Design and Application of an Integration Testing Coverage Tool for Object Oriented Software Using Sandwich Approach

Reetesh Gupta*, Neelabh Sao**

Abstract

Programming testing is an activity which is accustomed to finding bugs or oversights in the product item. There is such a mixture of methods to testing the product item. Subsequently, the picking a best technique will make the product testing capable and feasible. Testing is a discriminating stage in the general structure advancement process. Using testing procedures that bolster highlights of the basic programming perfect model more satisfactorily tests program than do testing methodologies that bolster highlights of distinctive principles. Structures made with the item arranged standard oblige strategies that bolster OO highlights, for instance, legacy, exemplification, information reflection, and element tying. Various strategies that are used to test structures made with the organized model are not sufficient for the testing of article situated systems. The target of this investigation or examination is to make techniques that will upgrade the strategy of testing article situated structures. Specifically, accentuation is given to upgrading the level of testing of systems in light of the way that the level of technique or system testing is generally seen as deficient. Calculations are joined that perceive the plan of methods or routines, both entomb item class and intra article class that should be tried for a given system. These computations are realized as a bit of a robotized testing structure that decides a framework for the testing of systems. This structure consolidates the programmed era of test drivers to support the testing. It catches the outcomes of tests for the reasons of reuse for future structure upkeep. This framework gives the item or programming specialist who is attempting a structure a segment to center the level of scope system that has been finished in the testing strategy. Nowadays OO programming is more standard or straightforward. In this paper we proposed a supporting instrument apparatus for OO programming. This device can survey the entomb class testing scope of OO programming utilizing sandwich approach. Likewise, if a bit of the revealed classes or technique has not been tried by existing experiments then it can make additional or extra experiment. So here I am demonstrating structure of proposed device their design, utilization and detail moreover measures the item quality with respect to its ability for acknowledging to exactness, constancy, usability, common sense, reusability and testability.

Keywords: Code Coverage, Integration Testing, Object- Oriented (Oo) Software, Test Coverage Analysis, Coverage Measurement, Software Testing, Test Coverage, Software Quality

INTRODUCTION

A definite operation is an imperative trademark for programming, so programming testing is a basic strategy of programming headway to affirm the operation. One of concern issues of testing is test scope or scope in light of the fact that deserts may exist in uncovered part and show up when customers endeavor to use them. Nowadays, the OO programming gets the chance to be furthermore appealing in programming industry which the test strategy contrasts from the conventional programming. Likewise, some of open systems and apparatuses can't be used. Albeit there are various specialists about propose approaches for OO programming testing, there are couple of studies for mix testing or blend testing.

* Computer Science (Software Engineering), Rungta College of Engineering of Technology, Bhilai, Chhattisgarh, India. Email: greetesh@gmail.com

** Assistant Professor, (Department of computer science), Rungta College of Engineering of Technology, Bhilai, Chhattisgarh, India. Email: neelabhsao@gmail.com

Mix testing of the OO programming must be executed in light of the way that a bit of the bugs stay covered up amid unit testing and pop-up entirely when coordination for a specific manifestation of bunch [1]. Test scope investigation is a basic issue for joining testing. Improperly, there are couples of examines which relate to test scope examination of combination testing. Some creator proposed an approach to manage dissect the test scope, yet this examination does not indicate to class scope which is a discriminating scope measure [2].

This paper proposes design of an instrument for checking coordination testing scope of OO programming which can check the tried classes and systems and give scope measures that contain rate of called classes each every tried class and rate of called system each every tried strategy. Likewise, our gadget can make experiments by getting succession graphs and utilization case portrayal from database to construct the scope of joining testing in the event that that the present experiments can't cover the code.

Whatever is left of this paper is created as takes after. Segment 2 elucidates the related foundation learning while Section 3 portrays revive that are related to mix test scope of OO. Segment 4 demonstrates the auxiliary arranging of our instrument and its segments. Finally, conclusion and future work are delineated in area 5.

Background

This zone exhibits the foundation data which is imperative for this paper. The outline of coordination testing of item situated, test scope, and arrangement chart are specified in this segment.

Integration Testing of OO Software

The reconciliation testing of OO programming [1] is concerned with testing collaboration between units. The central reason for this test level is to check relationship between classes, so the inside at this level is on trying collaboration between classes through technique calls or option sign, furthermore participation with database or hardware (H/W). Joining testing is required in light of the fact that a couple of deformities are not appeared in unit testing; then again they are found mix testing. The incorporated unit can be a class or techniques however various analyze connote the unit of OO programming as a class.

