An Efficient Software Testing by Diminishing Number of Test Executions

Dr. Pradeep Udupa
MES College of Engineering, Kuttippuram, Thrissur, Kerala 679573 India
pradeepumlr@gmail.com

Abstract -- Software Testing is the process of executing a program or system with the intent of finding errors or it involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Software is not unlike other physical processes where inputs are received and outputs are produced.

In this study, execution code & algorithm is developed to optimize the testing efficiency by fetching test inputs from the database which will reduce time, effort & number of executions. Here an efficient code is developed to fetch test data from database and to fetch the data from data table to increase the execution speed, decrease the effort and increase testing efficiency.

Keywords: Software Testing, Efficient Codes, Testing Efficiency

I. INTRODUCTION

SOFTWARE Testing is the process of executing a program or system with the intent of finding errors. The main objective is:

• To reduce total numbers of test executions. Generally, the larger the input domain, the more exhaustive the testing would be. To avoid this problem, a minimum set of test cases needs to be created using an algorithm to select a subset that represents the entire input domain. In addition, every time feeding data to .exe file the testing itself would take longer to run, particularly for regression testing where every change in the program demands repeat testing. Therefore, reducing number of the test cases does have advantage in efficiency.

Automating execution by providing all test case data will decrease no of execution and increase the speed.

• One can develop execution code for fetching the data from data Table that is used to retrieve test case data directly from Data table without expecting data to be inputted. By implementing this procedure, it will automate the execution process and cost & effort involved in doing manual work will be diminished.

• Developing execution code to calculate overall time required to find an efficiency of execution in terms of speed.

II. THE BASICS

Software testing is a process of verifying and validating that a software application or program.

1. Meets the business and technical requirements that guided its design and development, and
2. Works as expected.

Software testing also identifies important defects, flaws, or errors in the application code that must be fixed. The modifier “important” in the previous sentence is, well, important because defects must be categorized by severity.

During test planning, we decide what an important defect is by reviewing the requirements and design documents with an eye towards answering the question “Important to whom?” Generally speaking, an important defect is one that from the customer’s perspective affects the usability or functionality of the application. Using colors for a traffic lighting scheme in a desktop dashboard may be a no-brainer during requirements definition and easily implemented during development but in fact may not be entirely workable if during testing, we discover that the primary business sponsor is color blind. Suddenly, it becomes an important defect (About 8% of men and 0.4% of women have some form of color blindness.)

The quality assurance aspect of software development—documenting the degree to which the developers followed corporate standard processes or best practices—is not addressed in this paper because assuring quality is not a responsibility of the testing team. The testing team cannot improve quality; they can only measure it, although it can be argued that doing things like designing tests before coding begins will improve quality because the coders can then use that information is used while thinking about their designs and during coding and debugging.

Software testing has three main purposes: verification, validation, and defect finding.

• The verification process confirms that the software meets its technical specifications. A “specification” is a description of a function in terms of a measurable output.
value given a specific input value under specific preconditions. A simple specification may be along the line of "a SQL query retrieving data for a single account against the multi-month account summary table must return these eight fields ordered by month within 3 seconds of submission."

- The **validation** process confirms that the software meets the business requirements. A simple example of a business requirement is "After choosing a branch office name, information about the branch's customer account managers will appear in a new window. The window will present manager identification and summary information about each manager's customer base."

A **defect** is a variance between the expected and actual result. The defect's ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

### III. WHY TEST SOFTWARE?

A "bug" is really a problem in the code; software testing is focused on finding defects in the final product. Here are some important defects that better testing would have found:

- In February 2003, the U.S. Treasury Department mailed 50,000 Social Security checks without a beneficiary name. A spokesperson said that the missing names were due to a software program maintenance error.
- In July 2001 a "serious flaw" was found in off-the-shelf software that had long been used in systems for tracking U.S. nuclear materials. The software had recently been donated to another country and scientists in that country discovered the problem and told U.S. officials about it.
- In October 1999, the $125 million NASA Mars Climate Orbiter—an interplanetary weather satellite—was lost in space due to a data conversion error. Investigators discovered that software on the spacecraft performed certain calculations in English units (yards) when it should have used metric units (meters).

