Automated Test Data Generation for Coupling Based Integration Testing

Amit A. Kadam
PG Student of Department of Information Technology
Pune Institute of Computer Technology
Pune, India.
kadamamit1811@gmail.com

Anant M. Bagade
Associate Professor of Department of Information Technology
Pune Institute of Computer Technology
Pune, India.
ambagade@pict.edu

Abstract— **In the area of software testing, researcher has a challenging problem for test data generation which is automated. Up till now, most of the work on generation of test data is at unit level. In unit level testing generation of test data contains the execution of test route at unit level where collaboration with other component is minimum. The problem of generation of test data automated becomes very difficult when we suffer from other levels of testing including integration testing or system level testing. The base of integration testing level, the variables are passed as an arguments to other module and variables change their names; also number of different paths are executed from different module to ensure the proper functionality. Recently evolutionary approaches have been verified a powerful tools for test data generation. Integration testing can be tests the interactions of different modules, when they are integrated organized in precise application, for the smooth functionality of the software system. The Coupling based testing is an integration testing technique that is based upon coupling interactions that occur between different variables through different call sites in functions. Until now no work has been done on generation of test data for coupling based integration testing using genetic algorithm. So, in this paper, we will propose architecture of tool Quick_Gen for generation of test data automated for coupling based integration testing of OO programs.**

Keywords- Software testing; Test Data Generation; Automation; Integration Testing; Coupling path; Genetic algorithm (GA).

I. INTRODUCTION

Software testing is a very essential part of software development practice. Once the source code has been produced, program must be tested to find out (and correct) as any faults as possible. Program testing is very costly, labor-intensive and boring, consuming at least 50% of the total development expenses but still the results are not constantly satisfactory. However, to confirm that the design and implementation of the programs match with the given requirements, software testing is always necessary job. Automation of the testing process can minimize the problems that occur in future and/or problem involved. Software testing is the process of assessing a system or its component(s) with the intending to find that whether it satisfies the specified requirements or not. Testing is a process of representing that there are compact critical errors in the software. Software testing can always display the presence of errors and not the lack of errors [2].

With the advancement in software technology, as number of software products are increasing, maintenance is becoming a challenging task. Maintenance activities account for over two third of the life cycle cost of a software system. Hence a lot of time and efforts are required for maintenance phase of software development lifecycle. Essential activities involved in SDLC are software testing, and without software test data it not possible. Many developers put significant amount of effort for generating test data and perform different test cases for fixing software bugs.

Software testing can be performed at different levels such as unit testing, integration, or system level testing. Unit level testing confirms the functionality of individual units. Integration level testing, tests the interaction of different modules after integration with other modules. System level testing gives the system as black box and checks the functionality of the system as a whole. Integration testing is an important level of testing which verifies the different modules interactions and message passing through interfaces. Unit level testing is a base for integration testing, if the units work correctly then different units are integrated together using different interfaces exposed by different modules. Integration level testing verifies that the interfaces are correctly integrated and message passing through interfaces is correct. Integration testing is concerned with the interactions among components [1].

Does a module call other modules correctly? Are the right parameters with right types and ranges are passed? Does the called method return the proper type and the value is in the correct range? These questions are focus of the integration testing. Unfortunately, very little research has been done in the area of integration testing for coupling based test data generation using evolutionary approaches [1].

The Coupling based integration testing is based on coupling relationships that existing among variables across call sites in actions. In the same way as unit level testing is base for integration testing, integration testing is the base for system level testing. So system level testing is very difficult to achieve before the integration testing.

Test data are an important part of test case, without test data test case execution is not possible. Research has explored several methods for test data generation using evolutionary approaches [9, 10]. A number of test data generation approaches have been developed and automated. Random test data generation, generates test data based on selective random inputs form some

distribution. Path- oriented and structural approaches use the program's control flow graph for test data generation; they select a path, and use a technique such as symbolic execution for generation of test data. Goal-oriented test-data generation approaches select inputs to execute the selected goal, such as statement, condition coverage, decision coverage, irrespective of the path taken. Evolutionary test approaches use evolutionary algorithms i.e. genetic algorithm, for selection and generation of test data by applying evolutionary operators, i.e., crossover and mutation. Maximum of the work on test data generation has been done at unit level testing. Unit level test data generation involves the test data that executes the test case for unit level testing [9, 10].

