. When the sequence produces an undefined exception, or violates the designed arguments, error-revealing will be the test case, which will be a failed JUnit test case. Nonetheless, when sequence shows standard behaviour, additional random method calls will extend it further[6]. Randoop includes the intervention of the test engineer to set the time-limits for the Reduced Series. Randoop is capable of exploring space more, the reason is not because it explore a greater area of state space but it only explore a small part of a huge state space[5].

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue IV Apr 2021- Available at www.ijraset.com

III.HOW IT WORKS

Randoop initialise with no data values available to it for generating parameters i.e, it starts from scratch. It creates input by calling methods from the class and incrementally applying it to further randomly selected methods which leads to create new sequence which is than applied to the software. The flow of the sequence moves in the following direction. An initial random sequence start with an empty set of sequences which moves on incrementally. A method m1 which is selected on a random basis belonging to output class. Randoop seems to be applying an extension operator to m1: this generates a new sequence s by combining a collection of subsequent input sequences accompanied by the method call m1. All this sequences will the executed to justify that it is non-redundant and accurate. The set of sequences which are valid and break no exception are then randomly selected and added to sequence S. The machine flow should add all of the specific s to the sequence S. Multiple calls to m1 are required in some specific cases to attain the required object state. Randoop induces mechanism to iterate for this particular purpose and for this reason it adds M calls with a default highest value equal to 100 instead of adding a single call m1 to create a new sequence[4],[3]. When the search get over Randoop generates the regression test from S, this happens when the stated or default time period is over. On the other hand the non valid or exception generating sequences will be generated in the error-revealing tests[3],[4]. At the end, Randoop places assertions over each statement of the valid sequence S by iterating while focussing on the return data value of each method that returns a datatype and puts the correlating assertion. In some instances the statement is not implemented for strings that represent unprocessed object references[4].

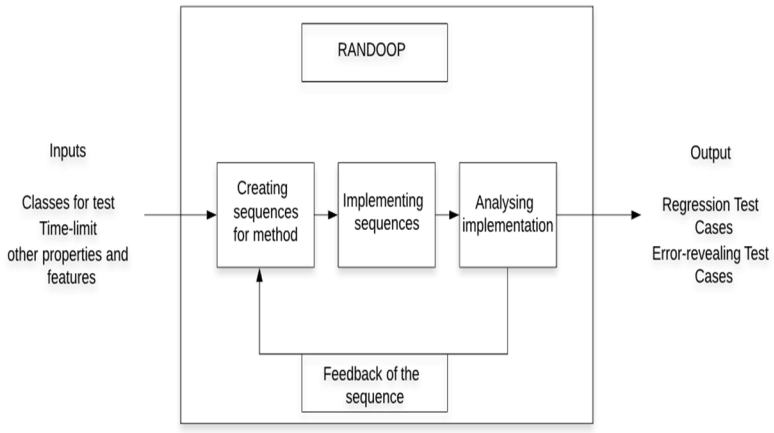


Fig. 2. Functional Architecture of the RANDOOP [12]

Test suite reduction or minimization methods were supported by Randoop, randomised device testing. Randoop has an Object Oriented model based coverage approach with a single purpose optimisation. It covers the source code in offline mode on a single server and has no license[5]. The single-objective optimisation module relies on expenditure such as failure in the ability to detect faults or efficiency with regard to the size of the reduced suite. Randoop produces test methods, for every new sequence it explores, it generates new test adapter instances. The creation of a new adapter instance often introduces a test-level Graphical User Interface Control Instance. However, this is a big issue in integration and system testing due to the System under test startup time[6].

In the subject programs, test cases have been created only for classes. Randoop helps the developer define the classes to be assessed, and even the methods to name the test cases[8]. One can refrain from calling the test cases directly from other methods and advice using such feature.

In the public version of Randoop, the tool is used to create assertions which verify the uniformity of the inputs for missing value, reflexivity and equality. Interestingly, observer methods can also be said to produce stronger, application-specific assertions by using observer alternative choice[8].

While completing the execution the sequences are then converted into several regression test cases, which are already been passed JUnit tests cases by inserting statements representing the exercised code's recorded behaviour. When a specified time constraint or quantity of sequences are tested, the test generation ends[6].