Test Coverage Analysis

A test scope [3, 4] is a quality affirmation metric which chooses the exhaustiveness nature of the executed experiment of a bundle. Test scope examination can be joined with every level of testing: unit testing, mix testing, structure or framework testing. The yield of the test scope examination shows uncovered test parts and use for upgrading the test methodology, for instance, sort out experiments, construct or decrease experiment. In test scope examination strategy, code instrumentation is performed by inserting some additional code into a framework to gauge scope results. Along these lines, the scope data are assembled in the midst of test runtime. By then, the accumulated scope results are inspected and used for giving test strategy proposition remembering the finished objective to decrease, to manage or to adjust the fitting test suite.

Automatic Test Case Generation

The UML model system is most comprehensively used as a piece of programming design stage. The composition study exhibits that UML blueprints are normally used to program make experiment era. Different approaches for making experiments in a roundabout way from the UML examination and layout models, i.e. utilization case graph, gathering framework, composed exertion diagram, class outline, state chart or a combinational strategy are proposed. At any rate, a huge segment of them need to make an understanding of UML delineation into another portrayal, for instance, a chart (model based testing) or a table and after that surmise the experiments. This paper brings couple of such existing techniques.

TestNG Framework

TestNG is an testing structure expected to streamline an extensive extent of testing needs, from unit (testing a class in segregation of the others) to coordination (testing entire structures made of a couple of classes, a couple of packs and even a couple outside structures, for instance, application servers).

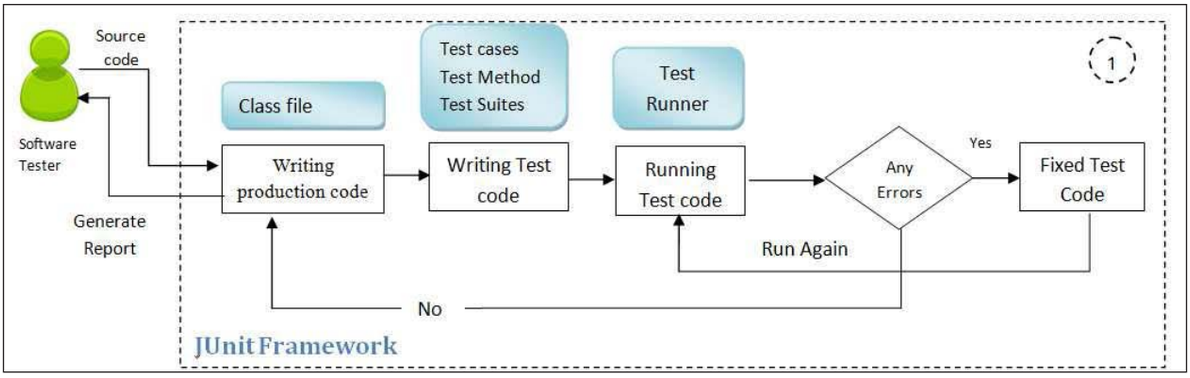

Figure 1 **TestNG Framework Flow**

TestNG (Next Generation) testing framework which, enthused by NUnit and JUnit, yet exhibiting various new innovative helpfulness like dependence testing, gathering thought to make testing all the more extreme and less difficult to do. It is proposed to cover all classes of tests: unit, integration, end-to- end, functional, etc.

Making a test is frequently a three-stage process:

- Make the business logic out of your test and TestNG annotations should be inserted in your code.
- Include the info about your test (e.g. the class name, the, run the groups as you wish, etc...) in a testng. xml record or in build.xml.
- Execute TestNG.

Sequence Diagram

Succession graph portrays the took part an interest question or articles and messages that send between them [5]. It gets the solicitation of frameworks from each thing or item, moreover the appeal in which it happens [6]. In UML-portrayed, OO programming, arrangement graphs is one of two outlines which is the reason for entomb class testing [7]. The contains two standard segments: protest and messages. The grouping chart involves progression of cooperation between different articles which is shown by rectangle. Another segment of arrangement chart is a message which is used to impart to the association between diverse things or article. Messages are shown by coordinated bolt and focus procedures which are sent between articles presented over the bolt [6].