In June 1996 the first flight of the European Space Agency’s Ariane 5 rocket failed shortly after launching, resulting in an uninsured loss of $500,000,000. The disaster was traced to the lack of exception handling for a floating-point error when a 64-bit integer was converted to a 16-bit signed integer.

### IV. WHAT DO WE TEST?

First, test what’s important. Focus on the core functionality—the parts that are critical or popular—before looking at the "nice to have" features. Concentrate on the application’s capabilities in common usage situations before going on to unlikely situations. For example, if the application retrieves data and performance are important, test reasonable queries with a normal load on the server before going on to unlikely ones at peak usage times. It’s worth saying again: focus on what’s important. Good business requirements will tell you what’s important.

The value of software testing is that it goes far beyond testing the underlying code. It also examines the functional behavior of the application. Behavior is a function of the code, but it doesn’t always follow that if the behavior is "bad" then the code is bad. It’s entirely possible that the code is solid but the requirements were inaccurately or incompletely collected and communicated. It’s entirely possible that the application can be doing exactly what we’re telling it to do but we’re not telling it to do the right thing.

A comprehensive testing regime examines all components associated with the application. Even more, testing provides an opportunity to validate and verify things like the assumptions that went into the requirements, the appropriateness of the systems that the application is to run on, and the manuals and documentation that accompany the application. More likely though, unless your organization does true "software engineering" (think of Lockheed-Martin, IBM, or SAS Institute) the focus will be on the functionality and reliability of application itself.

Testing can involve some or all of the following factors. The more, the better:

- Business requirements
- Functional design requirements
- Technical design requirements
- Regulatory requirements
- Programmer code
- Systems administration standards and restrictions
- Corporate standards
- Professional or trade association best practices
- Hardware configuration
- Cultural issues and language differences.

### V. WHO DOES THE TESTING?

Software testing is not a one person job. It takes a team, but the team may be larger or smaller depending on the size and complexity of the application being tested. The programmer(s) who wrote the application should have a reduced role in the testing if possible. The concern here is that they’re already so intimately involved with the product and “know” that it works that they may not be able to take an unbiased look at the results of their labors.

Testers must be cautious, curious, critical but non-judgmental, and good communicators. One part of their job is to ask questions that the developers might find not be able to ask themselves or are awkward, irritating, insulting or even threatening to the developers e.g.
• How well does it work?
• What does it mean to you that “it works”?
• How do you know it works? What evidence do you have?
• In what ways could it seem to work but still have something wrong?
• In what ways could it seem to not work but really be working?
• What might cause it to not work well?

A good developer does not necessarily make a good tester and vice versa, but testers and developers do share at least one major trait—they itch to get their hands on the keyboard. As laudable as this may be, being in a hurry to start can cause important design work to be glossed over and so special, subtle situations might be missed that would otherwise be identified in planning. Like code reviews, test design reviews are a good sanity check and well worth the time and effort.

Testers are the only IT people who will use the system as heavily an expert user on the business side. User testing almost invariably recruits too many novice business users because they’re available and the application must be usable by them. The problem is that novices don’t have the business experience that the expert users have and might not recognize that something is wrong. Testers from IT must find the defects that only the expert users will find because the experts may not report problems if they’ve learned that it’s not worth their time or trouble.

VI. NEED & SCOPE OF THE STUDY

1) To increase Testing Efficiency
2) Reduce No of Execution & Execution Time & Effort

Steps which are known to offer testers with diversity of testing methods and, hence, enhance probability of detection, are therefore recommended as the most efficient tools currently Available [4], [6].

1) Path testing; aims to inspect the validity of selected paths without the need for testing every possible path (as Required in Structural testing). The test is preferable when the Number of all available paths is so great that testing all of them become impractical [1].

2) Independent program paths: an independent program path is any path through the program that introduces at least one new set of processing statements or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.