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue IV Apr 2021- Available at www.ijraset.com

IV. APPLICATION

In order to create test cases using this automation tool, it is essential to study more about the manual provided. While working on generating test cases, we need the (.jar) extension file that covers all of the Randoop tool's new feature, in the initial phase, additional loads of error were faced while generating test cases of a small java class. This may happen due to the need to execute a large string command on the command prompt or terminal window. As per the observation we have seen, Randoop requires the directory path of the class file and the complete directory path of the.jar file of Randoop. After all the major paths we need to call Randoop's main function, which is the reason for the execution and generation of test cases, the source code needs to be called using randoop.main. Main followed by a feature that needs to be done. Generating test cases (i.e. gentests) and minimizing the test suite (i.e. minimizing) depend on the tester.

Gentests require the name of the test class and a number of other functions that are set to default if the command is not specified or changed.

The most fascinating method of the gentests is to give a class name in a list format that can be highly efficient and time-saving for the tester. While enlisting all the class name in the text file and passing the text file path, the unit test cases will be generated for each and every java class enrolled in the text file[10].

Reducing or minimizing the test suite requires the path of the suite directory and the path of the suite class. Having some minimization function improves the tool's reduction technique[10].

V. CONTRIBUTION

We have contributed a small User Interface to Randoop's Quick and Easy Testing tool. Introduce all the functions of Randoop to the interface by selecting from the drop-down menu, the buttons, the checkbox and the text field. The interface's interesting feature is to save all previously accessed variables and commands that are stored in a list format in a text file near the same RandoopUI directory[11].

Starting the interface will show a login page that will make you choose the values in the text fields that will allow you to run the command operation at a minimum of time. You can click the Help button to learn more about the Randoop Manual tool[10]. Whenever you select and create a new configuration option, it will be stored and this configuration which is stored can be access later[11].

Clicking on the major functions of gentests or minimizing you will be moved to the respective frame containing all the functions that initially have all the default values, and if any of the values are changed, they will be visible in the command. All fields required, such as classpath, jar file path, and the class that needs to be tested, can not be left empty. As soon as the Run or Next button is clicked, you save the configuration and you can access it whenever you are back. In the meantime, in gentests, we have given the JUnit directory path mandatory so that the generated test will be stored at the known location of the tester[11].

Finally, when you run the command, the execution window will show the exceptions and executed command steps and the sequence generated steps. The regression test generated and the error revealing test generated with the test suite will be clearly shown[11].

VI. LIMITS

Randoop fails to create a parameter graph instead of using previously created test sequences of a component set that are useful for creating similar objects or throwing exceptions[5]. This could be because the default Randoop configuration provides a fixed amount of test cases as output test suites. Consequently, amongst our current study, the amount of the tests generated does not vary that much.

There is a shortage of funding from Randoop for multi-threaded programmes[4]. We found that Randoop is not affected by the quality of the generated code: this implies that while these tools are being implemented, the refactoring of the existing code does not allow the production of better test cases. New solutions should be developed for such tools which enable a performance-aware random generation of test cases[4].

Randoop focused on feedback-driven system that allowed limited object access. For example, when Randoop can not identify an object of an appropriate version needed to override a method in the current collection of sequences, it will never invoke such method[5]. Furthermore, existing randomised approaches focus mostly on test execution-oriented cases , i.e. attempting to find cases for which software behavior differs from the targets assumed[1].