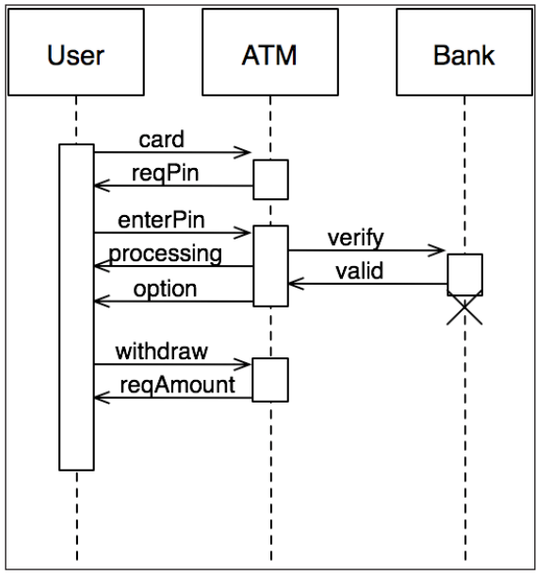

Figure 2 **Example ATM Sub Module Sequence Diagram**

RELATED WORK

The reconciliation testing's major point is not simply an approach, test scope is likewise key. Unpardonably, there are couples of examines proposed test scope. Analyst proposed an approach to manage perform the scope investigation of representation to help scope examination [2]. The OO scope a measure which is one yield of this investigation embodies strategy scope, polymorphic scope, legacy scope, and total procedure call scope. In any case, the yields need class scope that is a basic scope measure of OO coordination testing. What's more,

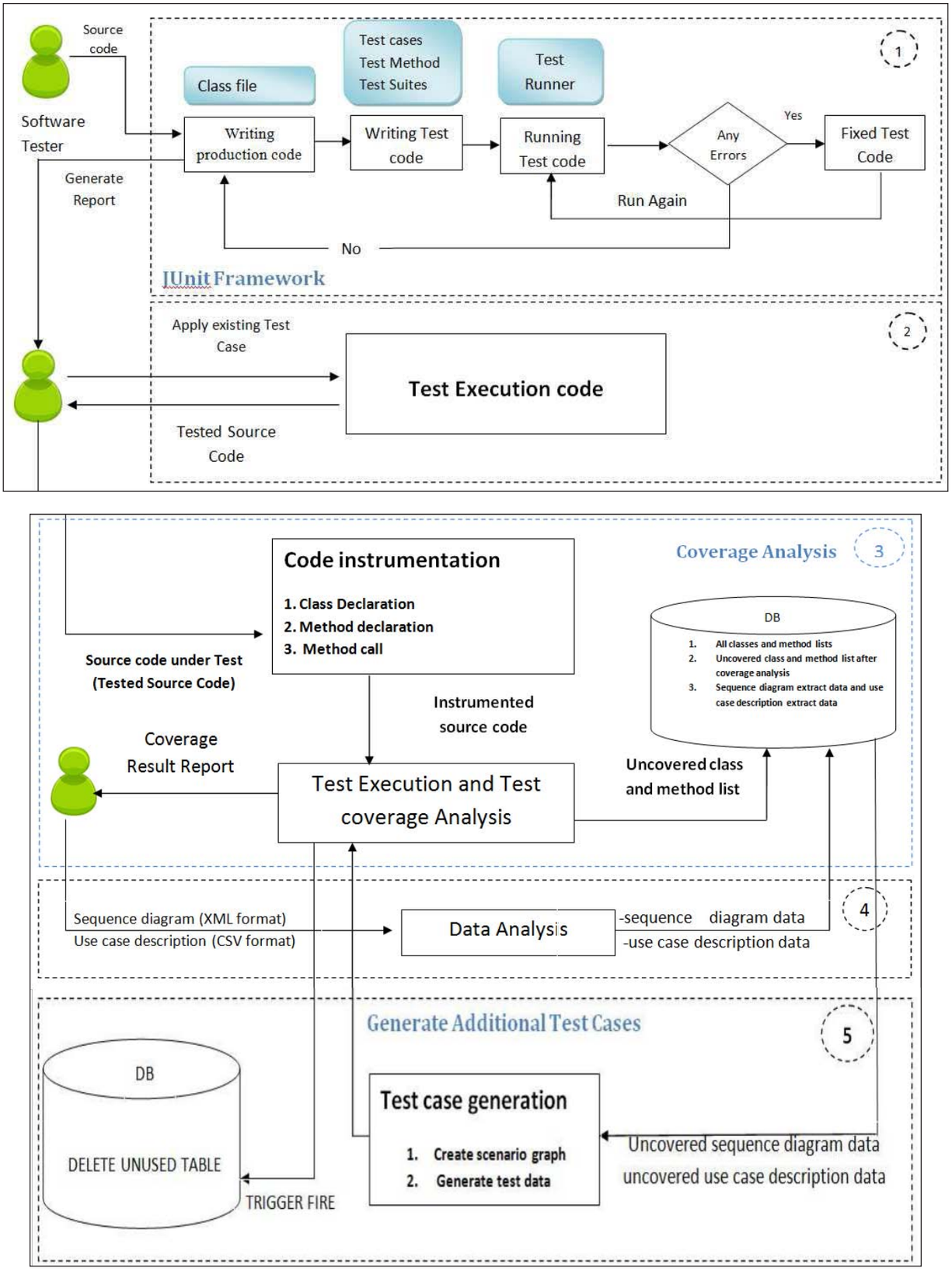

Figure 3 **Structure of Proposed Tool Using Sandwich Approach in TestNG Framework**

