IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 3, Ver. VIII (May-Jun. 2014), PP 16-23
www.iosrjournals.org

Test Exude: Approach for Test Case Reduction

Vaibhav Chaurasia[1], Thirunavukkarasu K.[2]
[1]*Student - School of Computing Science and Engineering, Galgotias University, India*
[2]*Asst. Professor – School of Computing Science and Engineering, Galgotias University, India*

Abstract : *Software development is a planned and structured process includes research, new development, modification, re-engineering results in software products. The structure of development of software is SDLC. It contains various process and activities to accomplish the task of the software product. In SDLC, testing is the most expensive phase. It is used to validate the software with all possible combinations. Exhaustive testing is impossible task in software testing, as, it is impractical with large software products. Now, automation is applied to generate the test cases for various applications, but the selection and reduction of test cases by automation is the main problem. This target only be achieved, when test cases is contained in large test suites. The technique we propose for test case reduction, called TestExude. It is a technique which gives test cases needed and discard other test cases from the Test Suite. It reduces the redundant test cases and its corresponding elements. This technique works on frequency of test cases in test suite and accepts largest frequency occurred in test suites and discard test cases on the base of frequency selected. Storage is well managed in this technique by reducing the test cases drastically in test suite. Test Suites size, user time, organization cost, storage cost are reduce efficiently. Ultimately, it improves and generate effective test case. This technique is more advantageous than any other techniques as it reduces the cost and time to its minimum extent.*

Keywords: *Delegate Set, SDLC, Test Case, Test Case Reduction, TestExude, Test Suite*

I. Introduction

Software Development has seven phases to develop software. In SDLC (Software Development Life Cycle) Development phase and Testing phase takes maximum time and cost, both phases are equivalent to each other. Instead of equivalence Software Testing phase is most expensive phase due to the varieties of testing in the current scenario (for e.g., Regression Testing, Unit Testing, Alpha-testing and so on).

The area of Software Testing is huge and sources are limited, so, instead of overall testing Software Tester focus on Test Cases. According to the definition [3] “A test case, is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do”.

Reduction of the Test Cases is most challenging than Generating a Test Case. Test Cases completely depend on the length of the program. For instance, any Program contains 10000 Lines of Code, and then Test Cases also in thousands of number and its cost vary from thousands to lakhs.

The cost of testing each Test Case cannot afford by an every organization, so, optimization is needed over there. As, every phase of SDLC needs to be considered in development of software. To optimize the test cases, Test Suite is formed instead of individual Test Cases. Test Suite is defined as [4] “a test suite, less commonly known as a validation suite, is a collection of test cases that are intended to be used to test a software program to show that it has some specified set of behaviors”. Each Test Suite contains thousands of Test Cases and each Test Suite has its own goals and detailed information related to it.

Other than, optimization of Test Cases, effectiveness of test cases counts. Optimization be done on the basis of the effectiveness of the Test Cases i.e., extract only those Test Cases which is useful from the Test Suite.

Automated Test Case Generation are so developed that for any application or software, produce test cases but the core problem regarding Automated Test Case Generation is that, Generation of test cases are not effective one. It generates overall test cases, not according to the need by the Software Tester. So, tester needs such technique with which useful Test Cases can be extracted.

Test suite contains large amount of test cases which is redundant i.e., same set of Test Cases repeating. Our goal is to reduce redundancy and produce effective test case. It reduce the time and cost of the development organization. Complexity is also reduces as compare to the traditional Test Case Reduction Techniques.

Automation is up to some level, but the skill of the Tester is the big issue. Crucial resource remains the human factor [1]. Beyond the availability of advance technology, tester skill and commitment can make a big difference between successful and ineffective generation of the test case [1].

The skill is attribute which is developed by the knowledge. Knowledge also is big aspect and plays vital role. The definition of Knowledge in one aspect [5] “Knowledge is familiarity, awareness or understanding of someone or something, such as facts, information, description, or skills, which is acquired through experience or education by perceiving, discovering, or learning". Testers need to have knowledge in each area i.e., development and testing. Knowledge is developed only by the experience, studies told us [2] "Practical knowledge that is developed in direct observation or participation in activities".