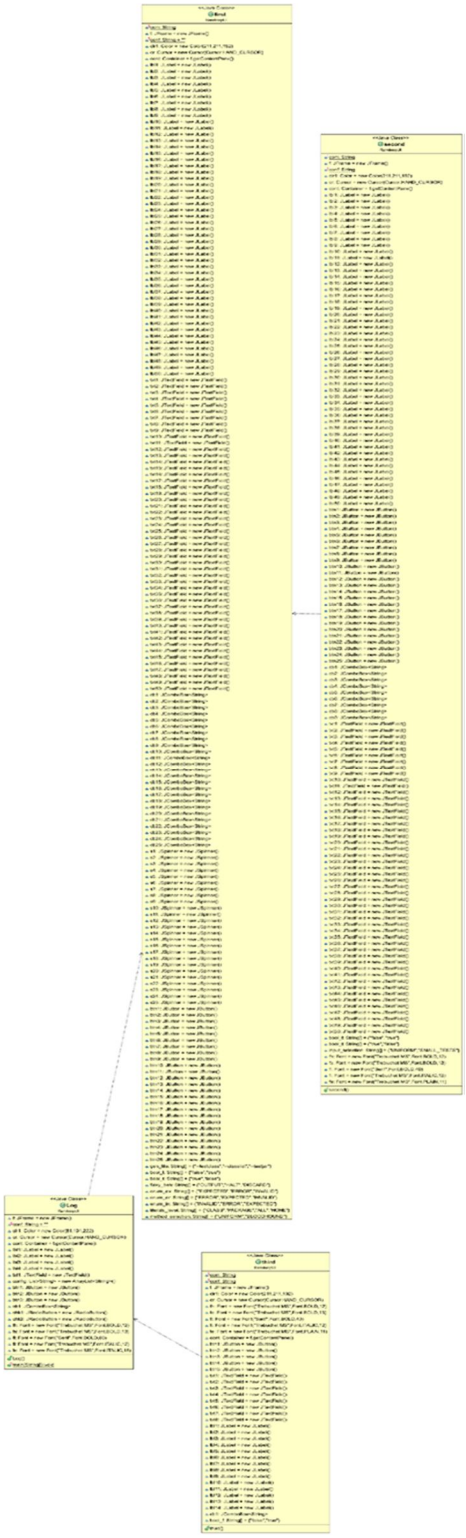


Fig. 3. Class Diagram of the Randoop User Interface [14]

VII. CONCLUSIONS

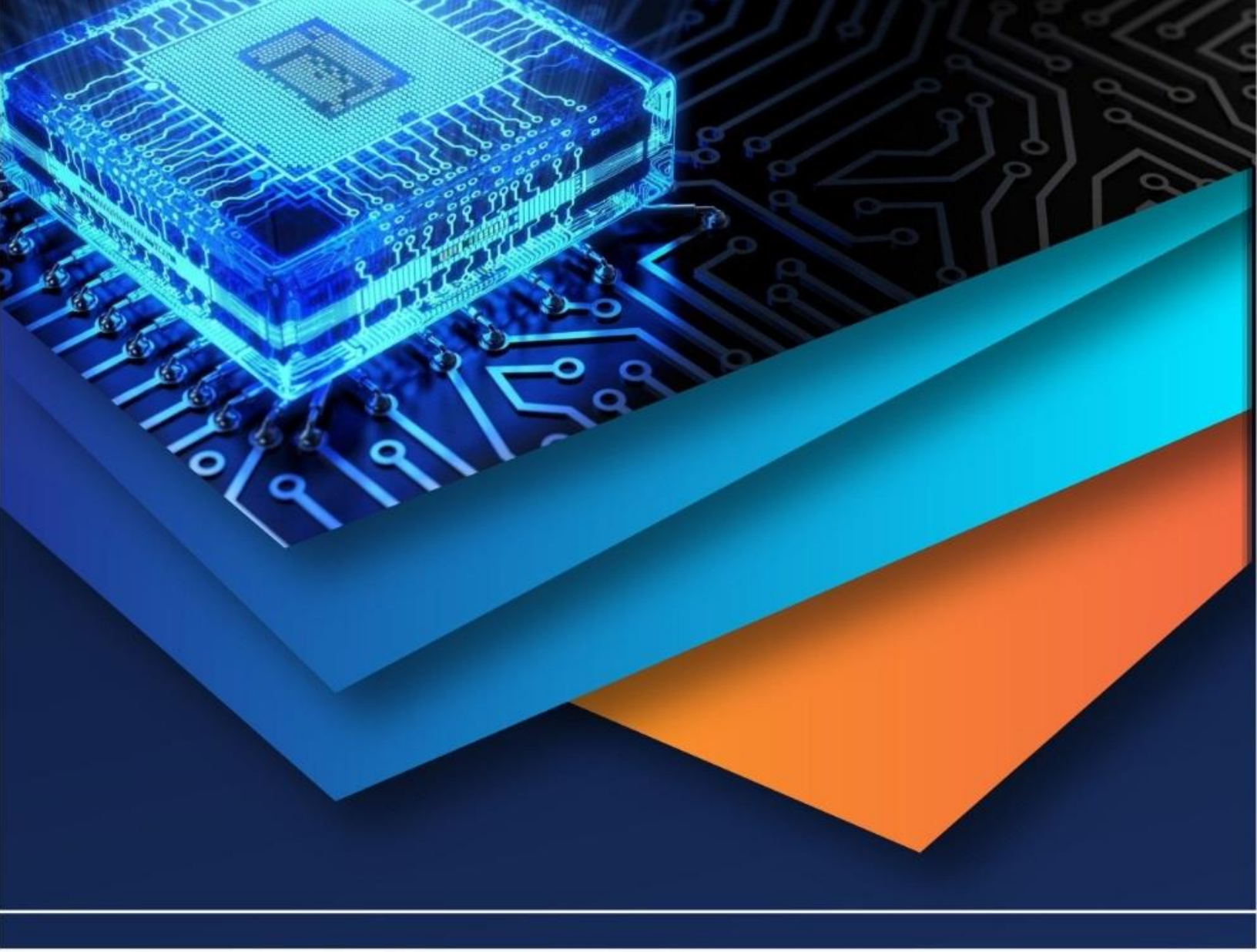
Randoop operates on randomly generated inputs and uses a feedback-directed approach and regression testing approach to prevent test generation exceptions in the code. Quality of the test code may affect the effectiveness of the test cases. For interface testing, Randoop allowed the development of a broad range of interaction sequences, which improved coverage and detected new faults.
In proposed development we intend to pursue the concept of test automation to additional sectors and development tools in order to gain more information on the benefits and disadvantages involved. We would like to investigate whether test modules can be used to make automation test generation suitable to advanced technology and scripting languages in which there are currently no appropriate Automation test generation tools available. Feedback-driven random test ranges to complex systems, quickly finding errors in intensively tested, widely deployed platforms and achieving behavioural exposure equivalent to quantitative techniques.

