

Impact of Length of Test Sequence on Coverage in Software Testing

B.Raveendranath Singh

Professor – CSE

Visvesvaraya Institute of Technology & Science,
Hyderabad



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. 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.

Index Terms –SDLC, software testing, internal state, search-based techniques, test case, test sequence, test suite

INTRODUCTION

Software testing has become a branch of computer science and much importance is given for it as it ensures the quality of software. 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 values. The output of the function 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.

```
public class BankAccount {
    private double balance;
    private int numberOfWithdrawals;
    public void deposit(double amount) {
        if(amount > 0.00)
            balance = balance + amount;
    }
    public void withdraw(double amount) {
        if(amount > balance) {
            printError();
            return;
        }
        if(numberOfWithdrawals >= 10) {
            printError();
            return;
        }
        dispense(amount);
        balance = balance - amount;
        numberOfWithdrawals++;
    }
}
```

Fig. 1 –A sample program with internal state (excerpt from [12])

As seen in fig. 1, the bank account program has internal state i.e. numberOfWithdrawals variable. As withdrawals are allowed 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). Benchmark container classes in Java language [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [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 is 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.

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.

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

Andrea Arcuri 2012
 100% [5]

Table 1 –Summary of Related Work (excerpt from [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.

PROPOSED APPROACH

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.

```

Choose population  $K$  uniformly at random from  $S(l)$ .
while global optimum not found
  Copy best  $n$  solutions from  $K$  to  $K'$ .
  while  $K'$  is not completely filled
    Select 2 parents from  $K$  according to selection criterion,
    Generate 2 offspring that are copies of their parents,
    Apply crossover on offspring with probability  $P_{crossover}$ ,
    Mutate each offspring,
    Copy the 2 new offspring into  $K'$ .
   $K = K'$ .
  
```

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

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.

EXPERIMENTAL RESULTS

Experiments are made with a prototype application that has been built using Java programming language with Graphical User Interface (GUI). Other software used includes Net Beans IDE, JUnit and JDK 1.7. 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.

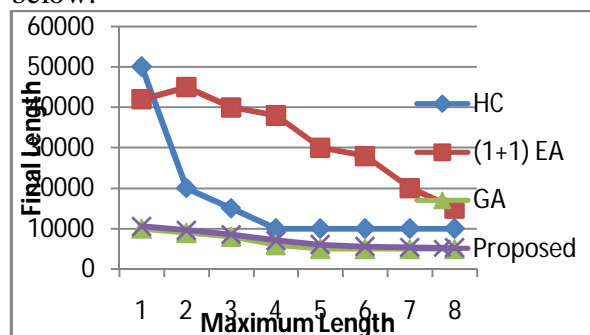


Fig. 3 –Average Number of Method Calls

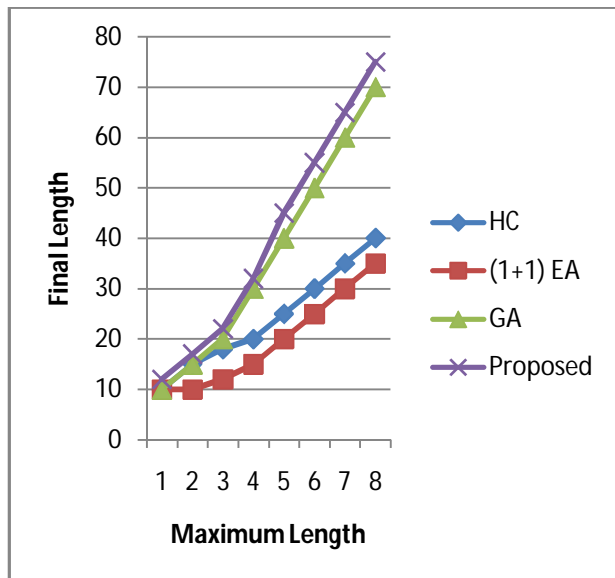


Fig. 4 – Average Length of Final Solutions

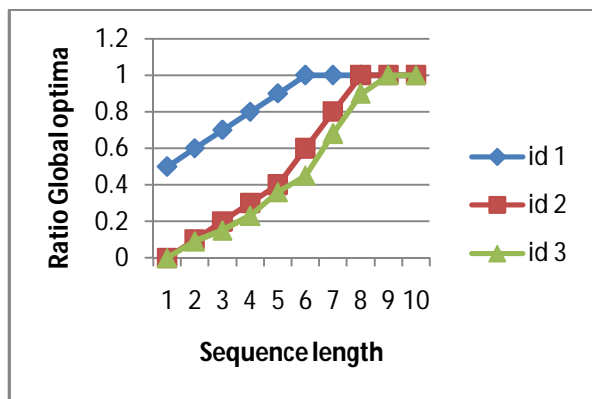


Fig. 5 – Sequence Length for Linked List

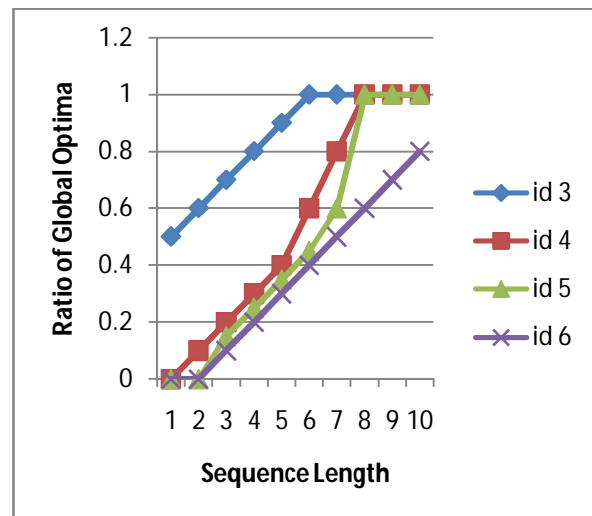


Fig. 6 – Sequence Length for Hashtable

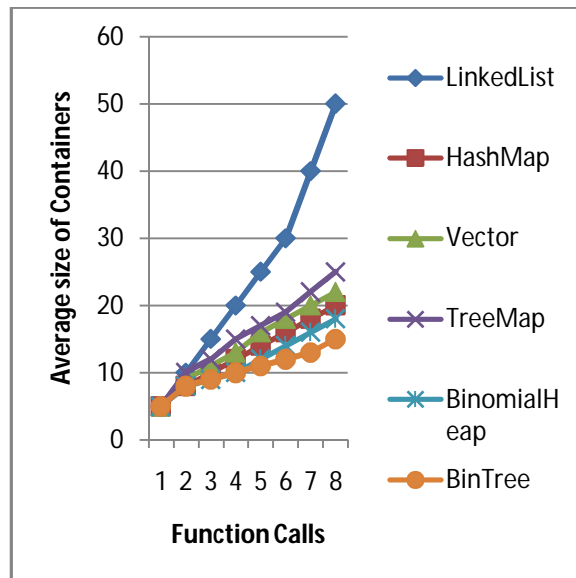


Fig. 7–Size of Container vs. Function Calls

V. CONCLUSION

In this paper, we studied on the role of the length of test sequences in software testing for complete structural coverage. We used the search techniques such as RS, HC, EA, 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. We found that test sequence length plays a crucial role in software testing for complete structural coverage. The experimental results revealed that the proposed application is useful in testing software and generate test cases for full branch coverage.

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. Pelánek, "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.