this investigation focuses on the scope examination, it doesn't indicate to making experiments to construct the scope measures of the tried programming. Nowadays, the OO writing computer programs is boundless and complex, so manual test scope investigation is genuinely troublesome. Creator proposed investigates which accumulate instruments for test scope investigation. The said devices are certainly insufficient and there is not the test scope investigation device for between class testing of OO programming. Likewise, there is not an instrument which creates experiments to extend incorporation test scope.

From the presented work, we found there is not any instrument for dismembering the test scope for between class testing of OO programming. What's more, the present device focuses on test scope examination. There is not a devices which makes experiments to extend the test scope measure. In like manner, we point arranging an instruments for checking the consolidation testing scope of thing organized programming and make additional experiments in the event that that the present experiments can't cover the code.

METHODOLOGY

This part exhibits framework of a device for checking the combination testing scope of OO programming. This apparatus can check entomb class testing scope of OO programming and deliver additional experiments on the off chance that the current experiments can't cover the classes and technique. Our apparatus involves 4 sections: Code instrumentation, Test execution in TestNG structure, Data investigation, and experiment time.

Test Coverage Analysis

Test scope examination methodology includes 2 modules: Code instrumentation and furthermore test execution and test scope report.

Code Instrumentation

In this part, the tried source code is instrumented to affirm which parts of the code are tried when test execution has been finished. The tried source code is filtered line by line and instrumented. After this part, the instrumented source code is sent to test execution part. The source code part which must be instrumented contains 3 sorts: formation of Class, making of Method, and Method called:

Algorithm for code instrumentation Input

Java file Output:
Class coverage in percentage;
Method coverage in percentage;
Individual class coverage in percentage;

Method:

Step 1: Create new java classes from existing software;
Class names same as existing java class;
Calculate no of method in a java file using java parser API
Create Two Dim String Array [n][n];
Assign "method" in Array of String [0] [0] index;
Assign "count" in Array of String [0] [1] index;

Step 2: method creation
Create instance variables same as method define in a class;
Assign 0 to all instance variables for individual classes;
Instance variable increment by 1 inside method body;

Step 3: Method called
Called objects property incremented by 1

Step 4: Call finalize method that write coverage report into CSV file;

Result:

Generate summary Report;

Creation of Classes

Exactly when the source code is "class creation", our apparatus will implant code which makes a table to accumulate system people from the made class after test execution.

Creation of Methods

Exactly when the source code is "strategy creation", our instrument will insert code which makes counter

variables and fills technique's name into the table which is made at class creation after test execution.

Method Call

Exactly when the source code is "strategy Called", our instrument add-on code which counters variables to demonstrate the system called

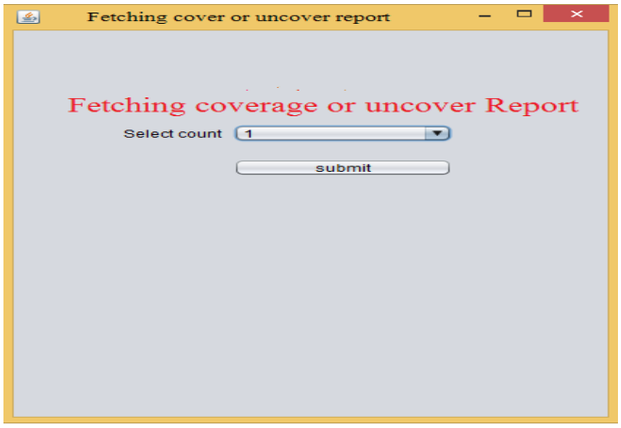

Figure 4 Fetched Report of Cover or Uncovered Classes and Methods

Additionally Test Execution and Test Coverage Report

Instrumented source code is done to the test execution part after code instrumentation part. In this part, the instrumented code is executed after test execution. Right when test execution has been fulfillment, our device will accumulate valuations of the counter variable. We make an uncovered class and technique list in case tables and condense rate of class scope moreover, system scope and return them to customers. A brief while later, at there are uncovered classes or frameworks from test execution sending to the database. A scope model can be exhibited as takes after.

