International Journal of Trend in Research and Development, Volume 2(2), ISSN: 2394-9333
www.ijtrd.com

Test Series Extent for Structural Coverage in Software Testing Using Multiple Techniques

[1]D.Khadar Hussain, [2]S.Md.Haroon, [3]M.Rajesh Babu, [4]C.Nagarjuna
[1,2,3,4]M.Tech,PG Scholar, CSE Department
[1,3,4]JNTUA College of Engineering, Anantapuramu-515002. A.P.

Abstract

Testing is an essential phase of software development life cycle (SDLC). A series of function calls is required to test software. When there is internal state in the software, there might be a need for invoking previously invoked functions and then call a method in order to ensure the correct internal state. Software testing is a method of assessing the functionality of a software platform. There are several different types of software testing, but the two major kinds are dynamic testing and static testing. In software testing consideration of internal state of software is important both in procedural and object oriented approaches. The length of test sequences in order to ensure complete structural coverage in the software is the area less studied. This paper aims at analyzing the role of length of test sequences for complete branch coverage. Towards achieving it we use multiple techniques. Empirical results revealed that the length of test sequences has its role to play in software testing.

Keywords **–** SDLC, software testing, search-based techniques, test case, test sequence, test suite

I. INTRODUCTION

Software testing has become a branch of computer science and much importance is given for it as it ensures the quality of software. Software testing is a method of assessing the functionality of a software program. There are many kinds of testing. Black box testing is a process of testing the software to see whether it gives intended outputs. The white box testing focuses on generating a suite of test cases in order to ensure the complete structural coverage in terms all branches in the software [1]. Test case is nothing but a program which invokes a function and with test input standards. The output of the method call is compared with the expected output. If both are same, the test case is success else failure. This way the functionalities in the software are tested with all possible input values even wrong input values. Thus the robustness of the software is tested. When the software is found with inconsistencies, then the test engineer sends it back to the development team thus life cycle of software development starts again.

When the software has internal state, testing requires more test cases. This is because, the software is likely to have many branches and all are to be covered to have complete structural coverage. In order to ensure that the internal state is present correctly and then test all branches needs more test cases i.e. test sequences or test suites.

Fig. 1 – A sample program with internal state

```
public class BankAccount
private double balance;
private int number OfWithdrawls;
public void deposit ( double amount)
{
If (amount > 0.00)
balance = balance + amount;
}
public withdraw (double amount)
{
If (amount > balance) {
printError();
return;
}
If (numberOfWithdrawls >=10) {
printError();
return;
}
dispense (amount);
balance = balance – amount;
numberOfWuthdrawls++;
}
}
```

As seen in fig. 1, the bank account program has internal state i.e. number Of Withdrawals variable. As withdrawals are allows only 10 times, to have complete branch coverage, the method needs to be called minimum of 10 times so as to test the condition effectively. Therefore it is essential to know the

shortest possible sequence length that covers all branches in the program.

In this paper, we study the role of the test sequence length to have complete structural coverage. As described in [2] there is a common practice that applies random testing then followed by other techniques to cover all branches? This has been proven in [3]. In this context two things are important. First of all a decision has to be made as to how long the random testing is to be applied. Second, effort required to cover all branches. According to [4] we can target a single branch at any given point of time but keeping track of rest of the branches might be required. Only the second one is the focus of this paper. After generating test cases through random testing, specialized test sequences are required to cover rest of the branches. In the process we try to reduce number of test cases while ensuring complete branch coverage. In [5] search based techniques are used to test software. The author of [5] compared four algorithms namely Random Search (RS), Genetic Algorithm (GA), Evolutionary Algorithm (EA), and Hill Climbing (HC). Bench mark container classes in Java language [6-17], are used here to perform testing. The author of [5] also performed empirical and theoretical analysis to know the length of test sequences.

The contribution of this paper to use more techniques in order to find the role of the length of test sequences for complete structural coverage of the software under test. The remainder of this paper is structured as follows. Section II reviews literature on the role of test sequence length in software testing. Section III provides insights into the proposed techniques. Section IV analyzes experiments and results while section V concludes the paper.