This paper presents a new approach for generation of test data automated for coupling based integration testing of OO programs based on genetic algorithm. Our method takes the test path as input, having different sub test paths, and test data is generated using genetic algorithm. We are proposing a tool Quick_Gen in Java for generation of test data.

The rest of the paper is organized as follows: Section II explains the background information of evolutionary approaches and coupling based integration testing. Section III represents the literature survey. Section IV describes the proposed method of test data generation for integration testing using genetic algorithm and Section V concludes the paper.

II. BACKGROUND

A. Evolutionary Algorithms

Among the traditional search and optimization methods, the development of Evolutionary Algorithms (EA) has been very important in the last decade. It success is depend on solving difficult problems has been the machine of a field called as Evolutionary Computation (EC).

Advantages using EC techniques mostly come from flexibility improvements and their fitness to the objective goal in combination with a healthy behavior. Now days, EC is study as an adaptable concept for problems solution, especially complex optimization problems.

Evolutionary algorithms [11] have been useful to software testing for generation of test cases and test data automated. The application of evolutionary algorithms to testing for generation of test data is known as evolutionary testing. Evolutionary algorithms include different types of methods such as genetic algorithms, particle swarm optimization, ant colony model etc. Evolutionary algorithms usage replicated development as a search strategy to evolve candidate solutions, utilizing operators inspired by genetics and natural selection. In an evolutionary computation, a population of candidate solutions is randomly generated. Each candidate solution is then given a fitness value based on a user specified evaluation function. A number of individuals are then selected as parents based on their fitness. The selected parents are then allowed to generate a set of offspring using evolutionary operators cross-over and mutation which are assessed and assigned a fitness using the same evaluation function defined by the user. This procedure of selecting parents based on their fitness, allowing them to generate offspring, and replacing weaker members of the population is recurring for a user specified number of iterations.

B. Coupling Based Integration Testing

Jin and Offutt [12] proposed an approach for integration testing of procedural languages that is built upon coupling relationships among variables through different call sites in different actions. They distinct three types of coupling relationship that must be tested: parameter coupling, shared data coupling and external device coupling. But in this paper we generally used parameter coupling and shared coupling. When one procedure passes parameters to other procedure is called parameter couplings. When two procedures references the same global variables is called shared data couplings. External device couplings arise when two procedures accesses the same external storage standard.

These coupling types are defined as [6]:

- **Call coupling**: component A calls another component B without passing parameters, A and B do not share any common variable references, or common references to external media.
- **Parameter coupling**: A calls B and passes one or more data items as a parameter.
- **Shared data coupling**: A calls B and they both refer to the same data object (either globally or non-locally).
- **External device coupling**: A calls B and they both access the same external medium (for example, a file or sensor).

Up until now, test data generation approaches cater only unit level testing. Up until now no work for generation of test data for coupling based integration testing. In this paper, we proposed a new approach for generation of test data automated for coupling based integration testing of OO programs using genetic algorithm. In this approach takes the coupling sequence/path as input, covering different sub paths, and generates the test data using genetic algorithm. Unit testing Involve test data generation for single path, execution and monitoring of single path is required in unit testing. In integration testing, as different modules interact with each other so execution and monitoring of multiple paths are required. In unit testing test data generation involves a single path and there is not at all usage of formal parameters. In integration testing, different procedures interact with each other via passing message passing through actual and formal parameters so there is a requirement of mapping between actual and formal parameters. We have to maintain a mapping table for that contains mapping between actual and formal parameters as variable change names in formal parameters. Def use analysis is very difficult to achieve as variables change name in formal parameters so a mapping must be maintain for actual to formal parameters. Our proposed approach takes coupling path as input and generates the test data for

given coupling path. A coupling path consist of sub paths, one path is for antecedent method that defines the coupling variable and other path is for consequent method that uses the coupling variable define in antecedent path [1].

III. LITERATURE REVIEW

Before many of the literatures are available on test data generation using evolutionary approaches there are different types of evolutionary approaches are available in the corporate world such as genetic algorithms, particle swarm optimization, ant colony model etc. So that automated test data generation is the basic need for software testing.

Ankur & Gursaran [3] perform test data generation for unit testing. In this approach the focus is on test data generation for unit testing of Object-oriented Java programs using Genetic Algorithms and program specification written in JML.