$$\text{Class Coverage} = \frac{\text{Total no of covered class}}{\text{Total no of classes in the software system}} \text{ X } 100 \quad (1)$$

$$\text{Individual class coverage} = \frac{\text{Total no of covered method in a class}}{\text{Total no of method in a class}} \text{X } 100 \quad (2)$$

The secured class is the class which each part member method is called at any rate once.

$$\text{Method Coverage} = \frac{\text{Total no of covered method}}{\text{Total no of method in the System}} \text{ X } 100 \quad (3)$$

The secured method is the method which is called at any rate once.

After the tool computes the coverage rate, it shows the class coverage rate, method coverage rate, uncover class name, and reveal method name to the analyzer. Additionally, the analyzer can take the report in html position. The report comprises of the coverage rate also as reveal class and method name.

Test Case Generation

Experiment time involves 2 modules: Data investigation likewise, experiment era. This technique is performed in case there are uncovered classes or routines in order to augmentation class scope and strategy scope rate. Grouping outline in XML plan and utilization case portrayals which are changed from [9] in CSV setup are required from the analyzer to deliver additional experiments.

Experiment time method is performed by importing grouping chart and utilization case depiction from the analyzer. After that, arrangement chart purposes of interest and utilization case portrayals' subtle element

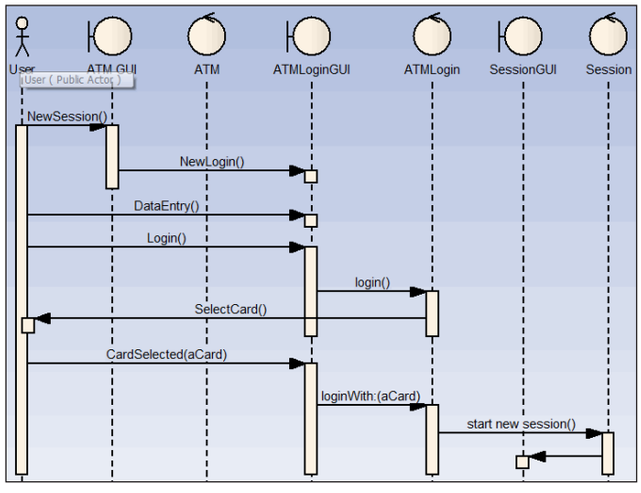

Figure 5 Sequence Diagram of ATM System

Id	first_name	last_name	rfid	pin	balance
1	Harry	Beyel	4294950819	1234	6704.00
2	Harsh	Chamria	4294937863	4760	99923403.99
3	Shop	Shopper	1000	0000	69932.00
4	Bruce	Land	4294940591	5555	1000.00

Figure 6 **Use Case Description of ATM System**

components are brought in data examination module. By then, the isolated subtle element components are saved to the database. From that point, reveal class and strategy name list which is made in test execution and test scope module is used for searching for the purposes of enthusiasm of related arrangement chart and utilization case depiction. By then, the subtle element components are used to deliver experiments in experiment period module. A brief while later, the created experiments are kept in a predefined envelope, Customer interface of presenting arrangement graph and utilization case delineations to deliver experiments can be demonstrated..

Data Analysis

Shown In figure 4, after the tool gets sequence diagram and use case description, the tool fetches the sequence diagram points of interest in request to create a scenario graph. Since a use case description is an acknowledgment of a sequence diagram [8], the points of interest in the use case description are additionally extracted to produce test information. The extracted points of interest from sequence graphs which are utilized to create test scenario comprise of request of system, method name, class name, and collaboration requirements of the consolidate piece. Also, the tool utilizes all points of interest as a part of use case description to create test data.

Test Case Generation

Shown in figure 5, at the point when points of interest of related sequence diagram and use case description are chosen from the database, the sequence diagram points of interest are utilized to make a scenario chart. The scenario diagram is a directed chart which a method name is spoken to by a node and a control stream is spoken to by an edge. The scenario diagram of the sample framework which is shown by the sequence graph

