International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 5, Issue 2, March - April 2016 **ISSN 2278-6856**

A customized Approach For Automated Test Case Generation and Optimization for system based software Testing

Krishnachandra.M

Sr. Assistant Professor, Nadeem Akram.N. Department of MCA

New Horizon college of Engineering . Bangalore-560103

Abstract

Testing is a process of verifying the correctness of a software which provides a way where the functioning of the system is understood with well defined constraints. It assures the quality of software to be developed in terms in presence of errors. Its a tedious process where in there is a lot of manual activities which consume long duration if not formulated in a productive and efficient manner. So that the tester will go automation testing ,even though they use automation some techniques will not perform the expected functionality. Software Testing is the process intended to discover error and omitting the bugs at the time of executing a program .It assists in determining the possible bugs that can make a program to malfunction ,sometimes it can be used for a periodical inspection to bring the quality ,reliability performance and efficiency of the system. This paper discuss the new strategy to automate the test case generation using the backtracking algorithm , optimization using orthogonal array approach and effective test case delivering using Nqueens problem

Keywords:-Automated Test case generation, Back tracking Algorithms, orthogonal array, optimization N queens problem

1. Introduction

So analysing all here proposing this concepts of back track algorithm to design a Automated test case .Now this method proposes how to generate test cases using back tracking algorithm instead of UML activity diagram. We first construct the diagram for given problem using back-tracking algorithm and then randomly generate test cases, for a program under testing.

Testing can be done at different stages during Various phases of software developments .with respect to their functionalities. Testing can be categorized into two types such as White Box testing, Black Box testing. White box testing can be used through detail analysis of program structure

Whereas black box methodology deals with specification and design document without the working of internal details..

1.1 Traditional Testing Approach

Usually a software consists of system that is further decomposed in to sub systems and each subsystem is developed along with all the constrains when a subsystem is developed we must verify the behaviour of the same .often a single test case is employed to verify the working of entire system which inefficient there fore the system will not produce expected outputs now the trend in testing domain is to break the system in to subsystems and understand its behaviour through test cases**,** This method will significantly increase the software reusability, extendibility, inter-operability, etc.

1.2 Draw-backs of Existing testing Approach.

The existing system testing approach has a number of drawbacks which is inefficient in performing a quick effective testing although OOAD is an emerging concept throughout the globe it has found widespread acceptance in the industry as well as in academics. The Problems with general approach is as follows

- requires lot of human resources
- single operation test case
- lack of system understand ability

It acts as a middleware between the developers and the end users UML plays an important role in the cost-efficient and effective analysis of the system. which is very much crucial in IT industry. UML is the modelling language, which supports object-oriented features at the core. UML accomplish the visualization of software at early stage of SDLC, which helps in many ways like confidence of both developer and the end user in understanding the system. earlier error detection can be done through proper analysis of design and etc. UML also helps in making the proper documentation of the software. Models are the intermediate artifacts between requirement specification and final code. Models preserve the essential information from the requirement, and are the basis for implementation.

1.3 Automation of test case

Earlier stages in software industry they were used manual testing later on they found its consuming more time and resources to over coming these issues switched to automated testing ,for that various drivers and tools exist in the market like selenium ,QTP,WIN RUNNER,LOAD RUNNER but it can test synarios not for test cases But there are several research reports on automatic test case generation from legal models [3].

UML provides a number of diagrams to describe particular aspects of software Artifacts. These diagrams can be classified depending on whether they are intended to describe functional, structural or behavioural implementation aspects of systems.

1.5 Automatic Test Case generation

Usually a set of test cases is required to test the system behaviour[5]. It serves a basis of test where the feasibility of the system can be measured depending on the level of test cases. A typical test case should have two components such as:

- An input data to the program.
- Description of correct output from the set of input data.

Generation of test cases depends on the type of the system and algorithm to implement them. Many researchers have focused on automation of this task and their reported results show with varying degrees of success.