The technique we propose a technique for test case reduction, called TestExude. This technique, calculate the frequency of the Test Cases occur in the Test Suit. It selects the highest frequency from the test suite and discards all the test conditions corresponding to it. Then, it calculates the new frequency of the test cases. The procedure continues till all the test conditions will discard. In case tie between the tests cases frequency, test case will select randomly.

In the field of reduction of test cases many algorithms are there which is effective but complex equally and creates confusion in the mind of the developer such as Dynamic Domain Reduction [8][9], Common Test Case Generator [10][11], Handling Constructs with Exceptions [12]. This algorithm based on statement coverage, covers all the statement from header files till the end of the source code.

As, we know very thoroughly that at the time of development, client change its requirement accordingly. In this case, developer has to do changes in their code. It is very dark side of the development and quality is also affected by changing the requirement again and again. So, TestExude handles or solve the problem of changing requirement in development scenario.

In the remaining part of paper, in Section II, we discuss Existing Technique for the same and limitations in that method. Section III contains, Proposed Algorithm, TestExude. Section IV contains the portion of the Experimental Study for the specific and result related to an algorithm. Section V contains the part of the Conclusion last part VI contains Future work of this technique.

II. Existing Technique

Filtration technique helps us to find effective test cases at the maintenance level of SDLC. It yields equivalent coverage with respect to some criteria [7].

The algorithm is as follows [6]:-

Algorithm

1) Inputs: Set of Requirements (SR): SR = $\{Req_i | i \in N, \quad i \leq m\}$

   Set of Test Cases (STC): each test case completely satisfies one or more requirements. STC = $\{tc_i | i \in N, i \leq n\}$

   Set of Test Suite (STS): STS = $\{TS_i | i \in N, i \leq m\}$

   Test Case (TS_k): It means that each test in the test suite satisfies the requirement Req_k

2) Output: Representative Set (RS).
3) Steps: The following steps in the algorithm are :-

- In Step 1 Weighted Set has been calculated of the test cases.
- Select the test case having highest weight. In case of tie, choose random between the two or more.
- Move $tc_h$to Representative Set, and mark the entire test suite from Set of Test Suite (STS). When all test suite is marked, then exit else repeat the steps until set suite marked.

1.1 Advantage and Disadvantage

This technique is more advantageous than any other techniques as it reduces the cost and time to its minimum extent. It also reduces the redundant and its corresponding elements. Storage is well managed in this technique and reduces the test suites drastically.

Instead of all the advantages, technique has limitation that, it works only on the Input given and Expected Output taken. In simple words, when user gives some input then, output is expected from the program and this technique work on the same.

In programming, there are the cases in which Input is not given by any user or there is no interference of any user. In these codes, input is generated automatically by the compiler. In that case, TestFilter [7] technique fails to reduce the test cases.

So, we have proposed a new technique which overcomes these limitations. In propose technique, these types of cases in code is handled successfully.

III. Proposed Technique

Limitation of the existing technique is that, it only works on the given set of exhaustive inputs given by the user. If, inputs taken dynamically by the system then, this technique not work appropriately.

For this purpose, new technique is proposed. This technique works on the frequency of the test cases (occurrence of test cases under certain test conditions).

TestExude is a technique which gives us test cases needed and discards other test cases from the Test Suite. The procedure of TestExude is as follows:-

Inputs:-

- Set of Requirements
- Set of Test Cases
- Set of Test Conditions
- Associated Test Cases

Outputs:-

- Delegate Set(DS)

Step 1:- Develop Initial Source Code according to the given requirement.

Step 2:- If requirement changes, then review the requirement, and do changes in the Source Code accordingly.

Step 3:- Decide the Requirements (Req_f) in the source code

Step 4:- According to the requirement (Req_f) and Test condition (TC_v), derive the associated test cases (tc_m)

Step 5:- Calculate the frequency of the test cases.

Frequency is the number of test condition occurred as follows :

$$\mathrm{Frequency}(\mathrm{tc_m}) = \sum_{i=1}^{n} \mathrm{Contains}\,(\mathrm{Domain}\,(\mathrm{TC_v}), \mathrm{tc_m}) \qquad - \quad (1)$$

TC_v is a test condition, m is varies from 1 to n.