VIII. ACKNOWLEDGMENT

I would like to acknowledge all my faculties and mentors whose guidance helped me a lot and all my faculties and staff members of NMIMS Shirpur.

REFERENCES

[1] Michael Kuperberg and Fouad Omri “Using Heuristics to Automate Parameter Generation for Benchmarking of Java Methods”
[2] Marcel Böhme, Abhik Roychoudhury, and Bruno C.d.S. Oliveira “Regression Testing of Evolving Programs ”
[3] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball MIT CSAIL, Microsoft Research “Feedback-directed Random Test Generation ”
[4] Giovanni Grano , Fabio Palomba , Dario Di Nucci , Andrea De Lucia , Harald C. Gall “Scented since the beginning: On the diffuseness of test smells in automatically generated test code”
[5] Saif Ur Rehman Khan, Sai Peck Lee, Raja Wasim Ahmad, Adnan Akhunzada, Victor Chang “A survey on Test Suite Reduction frameworks and tools”
[6] Rudolf Ramler , Georg Buchgeher, Claus Klammer “Adapting automated test generation to GUI testing of industry applications”
[7] Carlos Pacheco Michael D. Ernst “Randoop: Feedback-Directed Random Testing for Java”
[8] Fadi Wedyan , Sudipto Ghosh, Leo R. Vijayasarathy “An approach and tool for measurement of state variable based data-flow test coverage for aspect-oriented programs”
[9] https://www.tutorialspoint.com/software_testing_dictionary/unit_testing.htm
[10] https://randoop.github.io/randoop/manual/
[11] https://github.com/mohit67bohra/Randoop-User-Interface.git
[12] Figure 2. RANDOOP’s input is a set of classes, a time limit, and optionally, a set of contract checkers. It outputs contract- violating tests and regression tests. Adapted from Carlos Pacheco Michael D. Ernst “Randoop: Feedback-Directed Random Testing for Java”
[13] Figure 1. Unit Testing Lifecycle. Adapted from https://www.tutorialspoint.com/software_testing_dictionary/unit_testing.htm
[14] Figure 3. Class Diagram of the Randoop User Interface. Created from proposed Architecture https://github.com/mohit67bohra/Randoop-User-Interface.git



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY