Calculating Testcases for Data Structures using white box testing

D.Meenakshi , K.PradeepKumar

***Abstract*—Software testing is an activity which is done at the ending stage of software development life cycle with in a limited period of time . In particular, most developers tend to test their programs manually and automatically. In this paper, basic data structures are utilized to emphasize the significance of writing efficient test cases by testing their essential properties. The paper also includes white box testing at the unit level, to find the testcases of a program. This approach accomplishes two important objectives: (1) to find number of testcases and how to find hidden bugs and defects in their programs and (2) it illustrates them to test more efficiently by leveraging data structures that are already familiar to them.**

***Keywords*—Software Testing strategies; Data Structures; Unit Testing; white box Testing; Stacks;Cyclomatic complexity**

INTRODUCTION

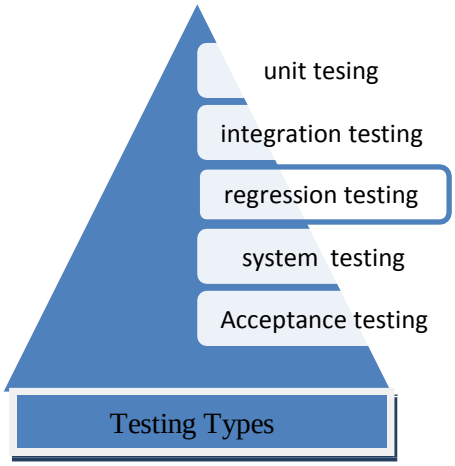
In general, software testing is an important area in the software development life cycle. It is used to find bugs present in the programs. Software testing is not only error detection; Testing software also means operating the programs under controlled conditions, to check whether it behaves "as specified" by the user. Testing contains both verification and validation ,verification means whether we are getting the output or not where as validation means getting the output according to the user requirements.

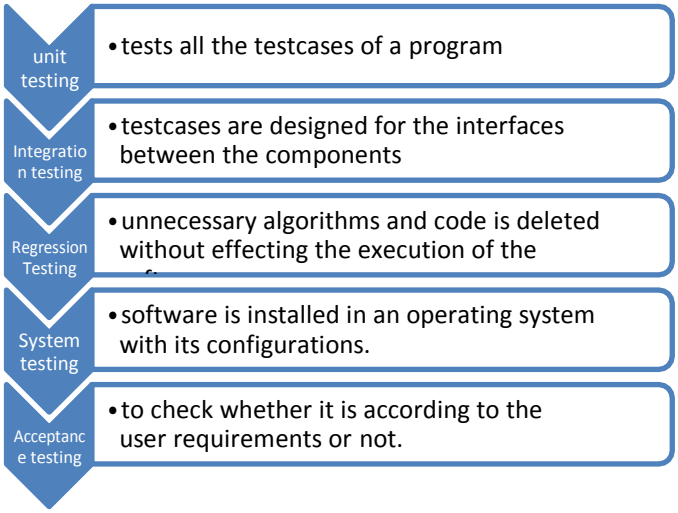
The different types of testing techniques are as follows:

unit testing contains white box testing and black box testing.white box testing tests all the testcases of a program where as the blackbox testing tests whether we get the output.

or not. Integration testing is done to combine all the components and the testcases are designed for the interfaces between the components.

Regression testing is done when theunnecessary algorithms and code is deleted without effecting the execution of the software. System testing is done when the software is installed in an operating system with itsconfigurations. Acceptance testing is done by the user to check whether it is according to the user requirements or not.





White box Testing:

The number of testcases are calculated using cyclomatic complexity of a program .cyclomatic complexity is calculated in three ways. They are:

1. Number of predicate nodes+1
2. Number of edges- Number of nodes+1
3. Number of Regions+1

Data Structures [8] is an important course existing in Computer Science, Computer Engineering and Software

***Manuscript received Dec, 2019*.**

***D.Meenakshi**, Lecturer in Computer Science, GDC Tiruvuru,Krishna University, Krishna, India, 9550078441,dmeenu1986@gamil.com.*

***K.Pradeep Kumar**, JKC Mentor,GDC Tiruvuru, Krishna University,, Krishna, India, 9550078441,pradeep.vspt@gamil.com.*

Engineering undergraduate and postgraduate programs. Though , software testing is always a required course. The moto behind testing may be simple to most of the users. But

,most commonly, users test their programs manually by giving the input and they will produce the correct results as output[1235] and after getting the output the users wont go for the other testcases and they do not find the bugs present in the program. This process is called as confirmation bias[2].as the users are short in time they wont find time to check all the testcases present in the program and to optimize the program.