II. PRIOR WORK

In the literature there are prior works that focused on test sequences [8], [9], [10]. In the process of analyzing the length of test sequences, they used various techniques to prune the search space in order to find shortest solution. In [7] a new approach is followed to reduce test sequences. That approach suggests having data structures equipped with a model checker. Evolutionary techniques are used to reduce the number of test sequences. In such experiments fitness is calculated using heuristics. Branch distance and approach level are the typical heuristics found in the literature [18]. However, only approach level is used in [6] which can find length of test sequences but can't minimize the test sequences required for structural coverage. Same work was carried out in [11] but with the help of Genetic Programming. Here also the fitness function is not aimed at reducing number of test sequences. Object reuse and purity analysis are the techniques used by Ribeiro et al. [15], [16] in order to reduce search space. In [12] a combined approach was studied. First of all the problems encountered in [8], [9], and [10] are overcome using evolutionary algorithm and then symbolic execution approach is used to get rid of problems in evolutionary algorithm. For testing object oriented software, search algorithms are used in [13], [14] and [5]. Table 1 summarizes the research carried out current state of the art in software testing that focuses in finding the role of the length of test sequences. Andrea Arcuri 2012 100% [5]

Table – Summary of Related Work

Authors	Year	Containers	References
Tonella	2004	83%	[34]
Visser et al.	2004	100%	[35]
Xie et al.	2004	81%	[39]
Xie et al.	2005	100%	[40]
Visser et al.	2006	100%	[36]
Wappler and Wegner	2006	100%	[37]
Inkumsah and Xie	2008	77%	[21]
Arcuri and Yao	2008	100%	[11]
Arcuri	2009	100%	[3]
Ribeiro et al.	2009	100%	[32]
Ribeiro et al.	2010	100%	[33]
Baresi et al.	2010	66%	[12]

More research went into search algorithms [19], [20] but very less was dedicated to apply those techniques for software testing. The exceptions include [21], [22] and [23] where some attempt is made to use search algorithms in software testing. In [5] four search techniques are used for software testing. This paper uses multiple techniques to study the role of the

length of test sequences for complete structural coverage.

III. PROPOSED WORK

The application of techniques such as RS, HC, EA, and GA are as described in [5]. However, in this paper we implemented a variation of GA with different search operator. Instead of using mutation uniformly, higher probabilities are given for function call mutating in the process of finding test sequence length. For this purpose, the GA algorithm which has been presented in [5] is modified.

Fig. 2 – Pseudo code for GA (Excerpt from [5])

Algorithm 1 A Genetic Algorithm Pseudo-Code

1: Choose an initial random population of individuals
2: Evaluate the fitness of the individuals
3: **repeat**
4: Select the *best* individuals to be used by the genetic operators
5: Generate new individuals using crossover and mutation
6: Evaluate the fitness of the new individuals
7: Replace the *worst* individuals of the population by the best new individuals
8: **until** some stop criteria

As can be seen in fig. 2, it is evident that the GA algorithm is using uniform mutation. We changed in our prototype built in Java to give higher probabilities for first function call mutations in the test sequence. The results of the techniques are presented in the next section.

IV. EXPERIMENTAL RESULTS

Experiments are made with a prototype application that has been built using Java programming language with Graphical User Interface (GUI). The application is tested in a PC with 3 GB RAM and Core 2 Dual processor. The datasets or software for testing are taken from Java collection API and also other source codes. The results are presented below.