An **Automatic test case generator** is a tool that has a predefined set of test classes and would put the system under test for the predefined set of test cases, also we can tailor them for specific needs

Automatic test case specifications is based on certain pre specified testing criteria called as test coverage criteria. Subsequently, the exact test data for each test specification is determined to form the test cases by generating test case. A set of input data provides an effective means to test system robustness. Test cases are from specifications, design artifacts, or the source code. Test cases are commonly designed based on program source code. This makes test case generation difficult especially for testing at cluster levels.

Manual generation of test case is
time consuming and tedious task. Hence either automatic or semi-automatic generation of
test case from design document is often in quest in the software community.

2.Methodology Employed

Basics of Backtracking Algorithm (BA)

A backtracking is a general approach to solve a given complex problem with the defined set of constraints ,it follows an elegant approach were it generates the solution incrementally the solution is put on hold until all the increments have been independently.

Using backtracking approach we can automate the process of test case generation where upon indentifying the given input and output behaviour and keeping in consideration of system constraints we can design the auto-test case behaviour of a given system.

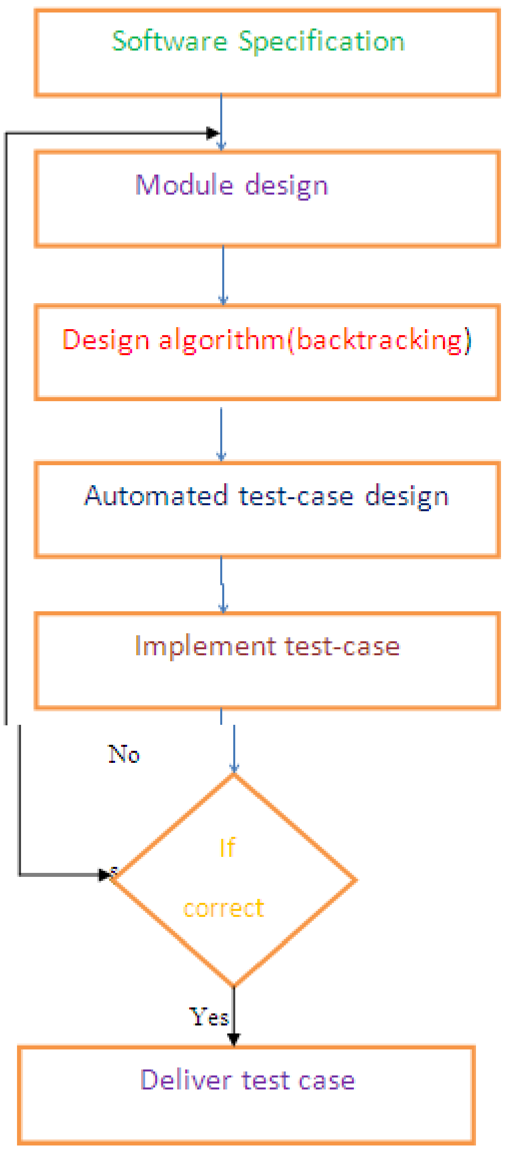
Various methods have been used for finding the complex problems to get a better result but here backtracking method will generate best results comparing other.

Generally a Response of the backtracking algorithm is fast and quick

Using Back tracking algorithm how the test case control flow is working is explained

Flow diagram-1

For Test case generation



2.1 Basics of Backtracking Algorithm (BA)

*A Backtracking algorithm is a searching technique used in computing exact or approximate solutions to optimization and search problems from various domains,

```
Algorithm: Backtrack(s)
Input: software specification with detailed description
Output: set of automated test cases
Begin
Loop :T=size(s)
Divide  T into K instances of same size Each
When T(k) T(k+1) are same size
Determinate nature of T(k);
Based on t(k) description derive the test case
Test t(k)
If t(k) acceptable
Deliver t(k) and move to t(k+1)
Loop the above process until t(s)=0
```

3 .Need for optimization

In software development process testing plays vital role to delivery of product, it s a last stage of the process before the product released to the stake holders. so every day a huge competition is raised in the market to bring quality of the software in economical manner . because of it testers will generate effective test cases on that way testers spend more time generate test cases and more over they are not getting time to do analyse and optimize the test cases. Even Various approaches are followed to do test cases optimization[2], to bring the quality soft ware but this approach will generate a feasible one.