Step 6:- Select the first test case has highest Frequency, if frequencies are the same of all test cases then, use random selection method.

Step 7:- Put every selected test case in the Delegate Set (DS) and discard other test cases and their relevant test cases.

Step 8:- Verify, the test case fulfills our requirement or not.

If
fulfill, then Extract all the elements of corresponding test case.
$tc_m \in TC_v$

Now, extract all the elements belong to TC_n
$TCv \rightarrow \{tc_{1\dots\dots\dots}tc_n\}$

else
Goto Step 6 and again follow same procedure.

IV. Experimental Study

In this part, study and analysis of the technique proposed is done. From this we conclude our final result. Mainly our concern is to reduce test cases for some special cases not the overall test cases.

Sometime, testing is to be done on special Data Sets and its output. In that case, we have to extract useful test cases from the Test Suite according to the given condition., for e.g. Only Prime numbers from the test suite given be or Even numbers from the given test suite. In this, only prime numbers are needed in a specific range. So, accordingly source code is written.

Now, the technique is proposed in this paper, we, evaluate them by a simple example.

2.1 Source Code

At first site, we have the different code :-

```
#include <stdio.h>
int main()
{       intn,j;
        printf("Numbers between 1 to 100\n");
        for(n= 1;n<= 100;n++)
        {           for(j=2;j<= 100;j++)
```

```
                {       if(n%j==0)
                        break;
                }
                if(j==n)
                {       printf("Prime Number :- %d \n",n);
                }
        }
        return 0;
}
```

In the program, Data Set is given in which Data Set contains even, odd and prime numbers as the requirement has been changed by third party. So, according to the test suite we have to modify our source code slightly according to the test suite Data Set. In this we have to extract theelements which is prime by nature. We have large number of Data Set contains in which it is difficult to give input for each and every element to check its status. So, simply we apply loop accordingly and extract all output in one go. Problem is that input is taken dynamically by the system. Although, code gives Even and Odd numbers due to the exhaustive testing by the user included in test suite. So, the modified source code is as follows :-

```
#include <stdio.h>
int main()
{       intn,j;
        printf("Numbers between 1 to 100\n");
        for(n= 1;n<= 100;n++)
        {       for(j=2;j<= 100;j++)
                {       if(n%j==0)
                        break;
                }
                if(j==n)
                {       printf("Prime Number :- %d \n",n);
                }
                else if(n%2==0)
                {       printf("Even Number :- %d \n",n);
                }
                else if(n%2==1)
                {       printf("Odd Number :- %d \n",n);
                }
        }
        return 0;
}
```

2.2 Test Cases

In this technique, test cases generated according to requirement given i.e., test suite is provided and we have to extract the test cases according to the need of the user given in Table I

TABLE I
TEST CASES FOR PRIME, EVEN OR ODD

Test Cases	Data(Given)	Output Expected
tc_1	100	Even number
tc_2	23	Prime number
tc_3	27	Odd number
tc_4	89	Prime number
tc_5	11	Prime number
tc_6	46	Even number
tc_7	94	Even number
tc_8	94	Even number
tc_9	77	Odd number
tc_{10}	79	Prime number
tc_{11}	56	Even number
tc_{12}	68	Even number
tc_{13}	91	Odd number
tc_{14}	52	Even number
tc_{15}	47	Prime number

tc_{16}	98	Even number
tc_{17}	43	Prime number
tc_{18}	73	Prime number
tc_{19}	37	Prime number
tc_{20}	39	Odd number
tc_{21}	73	Prime number
tc_{22}	53	Prime number
tc_{23}	91	Odd number
tc_{24}	95	Odd number
tc_{25}	66	Even number
tc_{26}	55	Odd number
tc_{27}	59	Prime number
tc_{28}	13	Prime number
tc_{29}	31	Prime number
tc_{30}	50	Even number

Initially we resume that Delegate Set is empty. Frequency of the test case is calculated and on the basis of number of elements occurred in the test conditions.