Fig. 3 – Average Number of Method Calls

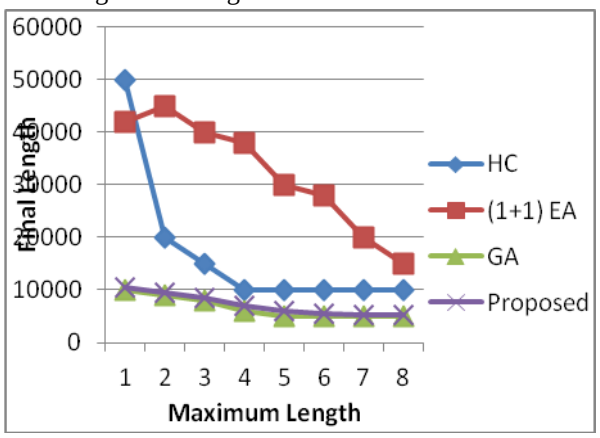


Fig. 4 – Average Length of Final Solutions

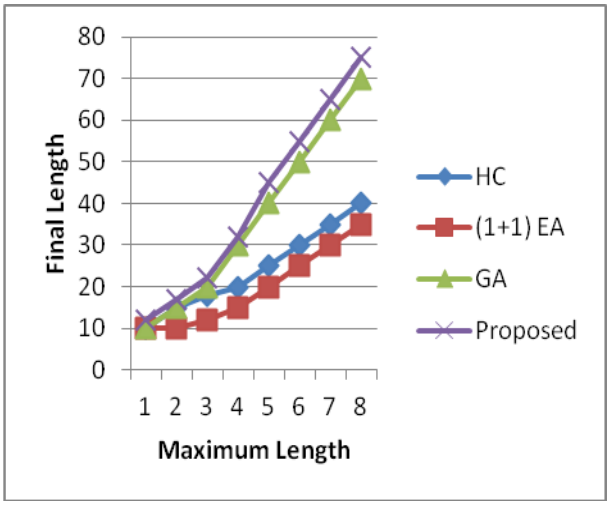


Fig. 5 – Sequence Length for Linked List

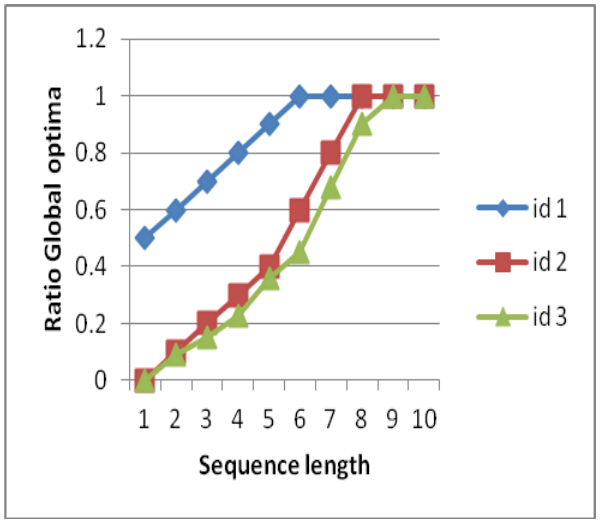


Fig. 6 – Load Factor of Hash table

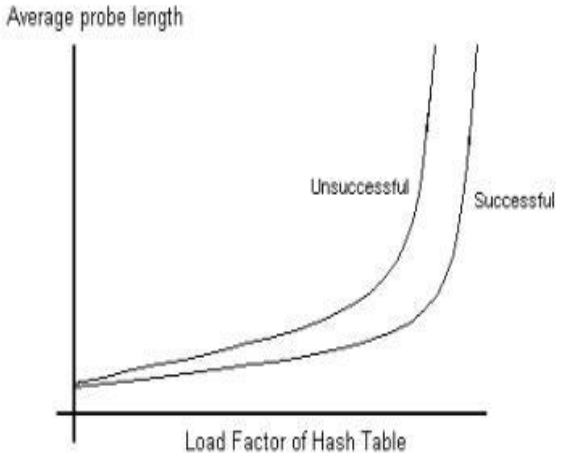
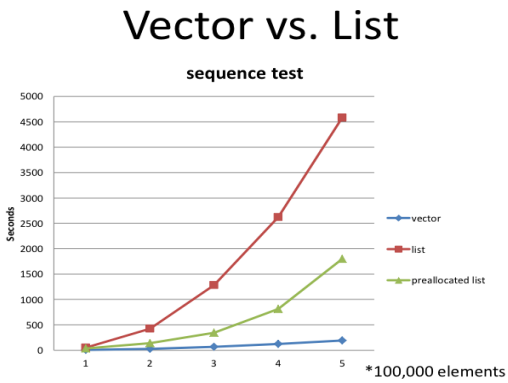
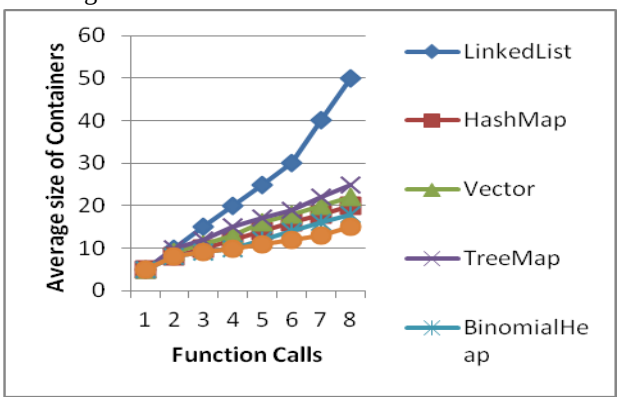


Fig. 7 – Diff b/w vector and list



As can be seen in fig. 7, we taken sequence test in x-axis under the same time this sequence test how to perform operation between vector vs list. Same as hash map vs vector etc.

Fig. 8 – Size of Container vs. Function Calls



V. Conclusion & Future Work

In this paper we have analyzed the role that the length of test sequences plays in software testing for structural coverage. We have empirically shown on common software testing benchmarks that having slightly longer sequences can drastically improve the results. The role of the test sequence length has received only little attention in the works. This paper provides the important role to rigorously support its importance. The findings of this paper can be easily exploited in most the techniques described in the literature. In fact, when choices about the length need to be taken in these techniques in the literature, our results suggest that using even slightly longer test sequence might have dramatic benefits. We used the search techniques such as EA, RS, HC, GA and other variations of GA. We have tested containers and other software products using a prototype application. The application is built using Java language in order to test software in user-friendly fashion. The software under test contains internal state as well. With internal state, we made experiments with various techniques in order to find the optimal test sequence length. During the experiments a number of issues were identified which could further improve the efficiency of the test methods presented. The role of the test sequence length has received only little attention in the works. The findings of this paper can be easily exploited in most the techniques described in the literature. In fact, when choices about the length need to be taken in these techniques in the literature, our results suggest that using even slightly longer test sequence might have dramatic benefits. The experimental results revealed that the proposed application is useful in testing software and generate test cases for full branch coverage.

Future work will follow two ways. First, representative software for which longer sequences give worse results need to be recognized and studied. Hybrid techniques that exploit variable length representation should be hence designed to work well on average. For example, a GA that uses a population of individuals with different lengths could be an appropriate first high-quality to analyze. Second, additional prescribed analyses on the role of the length are required to get a better understanding of which are the limitations and difficulties of software testing.