3.1.Optimization of test cases

To bring the efficient test case generation testers need to follows the test case optimization .for that which method is feasible in terms of technical ,economical and operational should be discussed and adopted in further process. Various researchers are following genetic algorithmic approach its not giving exact or accurate result, so the proposed method has been taken which is orthogonal array approach to be suggested [1] to bring the effective optimization.

3.2.Orthogonal Array Approach

The Orthogonal Array Testing (OATA) approach is a systematic Approach which is used for test case optimization from various test cases[1]. This method provides representative coverage of entire pair of variable combinations. The Orthogonal Array Testing Approach-(OATA) is particularly useful for unit testing of software components. It provides a combinations of options which can be used to optimize existing test case to improve system productivity.

Here in some terminology for working with orthogonal arrays followed by an example array

- Horizontal: the number of rows in the array. This directly translates to the number of test cases that will be generated by the OATA technique.
- vertical: the number of columns in an array. This directly translates to the maximum number of variables that can be handled by this array.
- stages : the maximum number of values that can be taken on by any single verticle. An orthogonal array will contain values from 0 to stages-1.
- PASS: the number of columns it takes to see each of the Levels Strength possibilities equally often.
- Orthogonal arrays are most often named following the pattern S(Stage Factors).

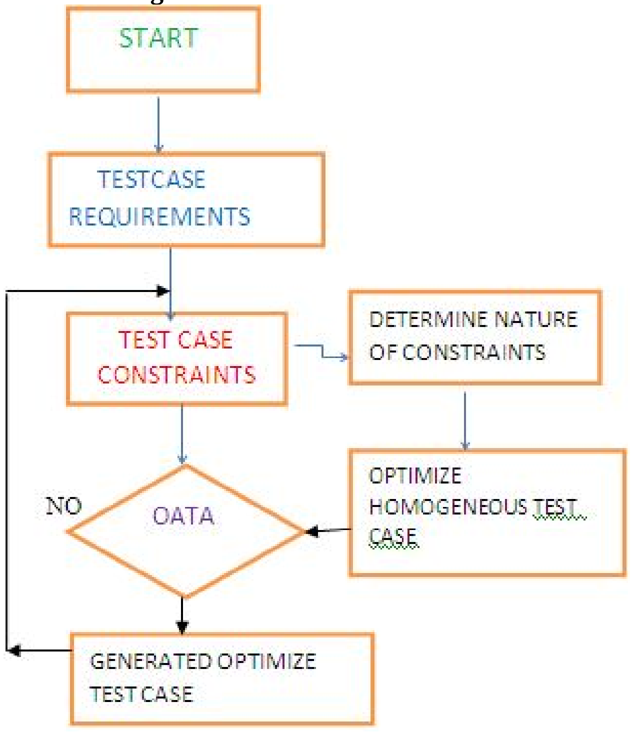
3.3. How to use this technique

The OAT technique is simple and straightforward. The steps are outlined below.

1. Decide how many independent variables will be tested for interaction. This will map to the vertical of the array.

- Decide the maximum number of values that each independent variable will take on. This will map to the stages of the array.
- Find a suitable orthogonal array with the smallest number of horizontal. A suitable array is one that has at least as many vertical as needed from Step 1 and has at least as many stages for each of those factors as decided in Step 2.
- Map the vertical values onto the array.
- Choose values for any "left over" Stages.
- Transcribe the into test cases, adding any particularly suspicious combinations that aren't generated

3.3 Flow diagram-2 For Test case OPTIMIZATION


START
TESTCASE REQUIREMENTS
TEST CASE CONSTRAINTS
DETERMINE NATURE OF CONSTRAINTS
OPTIMIZE HOMOGENEOUS TEST CASE
NO
OATA
GENERATED OPTIMIZE TEST CASE