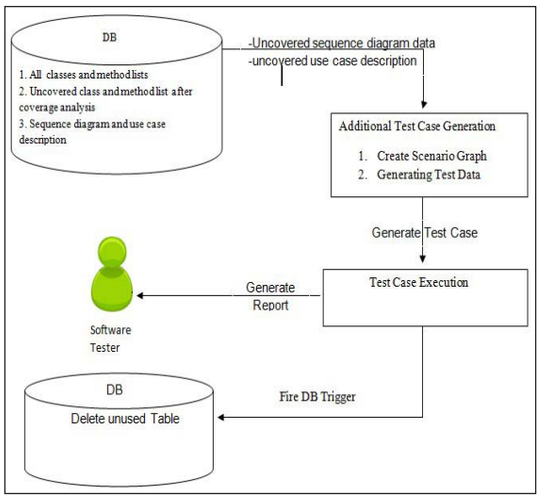

Figure 7 **Generate Additional Test Case**

RESULT AND DISCUSSION

After the instrument source code has done its send to the under test, the instrumented source code is implemented and unique test case are utilized. At the point when test execution is done, the counter variables and their data are concise and kept in touch with method called synopsis record by the source code that embedded before leaving program direction. From the sample framework which the analyzer picks in Fig. 2, an illustration of a method called rundown record can be demonstrated in Fig.4. As per the Fig. 6 the method div and mul of class autogen. Div, autogen.Mul is not called.

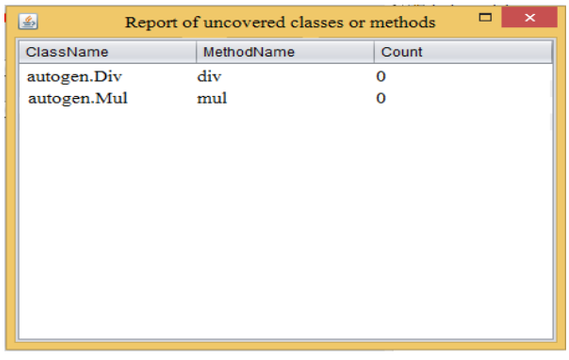

Figure 8 **Report of Uncovered Classes or Method**

After that, the tool utilizes method called synopsis document to make uncovered list and method name list. The rundown is utilized as a part of test case era process. At that point, the instrument examines the class coverage rate and

method scope rate using formula (1) and (2). The coverage criteria are indicated as takes after: [8]

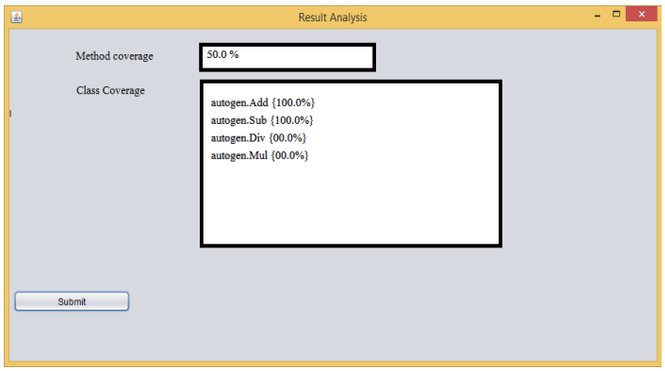

Figure 9 User Interface of Integration Testing Coverage Result

Coverage Report

Coverage Analysis

coverage class	50%
coverage method	50%

Uncover Class and Method

uncover class	*uncover method*
Autogen.Div	div
Autogen.Mul	mul

Figure10 An Example of Integration Testing Coverage

Figure 6 and figure 7 After the scenario chart is made, the condition method which contains the method which has not been called yet is looked. For the illustration in Fig. 10, the uncovered method is div,mul so the scenario path comprises of node [First, Main, decision=compute(choice), decision(add), decision(sub), decision(mul), decision(div), Println (value), Last]. At the point when the situation way is discovered, the device utilizes the subtle elements as a part of a utilization case portrayal to produce test information in html form. A created experiment comprises of experiment ID, Test case name, Test case depiction, Input Value, and Expected output. An illustration of produced test cases of illustration framework which the uncovered method is "div","mul" can be indicated in Fig.11