2.3 **Statements Covered and Associated Tests**

Now, from the source code, statement to be covered decided by the user. In this all the statement not included. Only the condition part in the source code is included in the requirement. In this Test Condition is also defined and the test case associated with the different Test Conditions given in Table II

TABLE II
REQUIREMENTS AND ASSOCIATED TEST CASES

Requirement (Req_f)	**Statement Covered**	**Test Condition (TC_v)**	**Test Associated (tc_m)**
Req_1	for(n= 1;n<= 100;n++)	TC_1	tc_1-tc_{30}
Req_2	for(j=2;j<= 100;j++)	TC_2	tc_1-tc_{30}
Req_3	if(n%j==0)	TC_3	tc_1-tc_{30}
Req_4	if(j==n)	TC_4	tc_1-tc_{30}
Req_5	printf("Prime Number :- %d \n",n);	TC_5	tc_2,tc_4,tc_5,tc_{10},tc_{15}, tc_{17},tc_{18},tc_{19},tc_{21},tc_{22}, tc_{27},tc_{28},tc_{29}
Req_6	else if(n%2==0)	TC_6	tc_1-tc_{30}
Req_7	printf("Even Number :- %d \n",n);	TC_7	tc_1,tc_6,tc_7,tc_8, tc_{11},tc_{12},tc_{14},tc_{16},tc_{25},tc_{30}
Req_8	else if(n%2==1)	TC_8	tc_1-tc_{30}
Req_9	printf("Odd Number :- %d \n",n);	TC_9	tc_3,tc_9,tc_{13}, tc_{20},,tc_{23},tc_{24}, tc_{18},tc_{26}

2.4 Frequency Calculation

Frequency is calculated by counting the associate test case occurred in particular test condition. Then, test case having highest frequency is selected. If the frequencies are same then, choose it randomly shown in Table III

TABLE III
FREQUENCY CALCULATION

Associated Test Cases (tc_m)	**Test Condition (TS_v)**	**Frequency**
tc_1	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7
tc_2	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_3	TC_1,TC_2,TC_3, TC_4,TC_6,TC_8, TC_9	7
tc_4	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_5	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_6	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7
tc_7	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7
tc_8	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7
tc_9	TC_1,TC_2,TC_3, TC_4,TC_6,TC_8, TC_9	7
tc_{10}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{11}	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7
tc_{12}	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7
tc_{13}	TC_1,TC_2,TC_3, TC_4,TC_6,TC_8, TC_9	7
tc_{14}	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7
tc_{15}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{16}	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7
tc_{17}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{18}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{19}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{20}	TC_1,TC_2,TC_3, TC_4,TC_6,TC_8, TC_9	7
tc_{21}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{22}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{23}	TC_1,TC_2,TC_3, TC_4,TC_6,TC_8, TC_9	7
tc_{24}	TC_1,TC_2,TC_3, TC_4,TC_6,TC_8, TC_9	7
tc_{25}	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7
tc_{26}	TC_1,TC_2,TC_3, TC_4,TC_6,TC_8, TC_9	7
tc_{27}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{28}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{29}	TC_1,TC_2,TC_3, TC_4,TC_5,TC_6, TC_8	7
tc_{30}	TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8	7

Now, take randomly any test case, as we have now same Frequency of all test cases.

So, let us take tc_6 in the Delegate Set, the corresponding Test Condition satisfied are:-TC_1,TC_2,TC_3, TC_4,TC_6, TC_7, TC_8. So, we discard all the Test Conditions.

Now, check whether tc_6, fulfil our requirement or not, as tc_6 belongs to other set not the requirement needed. It does not belong to satisfied condition.

New Frequency, now been calculated:-

$tc_2 – tc_5 = 7 – 6 = 1$
$tc_9 – tc_{10} = 7 – 6 = 1$
$tc_{13} = 7 – 6 = 1$
$tc_{15} = 7 – 6 = 1$
$tc_{17} – tc_{24} = 7 – 6 = 1$
$tc_{26} – tc_{29} = 7 – 6 = 1$

Now, again the frequency are same, So, we take any test case randomly.
tc_2 has been taken in Delegate Set
DS – $\{tc_2, tc_6\}$

Corresponding elements of tc_2 is TC_5

Now, tc_2fulfill our requirement and the Test Condition associated with it, test cases in the Delegate Set other than tc_2 will be discarded. As. rest of test cases we not needed.