In [4] this approach, authors given a new method for generating test data based on program slicing and particle swarm optimization. With the concern facts selected from a target path, they performed a program slicing to eliminate the statements which are not relevant to the interest points. In this approach method shortens the target path and the actual path to get a better fitness value. After program slices got, the population is developed using particle swarm optimization to improve the efficiency of generation test data.

Cheon, Kim & Perumandla [5] approach unit testing of object-oriented program is completely automated. The author proposes that a programmer should be able to perform unit testing by a single click of key or a single command execution. A complete automation of program testing contains automating the three components of testing: test data selection, test oracle, and test execution. The essence of their approach was to combine JML and genetic algorithms. The approach uses conditions as test oracles, and JML is used to write such conditions. So, that in this approach generate the test data for unit test programs only.

P. McMinn [6] provides a broad survey on evolutionary testing approaches and deliberates the application of evolutionary testing. Baresel et al. [7] recommends some modifications in fitness function strategy to improve evolutionary structural testing.

Fraser and Zeller [8] approach measure the quality of test suites, mutation analysis gives artificial defects (mutations) into series; a non-detected mutation shows a weakness in the test suite. They proposes an automated approach to generate unit tests that detect these mutations for OO programs. This has two benefits: **First**, the resulting test suite is optimized to finding defects modeled by mutation operators rather than covering code. **Second**, the state change caused by mutations encourages oracles that precisely detect the mutants.

IV. PROPOSED METODOLOGY

A. *The Problem & limitations*

Generation of test data for integration testing when two different unit tested modules are integrated and Observe effect of test data after integration. A number of test data generation approaches have been developed and automated. Random test data generation, generates test data based on selective random inputs form some distribution. Path- oriented and structural approaches use the program's control flow graph for test data generation; they select a path, and use a technique such as symbolic execution for generation of test data. Goal-oriented test-data generation approaches select inputs to execute the selected goal, such as statement, condition coverage, decision coverage, irrespective of the path taken.

B. *Approach*

Our proposed approach has divided into two phases, in first phase we identify the coupling path/sequence using Def-use analysis of the code which is based on integration testing criteria i.e.

- ✓ Call coupling
- ✓ All coupling Defs
- ✓ All coupling uses
- ✓ All coupling paths.

In the second phase, we generate the test data for coupling sequence/path generated and random data generated for coupling in first phase of our approach. In figure first phase of approach for automated test sequence /path generation using all coupling information and integration testing criteria. The main important step of our proposed approach are as follows:

- ✓ Call site Identication
- ✓ Coupling Variable Identication
- ✓ Def-use Methods
- ✓ Coupling Sequence/path.

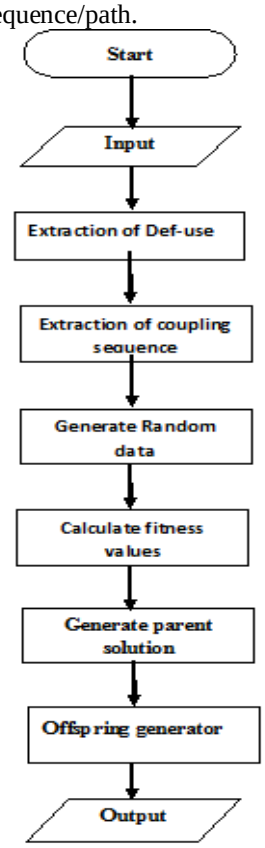

Figure 1. Proposed System for Test Data Generation.

In our approach it consists of different steps-

1) Input:

In our system take source file and mapping table as a input that will help to extract Def-use variable and methods.

2) Extract Def-use:

In this step we are identifying different variable that are define and use. Or Def-use extractions will result in identification Def-use variables, methods and coupling type.

3) Extraction coupling sequence:

In this step we are identify the coupling paths or sequence statically using Def-use analysis of code based on integration testing criteria.

4) Generate Random Data:

In this step we are generating the random data as mapping table given in the input for coupling sequence.

5) Fitness value calculation:

In this step we taking both coupling sequence and randomly generated solutions as an input. As we have to find the optimal solution that act as parent for next generation on the basis of fitness value.