The main goal of this paper is to share a reliable teaching approach which will enable users to put pen to paper the automated tests by taking into account the fundamental properties and constraints of a problem. It introduces a direct approach to unit testing by using common data structures that are often used in coding and software development. By using data structures, along with distinguished problems that are introduced earlier in the curriculum or in a prerequisite course, users can flawlessly learn the ideology and application of software testing without the added burden of learning new unfamiliar content. The rest of the paper is organized as follows. Section 2 presents a fundamental overview of Stacks . Section 3 explains how to test the fundamental properties and implementation of a stacks. Section 4 presents the calculation of number of testcases using cyclomatic complexity to illustrate white box testing at the unit level. Section 5 concludes the paper.

USING DATA STRUCTURES FOR SOFTWARE TESTING

As stated earlier, software testing is not always a requisite course in most under graduate degree programs. However, it is a key aspect of software development and is typically introduced briefly in the later stages of most Software Engineering courses.

users become plagued with the software testing tools they want to learn to test software using automated testing. They often struggle with the concept of testing to find errors rather than just testing to show that their software is operating with a given set of inputs and giving the correct output results. To solve this issue, a variety of software testing problems are given to users, and it becomes immediately apparent that they do not quite understand the fundamental properties to find bugs. A natural approach is to utilize Abstract Data Types (ADT) to teach them this type of testing [8].

Abstract Data Types [8] are taught in Data Structures, and most users learn about ADTs to aid and develop their programming skill set and knowledge. It, therefore, makes perfect sense to utilize ADTs in teaching software testing, because doing so provides continuity and allows users to concentrate more on learning and applying testing principles.

Stacks work on the principle of last-in-first-out (LIFO) data structure. In a stack, the element will be inserted in the last and removed first. Similarly, stacks are implemented using arrays and linked lists. The number of testcases are calculated for the stack operations during its implementation in section 3.

THE SOFTWARE TESTING STRATEGIC APPROACH

White box testing is first introduced to testing at the unit level [7]. These tools allow them to develop automated test methods and test classes [6, 8]. Unit testing is a software testing process in which the smallest testable parts of a program are individually and independently analyzed for proper operation. Unit testing focuses more on finding bugs in objects, functions and classes. In particular, how to test Stacks and its operations when implemented using arrays to ensure that their fundamental properties are not violated. They are also introduced to performance testing at the unit level.