References

[1] G. Myers, the Art of Software Testing. Wiley, 1979.
[2] J.W. Duran and S.C. Ntafos, "An Evaluation of Random Testing," IEEE Trans. Software Eng., vol. 10, no. 4, pp. 438-444, July 1984.
[3] A. Arcuri, P. Lehre, and X. Yao, "Theoretical Runtime Analysis in Search Based Software Engineering," Technical Report CSR-09-04, Univ. of Birmingham, 2009.
[4] A. Arcuri, M.Z. Iqbal, and L. Briand, "Formal Analysis of the Effectiveness and Predictability of Random Testing," Proc. ACM Int'l Symp. Software Testing and Analysis, pp. 219-229, 2010.
[5] Andrea Arcuri, "A Theoretical and Empirical Analysis of the Role of Test Sequence Length in Software Testing for Structural Coverage", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 38, NO. 3, MAY/JUNE 2012.
[6] P. Tonella, "Evolutionary Testing of Classes," Proc. ACM Int'l Symp. Software Testing and Analysis, pp. 119-128, 2004.
[7] W. Visser, C.S. Pasareanu, and S. Khurshid, "Test Input Generation with Java Pathfinder," Proc. ACM Int'l Symp. Software Testing and Analysis, pp. 97-107, 2004.
[8] T. Xie, D. Marinov, and D. Notkin, "Rostra: A Framework for Detecting Redundant Object-Oriented Unit Tests," Proc. IEEE/ ACM Int'l Conf. Automated Software Eng., pp. 196-205, 2004.
[9] T. Xie, D. Marinov, W. Schulte, and D. Notkin, "Symstra: A Framework for Generating Object-Oriented Unit Tests Using Symbolic Execution," Proc. 11th Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems, pp. 365-381, 2005.
[10] W. Visser, C.S. Pasareanu, and R. Pela`nek, "Test Input Generation for Java Containers Using State Matching," Proc. ACM Int'l Symp. Software Testing and Analysis, pp. 37-48, 2006.
[11] S. Wappler and J. Wegener, "Evolutionary Unit Testing of Object- Oriented Software Using Strongly-Typed Genetic Programming," Proc. Genetic and Evolutionary Computation Conf., pp. 1925-1932, 2006.
[12] K. Inkumsah and T. Xie, "Improving Structural Testing of Object- Oriented Programs via Integrating Evolutionary Testing and Symbolic Execution," Proc. IEEE/ACM Int'l Conf. Automated Software Eng., pp. 297-306, 2008.
[13] A. Arcuri and X. Yao, "Search Based Software Testing of Object- Oriented Containers," Information Sciences, vol. 178, no. 15, pp. 3075-3095, 2008.
[14] A. Arcuri, "Insight Knowledge in Search Based Software Testing," Proc. Genetic and Evolutionary Computation Conf., pp. 1649-1656, 2009.
[15] J.C.B. Ribeiro, M.A. Zenha-Rela, and F.F. de Vega, "Test Case Evaluation and Input Domain Reduction Strategies for the Evolutionary Testing of Object-Oriented Software," Information and Software Technology, vol. 51, no. 11, pp. 1534-1548, 2009.
[16] J.C.B. Ribeiro, M.A. Zenha-Rela, and F.F. de Vega, "Enabling Object Reuse on Genetic Programming-Based Approaches to Object-Oriented Evolutionary Testing," Proc. European Conf. Genetic Programming, pp. 220-231, 2010.
[17] L. Baresi, P.L. Lanzi, and M. Miraz, "Testful: An Evolutionary Test Approach for Java," Proc. IEEE Int'l Conf. Software Testing, Verification and Validation, pp. 185-194, 2010.
[18] P. McMinn, "Search-Based Software Test Data Generation: A Survey," Software Testing, Verification and Reliability, vol. 14, no. 2, pp. 105-156, 2004.
[19] P.S. Oliveto, J. He, and X. Yao, "Time Complexity of Evolutionary Algorithms for Combinatorial Optimization: A Decade of Results," Int'l J. Automation and Computing, vol. 4, no. 3, pp. 281-293, 2007.
[20] P.K. Lehre and X. Yao, "Runtime Analysis of Search Heuristics on Software Engineering Problems," Frontiers of Computer Science in China, vol. 3, no. 1, pp. 64-72, 2009.
[21] P. Lehre and X. Yao, "Crossover Can Be Constructive When Computing Unique Input Output Sequences," Proc. Int'l Conf. Simulated Evolution and Learning, pp. 595-604, 2008.
[22] P.K. Lehre and X. Yao, "Runtime Analysis of (1þ1)EA on Computing Unique Input Output Sequences," Proc. IEEE Congress Evolutionary Computation, pp. 1882-1889, 2007.
[23] P.K. Lehre and X. Yao, "Runtime Analysis of the (1þ1)EA on Computing Unique Input Output Sequences," Information Sciences, pp. 1882-1889, 2010.