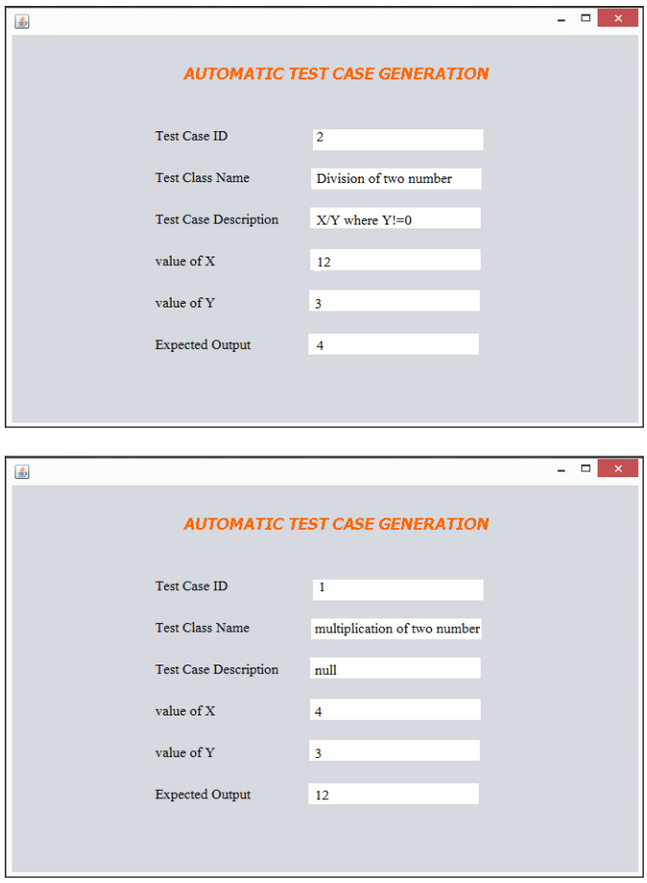

Figure 11 Additional Test Case Generation

CONCLUSION AND FUTURE SCOPE

This thesis introduces design and implementation of a tool for inter-class integration testing coverage analysis of object-oriented software using sandwich approach. This approach based on top down and bottom up strategy. Whereas some coverage (method covered) are solving within top down approach and some are on bottom up (class coverage). Our tool can examine the test scope of inter class testing. Additionally, it can create test cases on the off chance that the current test case can't cover the code to build the coverage measure. The advantages of this tool are expanding dependability and execution of inter class testing of OO programming and additionally cost and efficient.

This instrument can investigate the inter class testing coverage of OO programming and demonstrate the report of scope analysis. Furthermore, this tool can produce test cases in the event that there are revealed classes

or routines or method to build scope measure. This apparatus can build the dependability of inter class testing and decrease cost and time for examining the testing scope.

For future work, we will apply this tool using different approaches to find out the which approach is best that can be increasing reliability and performance of integration testing of object-oriented software as well as cost and time saving

.Afterward, we will test our tool by 3 object-oriented systems through cloud services. This tool also examines other object oriented languages like C#, C++, Python.

REFERENCES

Tsantalis, N., Chatzigeorgiou, A., & Stephanides, G. (2005). Predicting the probability of change in object-oriented systems. *IEEE Trans. on Soft. Eng.,* 31(7), 601-614.

Bansiya, J., & Davis, C. (2002). A hierarchical model for object- orienteddesign quality assessment, *IEEE Trans. Software Eng.,* 28(1), 4-17.

Sharafat, A. R., & Tahvildari, L. (2007). A probabilistic approachto predict changes in object-oriented software systems, *IEEE CSMR'07,* (pp. 27-38).

Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Transactions on Software Engineering*.

Briand, L.C., Rehman, M. J. U., Asghar, H., Iqbal, M. Z. Z.,& Nadeem, A. (2007). A state based approach to integration testing based on UML models. *Information and Software Technology,* 9, 1087–1106.

Lee, M., Offutt, J., & Alexander, R. T. (1996). Algorithmic analysis of the impact of changes to object-oriented software. *In Proceedings of the IEEE International Conference on Software Maintenance (ICSM),* pp. 171-184.

Hassan, A. E., & Holt, R. C. (2004). Predicting change propagation in software systems. *In Proceedings of the IEEE International Conference on Software Maintenance* (ICSM) (pp. 284–293).

Version Management with CVS for CVS 1.11.21, (2005). Free Software Foundation Inc., Retrieved from http://ximbiot.com/cvs/manual/.

Xia, F. (2004). A change impact dependency measure for predicting the maintainability of source code. I*n Proceedings of the Annual International Computer Software and Applications Conference (COMPSAC),* 2, 23-24.

Lee, Y., Yang, J., & Chang, K. H. (2007). *Metrics and Evolution in Open Source Software*. Seventh International Conference on Quality Software (QSIC 2007).

Sato, D., Goldman, A., & Kon, F. (2007). Tracking the evolution of object-oriented quality metrics on agile projects. *Proceedings of the 8th international conference on Agile processes in software engineering and extreme programming* (pp. 84-92).

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Trans. Software Eng.,* 20(6), 476-493, 1994.

Song, O., Sheppard, M., Cartwright, M., & Mair, C. (2006). Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering,* 32(2), 69-82.