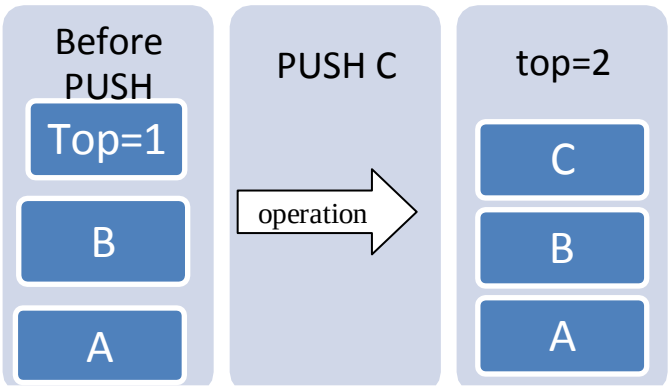


Fig. 1. Example of stack push operations

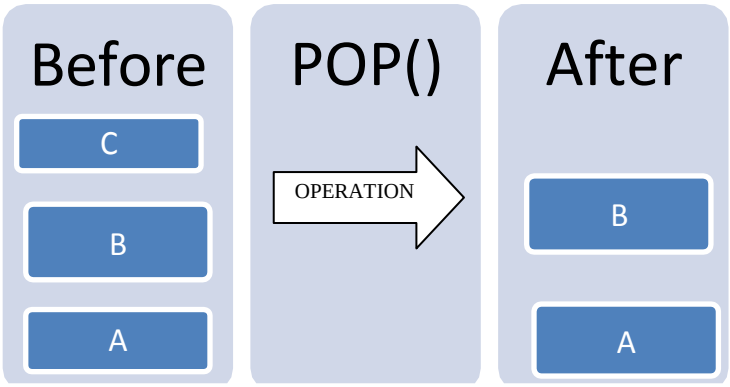


Fig 2: Example of stack pop operations

A. *Stacks*

operations consists of initializing the stack, using it, and then de-initializing it. A stack has two basic operations: (a) **push()** – inserting (storing) an element on the stack; and (b) **pop()** – deleting (accessing) an element from the stack. Additionally, other supporting operations that must be defined to efficiently use a stack are:

- **peek()** − getting the top data element of the stack, without removing it.
- **isFull()** – to check whether the stack is full.
- **isEmpty()** – to check whether the stack is empty.

Fig. 1 shows the basic push operation behind a stack. A new element is always added at the top of the stack using the push() operation.Fig 2 shos the basic pop operation behind a stack. The element at the top of the stack is always removed with the pop() operation.

B. The Stack Test

Users are asked to create a test that will effectively test the properties of a stack. This is simple to test; it involves adding a bunch of elements on a stack, and ensuring that they are removed in the correct order.

For example, if A and B are pushed unto a stack one at a time, and if the stack is popped (the element at the top of the stack, is removed first, one at a time) until it is empty, then this means the stack is adhering to its fundamental LIFO property..

In this example A was inserted on the stack first; this means that A will be the last element to be popped from the stack. Similarly, B was the second element to be inserted on the stack. Therefore, B must be the first element to be popped from the stack. Thus, the first pop operation should remove an element with the value B. In other words, the sequence and value of elements added must stick on to the LIFO constraint. In the example given, notice that each element holds a unique value to better illustrate the basic dynamics of this test. If the first pop operation deleted an element with a different value, then clearly the stack is not adhering to its fundamental LIFO constraint.

C. Implementation and Constraints on stack

When a stack is implemented using arrays in c language ,as the array is having a fixed size with 'n' elements then the constraints for push operation are:

1) if top=0 means the stack is empty and the insertion can be done by incrementing the top value.

2) if top=n means the stack is full or stack overflow and the insertion cannot be done into the stack. If the stack is full then the insertions cannot be done into the stack.

```
void push()
{
  if(top>=n-1)
  {
    printf("\n\tSTACK is over flow");
  }
  else
  {
    printf(" Enter a value to be inserted:");
    scanf("%d",&x);
    top++; stack[top]=x;
  }
```

The constraints on pop operation are:

```
If(top=-1)
{
printf("\n\t Stack is under flow");
 }
 else
 {
    printf("\n\t The popped elements is
    %d",stack[top]); top--;
 }
}
```

CALCULATING THE TESTCASES USING WHITE BOX TESTING:

In white box tsting testing [4], sometimes the performance of a given method or class is tested to resolve its efficiency in problem solving. Exhaustive testing is expensive (and time consuming). Therefore, evaluating the effectiveness of a solution can be used as a performance test at the unit level. Recursion is a topic that is covered in Data Structures. Essentially, recursion is used where a large problem can be splitted into smaller repetitive ―sub-problems. A recursive method will call itself to perform those sub-problems, and eventually the method will come across a sub-problem so trivial, that it can handle it without recalling itself. This is known as a basis path testing, and it is required to prevent the method from calling itself repeatedly without ever stopping.