Now,

$tc_2 \in TC_5$

Now, extract all the elements belong to TC_5
$TC_5 \rightarrow \{tc_{2,}tc_4,tc_5,tc_{10},tc_{15},tc_{17},tc_{18},tc_{19},tc_{21},tc_{22}, tc_{27},tc_{28},tc_{29}\}$

All the elements, belongs to TC_5 are useful to us as per the requirement. So, this technique helps us to extract the useful elements and another advantage that it solves a problem of a program taking dynamic input.

V. Conclusion And Future Scope

Finally, we conclude that the Proposed algorithm is better in aspects of Existing algorithm. As, Existing one has the limitation of taking input at the run time of the program and fails over there.

The Proposed algorithm TestExude works on the limitations of previous algorithm and gives the better result compare. In addition, this work extract the special test cases, which is required for testing.

It takes little more space, but at the cost of little space it drastically reduce the test cases and time of the user to check manually even after extracting overall test cases.

Test Suites size, user time, organization cost, storage cost all are reduce efficiently. Ultimately, it improves and generate effective test case.

It shows the better result due to the Delegate Sets. In Delegate Sets, discard criteria is applied. No need to keep those test cases which is not useful to us. So, this effective technique prove to be superior.

It prove to be good technique in current scenario. As, it reduces the test case drastically. In future, lot of work has to be done on this technique. In future, make this technique fully automated and compatible with other tools. It has certain limitations i.e., in maximum program frequency of the requirements is same, so we try to find other parameters to overcome this.

Acknowledgements

This technique has been made by the hard work and support of my guide Mr. Thirunavukkarasu K., Assistant Professor, Galgotias University, Greater Noida, Uttar Pradesh I would like to thanks my friends to give me unconditional support.

References

[1] Antonia Bertolino, Software Testing Research: Achievements, Challenges, Dreams Future of Software Engineering, IEEE, 2007, 0-7695-2829-5/07.
[2] Juha Itkonen, Nika V. Mantyla and Casper Lassenius, The Role of the Tester's Knowledge in Exploratory Software Testing, IEEE Computer Society, vol. 39, no. 5, 2013, 707-724.
[3] (2014) The Wikipedia website. [Online]. Available: http://en.wikipedia.org/wiki/Test_case.
[4] (2014) The Wikipedia website. [Online]. Available: http://en.wikipedia.org/wiki/Test_suite.
[5] (2014) The Wikipedia website. [Online]. Available:http://en.wikipedia.org/wiki/Knowledge.
[6] Saif-ur-Rehman Khan, Aamer Nadeem, TestFilter: A Statement-Coverage Based Test Case Reduction Technique, IEEE, 2006.
[7] S. McMaster, A.M. Memon, Call Stack Coverage for Test Suite Reduction, in Proceedings of the 21st IEEE International Conference on Software Maintencance (ICSM'05), Budapest, pp. 539-548, Sept 2005.
[8] A. Jefferson Offutt, Zhenyi Jin and Jie Pan, The Dynamic Domain Reduction Procedure for Test Data Generation, Software Practice and Experience, vol 29, no 2, 1999, pp. 167-193.

[9] A. Jefferson Offutt, Zhenyi Jin and Jie Pan, The Dynamic Domain Reduction Procedure for Test Data Generation: Design and Algorithms*, ISSE Tech. Rep. ISSE-TR-94-110, George Mason University, Washington D.C., USA, 1994.
[10] Dr. R.P. Mahapatra, M. Mohan and A. Kulothungan, Effective Tool for Test Case Execution Time Reduction, In IACSIT, 2011, International Symposium on Computing, Communication and Control (CSIT), Singapore.
[11] R.P. Mahapatra and Jitendra Singh, Improving the Effectiveness of Software Testing through Test Case Reduction, In World Academy of Science, Engineering and Technology, 2008.
[12] Quingtan Wang, Shujuan Jiang and Yanmein Zhang, An Approach to Generate Basis Path for Programs with Exception-Handling Constructs, In IACSIT Press, 2012, International Conference on Computer Science and Information Technology (ICCSIT), Singapore.