PRE TEST

ROWS	1	2	3	4
1	1			
2	1			
3	1			
4	2			
5	2			
6	2			
7	3			
8	3			
9	3			

Ref[1]

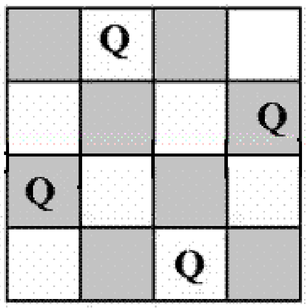
POST TEST

ROWS	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2			
5	2			
6	2			
7	3			
8	3			
9	3			

Ref[1]

4. Implementing optimized test case using N Queens problem approach

The n-queens problem consists in placing n non-attacking queens on an n-by-n chess board. A queen can attack another queen vertically, horizontally, or diagonally. E.g. placing a queen on a central square of the board blocks the row and column where it is placed, as well as the two diagonals (rising and falling) at whose intersection the queen was placed.



Based on the ref[4] an idea was generated to design this method.
Working of n-queens problem

4.1. QApproach

- Create a solution matrix of the same structure as chess board.
- Whenever place a queen in the chess board, mark that particular cell in solution matrix.
- At the end print the solution matrix, the marked cells will show the positions of the queens in the chess board.

4.2.Algorithm

1. Place the queens column wise, start from the left most column
2. If all queens are placed.
1. return true and print the solution matrix.
3. Else
1. Try all the rows in the current column.
2. Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
3. If placing the queen in above step leads to the solution return true.
4. If placing the queen in above step does not lead to the solution , BACKTRACK, mark the current cell in solution matrix as 0 and return false.
4. If all the rows are tried and nothing worked, return false and print NO SOLUTION.

4.3.Implementing optimized test cases uses of n queens problem

Consider a system with 16 modules where each module would require a test case each to verify its correctness.
Using queens problem we can reduce a number of test cases from 16 to 4 ,yet provide the same functionality .
Working of test cases is as follows.

1. consider all the 16 modules to be placed on a plane
2. According to the constraints of n queens problem a queen should not meet any other queen either vertically or horizontally or diagonally.
3. considering each test case as a queen and using the above rule no-test case must match any other test cases either vertically or horizontally or diagonally placed modules .
4. Now Each test case can be used to verify the correctness of the modules adjacent to it either in row or columns.
5. These way a single test case can be used to verify correctness of 7 modules parallel
6. this method can be used to perform unit testing in all the modules using fewer test case.
7. every test case is provided with the functionality to test corresponding adjacent modules

5. Conclusion

The above method can be used to perform unit testing for modules 2 ^n which requires n test cases only .There are three various techniques are used to generate ,optimize and implement and produce the effective test cases in the systems. An above approaches can be employed in distributed and as well as standalone system.

References

[1]. Shubhra Banerji Project Manager, Software Testing Research & Development Unit, IBM India Pvt. Ltd., Bangalore, India

[2]. Abhishek singhal,swathi chandana,abhai banshal,Aminity school of engineeringand technology,Amnity university,Noida. ‘Optimization of test cases Using genetic Algorithm’- International g technology and advancedEngineering,vloume2,issue3, march2012,Website:www.ijetae.com(ISSN 2250-2459)

[3]. Mr.sridhar yemul ,prof.kapil vhatkar, prof.vipulbag. “Testing approach for automatic test generation and optimization using GA” IJETTCS- Volume-3,issue-5,-october-2014(issn 2278-6856).

[4]. K.Devika Rani Dhivya,C.Sunitha,Department of computer Applications,Sri krishnaArts and Science college india-“A review on optimization used for randomized unit testing”-www.ijarcsse.com-Volume 4,issue6,june2014(1ssn:2277128x)

[5]. Sapnavarshney,monica mehrotraDepartment of Computer Science,jamia millia Islamia,India- “Automated Software Test Data Generation for Data Flow Dependencies using Genetic Algorithm” www.ijarcsse.com, Volume 4, Issue 2, February 2014 ISSN: 2277 128X