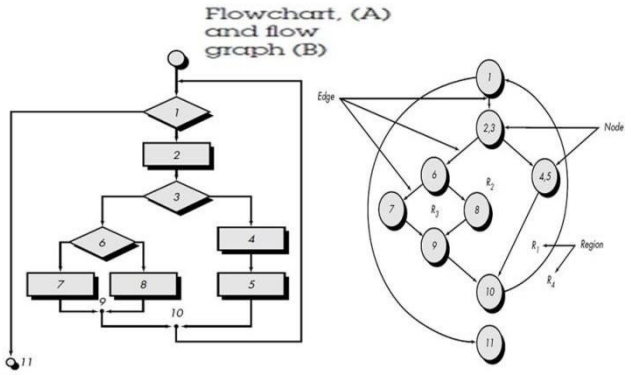
Whitebox testing is used test each and every statement present in the program.it is also called as glass box testing or transparent testing the number of testcases are calculated by using flowgraph of the program.for example if we consider a program for biggest of three numbers.

```
1.if (A >= B) {
2.        if (A >= C)
            printf("%d is the largest
number.", A);
3.        else
            printf("%d is the largest
number.", C);
4.    }
5.    else {
```

```
6.        if (B >= C)
7.            printf("%d is the
largest number.", B);
8.        else
9.            printf("%d is the
largest number.", C);
10.    }

   }
```

The flow graph for this program is:



By using data structures, along with well-known problems that were introduced to users earlier or in a prerequisite course, they can flawlessly learn the principles and application of software testing without focusing on learning new unfamiliar content. Furthermore, users often utilize the same data structures to implement software programs in other upper level courses and internship projects. Therefore, software testing with ADTs, provides users with a second opportunity to master their skills and knowledge in software development and Cyclomatic complexity for this graph is:

1.Number of edges-no of vertices+2=11-9+2=4

2.Number of predicate nodes+1=3+1=4

A predicate node is a conditional node present in the program

3. Number of regions present in the flow graph=4[Here External region is also considered as one region]

Like this for stack data structure for push operation the number of testcases will be 2 and for pop operation the number of testcases will be 2.if we know the number of testcases for a program we can check all the errors present in the program.

FUTURE WORK AND CONCLUSION

Future work entails identifying, and developing, other relatable examples that can be used to teach software testing at other testing levels-like integration, and system testing levels. Additionally, finding techniques and relatable exercises that help users understand code coverage in terms of data path, and input partition coverage, are also important.

Software testing is a very important activity that requires more relatable strategies to help users learn how to effectively test their programs. Testing does not get enough attention in the SDLC and so, naturally, users do not spend enough time to fully understand the problems they are solving at a fundamental level. As a result, this abandon propagates into how they test their code.

Using the above examples in Sections 3, it was demonstrated that effective testing can be achieved by utilizing some of the basic topics covered in a distinctive Data Structures course. This approach focuses on understanding constraints and the fundamental properties associated with solving a particular problem. The aim is to encourage users to invest the minimum time to fully understand a problem in order to create test cases that will effectively find bugs and defects, which are the primary goals of software testing. Additionally, we extended the scope of unit testing to include performance testing of a recursive method, which was applied to other programs.

References:

[1] K. Muşlu, B. Soran, and J. Wuttke, ―Finding bugs by isolating unit tests‖, In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering

[2] G. Calikli, B. Arslan, A. Bener, ―Confirmation bias in software development and testing: An analysis of the effects of company size, experience and reasoning skills‖, In Proceedings of the 22nd annual psychology of programming interest group workshop, 2010.

[3] K. Buffardi, S. H. Edwards, ―Exploring influences on student adherence to test-driven development‖, In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*

[4] D.Meenakshi –"Software testing techniques in software development life cycle" International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 3729-3731,ISSN:0975-9646.

[5] G. Calikli, A. Bener, ―Empirical analyses factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance. In Proceedings of 5th international workshop on predictor models in software engineering, 2010.

[6] Y. Langsam , M. Augenstein, A. M.Tenenbaum, "Data Structures using Java", Pearson Prentice Hall, ISBN: 0-13-047721-4.

[7] A. Hunt, D. Thomas, ―The Pragmatic Programmer From Journeyman to Master‖, Addison-Wesley, ISBN: 978-0-2016-1622-4

[8] . D. McGregor, D. A. Sykes. ―*A Practical Guide to Testing Object-Oriented Software"*, Addison-Wesley Longman Publishing Co., Inc., 2001, Boston, MA, USA.

D.Meenakshi,MTech, working as lecturer in computer science and interested in software engineering research work and published journals in IJARCET previous volume and participated in IEEE conference at coimbatore .

K.PradeepKumar,MTech,Working as JKC Mentor and interested in DataMining research work and published journals in DBMS in a conference conducted by AndhraLayola college of engineering and technology.