6) Extract Parent solution:

On the basis of fitness value we will generating potential solution are called as parent solution on the basis of their fitness. They are able generate new child by applying two genetic operators such as Crossover and mutations on that potential solutions. Usually these solutions are represented as bit string.

7) Offspring Generator:

There are two main genetic operators – crossover and mutation. In crossover two members of the population are selected and a part of bit string of one is exchanged with the other one. So the resultant individual called offspring contains the best features from both the parents. Then a number of offspring are mutated in order to introduce the diversity in the population. In mutation just a bit is flipped. In this way a new generation is produced. Again the fittest members are chose from the initial and the next generation and the whole process is repeated until we get the desired set of potential solutions[2].

8) Test Data:

At the End the data produces which is our final test data will help tester do execute different test case.

In the first step, the source code is examined for call site identification. A call site is a method call from one method to some other method. Parameters are generally passed through a call site from one method to other method. Coupling variables are identified by analyzing the source code and call sites. Coupling variables are defined in one method and used in other method through a call site. Def-use extractions of coupling variables are performed. Def-use extractions will result in identification of Def-use methods and coupling type. Coupling variables mapping table is also defined in this phase. Coupling sequences/test paths are generated using coupling based testing criteria. Def-use variables and methods are used in coupling paths generation. Up until, no any work done for random data generation for each datatype in java and sequence path are analyze statically. This given to the next phase of test data generation using genetic algorithm and fitness is calculated on the basis on fitness value evaluation function, then it becomes potential solution is called parent solution and generate new offspring's by applying genetic operator CROSSOVER and MUTATION. Finally we get actual test data.

V. CONCLUSION

In this paper, we have proposed a new approach for generation of test data automated for coupling based Integration testing. Our proposed approach consist of two phases, in first phase, source code and mapping table is taken as input and coupling sequences are identified using static analysis of the program and random data is generated for each sequence while in second phase, test data generation for coupling paths and random generated data for those sequence, identified in first phase by using genetic algorithm.

We have proposed a prototype tool called Quick_Gen in which it will generate test data for all data types in java. Quick_Gen gives optimized test data which help developer as well as tester for better result after testing. Now the prototype tool, Quick_Gen, implemented is used for test data generation while paths are provided manually to Quick_Gen.

REFERENCES

[1] Shaukat Ali Khan, Aamer Nadeem, "Automated Test Data Generation for Coupling Based Integration Testing of Object Oriented Programs using Evolutionary Approaches," IEEE 10th International Conference on Information Technology, pp. 369-374, 2013.

[2] Supriya, Chinky Aneja " Test Case Minimization For Object Oriented Technique On The Basis Of Object Oriented Coupling," International society of thesis publication, Volume 2, Issue 4, July 2013.

[3] Hitesh Tahbildar and Bichitra Kalita, " Automated software test data generation: direction of research," International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.1, pp. 99-120, 2011

[4] Shujuan Jiang, Dandan Yi, Xiaolin Ju, Lingsai Wang, Yingqi Liu," An approach for test data generation using program slicing and particle swarm optimization", Springer, 2014.

[5] Cheon, Y., Kim, M. Y., Perumandla, A., "A Complete Automation of Unit Testing for Java Programs," The International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, USA pp. 290–295, 2005.

[6] McMinn P., "Search-Based Software Test Data Generation: A Survey," ACM, Software Testing, Verification and Reliability, vol. 14, no. 2, pp. 105-156, 2004

[7] Baresel, A., Sthamer, H., Schmidt, M., "Fitness Function Design to improve Evolutionary tructural Testing," Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 02), New York, USA, 2002.

[8] Gordon F, Andreas Z, "Mutation-Driven Generation of Unit Tests and Oracles," IEEE Transaction On Software Engineering Vol.38, NO. 2, pp. 278-292, 2012

[9] Lee Copeland, "A Practitioner's Guide to Software Test Design," STQE Publishing, 2004.

[10] Beizer B, "Software Testing Techniques," International Thomson Computer Press, 1990.

[11] Jin Zhenyi and A. Jefferson Offutt. "Coupling-based criteria for integration testing," The Journal of Software Testing, Verification and Reliability, 8(3), pp. 133–154, 1998.

[12] Sthamer, H., "The automatic generation of software test data using genetic algorithms," PhD Thesis, University of Ghamorgan, Pontyprid, Wales, Great Britain, 1996.