Fenton, N., & Ohlsson, N. (2000). Quantitative analysis of faults and failures in a complex software system. *IEEE transactions on software engineering,* 26(8), 797-814. doi:10.1109/32.879815, 2000.

IEEE Standard Glossary of Software Engineering Terminology (IEEEStd 610.12-1990)

Mehdi Kessis, Yves Ledru, Gérard Vandome, "Experiences in coveragetesting of a Java middleware," in Proc. 5th international workshop onSoftware engineering and middleware., Portugal, 2005, pp. 39-45.

Shahid, M., & Ibrahim. (2011). An evaluation of test coverage tools in software testing. *In Proceedings 2011 International Conference on Telecommunication Technology and Applications.,* Singapore ?(pp. 216-222).

Najumudheen, E. S. F., Mall, R., & Samant, D. (2009). *A dependence graph-based test coverage analysis technique for object- oriented programs. In 2009 Sixth International Conference on Information Technology: New Generations* (pp. 763-768). USA.,

Dennis, A., Wixom, B. H., & Tegarden, D. (2010). Interaction diagrams, *In Systems Analysis and Design with UML An Object-Oriented Approach (3rd ed.).* (pp. 240-245). New Jersey, John Wiley & Sons.

Jorgensen, P. C. (2008). Object-Oriented Integration testing. In *Software Testing A Craftsman's Approach* (3rd ed.), New York, Auerbach Publications, (pp. 311-313).

Augsornsri, A., Suwannasart, T.(2013). Design of a tool for checking integration testing coverage of object-oriented software. *In 2013 International Conference on Information Science and Applications,* ICISA (pp. 1-4).

Leeraharattanarak, S. (2007). An approach for automatically generating test cases from use cases, M.S. thesis, Chulalongkorn Univ., Bangkok, Thailand, 2004. ISTQB. *International Software Testing Qualification Board* version 2.0, Retrieved from www.istqb.org

Grinwald, R., Harel, E., Orgad, M., Ur, S, & Ziv, A. (1998). *User defined coverage – A tool supported methodology for design verification*. IBM Research Lab, Haifa Dac, San Francisco, CA USA, (pp. 158-163).

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing Systematic Literature Reviews in Software Engineering,* EBSE Technical report, Keel University, UK.

Williams, B. S. A. L. (2008). *A survey on code coverage as a stopping criterion for unit testing.* Technical report (North Carolina State University. Dept. of Computer Science), TR-2008-22.

Yang, Q., Li, J. J., & Weiss, D. M. (2009). A survey of coverage-based testing tools. *The Computer Journal,* 52(5), 589-597.

Prasanna, M., Sivanandam, S. N., Venkatesan, R., & Sundarrajan, R. (2005). A survey on automatic test case generation. *Academic Open Internet Journal (AOIJ),* 15.

Mockus, A., Nagappan, N., & Dinh-Trong, T. T. (2009). Test Coverage and Post-Verification Defects: A Multiple Case Study. In *Proceedings of the ACM-IEEE Empirical Software Engineering and Measurement Conference (ESEM),* IEEE Computer Society.

Silva, L., & Soares, S. (2009). Analyzing structure-based techniques for test coverage on a J2ME software product line. *In 10th Latin American Test Workshop,* LATW '09. pp.1-6.

Wei, Y. (2008). Is Coverage a Good Measure of Testing Effectiveness? *Chair of software engineering ETH Zurich,* CH-8092 Zurich, Switzerland.

Derezińska, A. (2008). Experiences from an Empirical Study of Programs Code Coverage. Book section, *Advances in Computer and Information Sciences and Engineering*, 57-62, DOI: 10.1007/978- 1-4020- 8741-7_11

Jun-Ru, C., & Chin-Yu, H. (2007). A study of enhanced MC/DC coverage criterion for software testing. *In 31st Annual International Computer Software and Applications Conference* (COMPSAC), (pp. 457-464)

Berner, S., Weber, R., & Kellar, R. K. (2007). Enhancing software testing by judicious use of code coverage information. *In 29th International Conference on Software Engineering.*

Lloyd, E. L. (2005). A Study of Test Coverage Adequacy in the Presence of Stubs. *In the Journal of Object Technology,* 4, 117-137

Lawrance, J., Clarke, S., Burnett, M., & Rothermel, G. (2005). How Well Do Professional Developers Test with Code Coverage Visualizations? An Empirical Study. *IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 53-60). Dallas, TX.

Kim, Y. W. (2003). Efficient use of code coverage in large-scale software development. *In the Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research CASCON* '03.