For example:

```
1. Using the design or code, draw the corresponding flow
2. Determine the cyclomatic complexity of the flow graph
3. Determine a basis set if independent paths.
4. Prepare test cases that will force execution of each path in the basis test.

Example
1. Number of regions of flow graph
2. Edges—nodes+2
3. Predicate node+1.
```

Independent paths:
Path 1: 1-11
Path 2: 1-2-3-4-5-10-1-11
Path 3: 1-2-3-6-8-9-10-1-11
Path 4: 1-2-3-6-7-9-10-1-11

Note that each new path introduces a new edge. The path 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 is not considered to be an Independent path because it is simply a combination of already specified paths and does not traverse any new edges. Paths 1, 2, 3, and 4 constitute a basis set for the flow graph in the Figure.
That is, if tests can be designed to force execution of these paths (2, 4, 6, 7), every statement in the program is Guaranteed to be executed at least one time, and every condition will have been executed on its true and false sides. It should be noted that the basis set is not unique. In fact, a number of different basis sets can be derived for a given procedural design.

B. Dynamic Domain Reduction (DDR)

DDR is the technique that creates a set of values that executes a specific path. It transforms source code to a control Flow Graph (CFG). A CFG is a directed graph that represents the control structure of the program. Each node in the graph is a basic block, a junction, or a decision node [8].

C. Test Case Generation Technique

DDR uses the GetSplit algorithm to find a split point to divide the domain. The GetSplit algorithm is as follows:

Algorithm GetSplit (LeftDom, RightDom, SchIndx)

Precondition

LeftDom and RightDom are initialized appropriately And SchIndx is one more than the last time GetSplit was called with these domains for this expression.

Postcondition


Input

LeftDom: left expr 's domain with Bot and Top values
RightDom: right expr 's domain with Bot and Top values

Output

Split—a value that divides a domain of values into two sub domains.

BEGIN

— Compute the current search point
— srchPt = (1/2, 1/4, 1/8, 3/8 …)
— Try to equally split the left and right expression’s domains.

END IF
RETURN split
END GetSplit

In the dynamic domain reduction procedure, loops are handled dynamically instead of finding all possible paths. The procedure exits the loop and continues traversing the path on the node after the loop. This eliminates the need for loop unrolling, which allows more realistic programs to be handled. [2][7]

IV. PROPOSED TECHNIQUE

A. Objectives

1) In this study execution code’s algorithm is developed to optimize the testing efficiency by fetching test inputs from the database which will reduce time, effort & no of executions. Here an efficient code is developed to fetch test data from database and to fetch the data from data table to increase the execution speed, decrease the effort and increase testing efficiency.

2) To develop execution code for fetching the data from data Table which is used to retrieve test case data directly from data table without expecting data to be inputted. By implementing this procedure, it will automate the execution process and cost & effort involved in doing manual work will be diminished.

3) Developing execution code: to calculate overall time required to find an efficiency of execution in terms of speed example test run.

In this paper, a new algorithm is used to meet the above-mentioned objectives, using the following steps.

A. Test Cases Generation Technique

There are four steps to generate test cases:

1) Finding all possible constraints from start to finish nodes. A Constraint is a pair of algebraic expressions which dictate conditions of variables between start and finish nodes (> , >=, <, <=, ==, !=).

2) Identifying the variables with maximum and minimum values in the path, if any. Using conditions dictated by the constraints, two variables, one with maximum value and the other with minimum value, can be identified. To reduce the Test cases, the maximum variable would be set at the highest value within its range, while assigning the minimum variable at the lowest possible value of its range.

3) Finding constant values in the path, if any. When constant values can be found for any variable in the path, the values would then be assigned to the given variables at each node.
Using all of the above-mentioned values to create a table to present all possible test cases.

B. Expected Results
Using the methodology, the new algorithm would have the following characteristics:
1) Number of test cases. The number of test cases is smaller since each variable has a fixed value, either as maximum, minimum or constant values.
2) Automatic test cases generation. The test cases can be automatically generated with the reduction process.
3) Less time to test run. A single generation of test cases reduces the time of test run and compilation.

V. EVALUATION
A comparative evaluation has been made between the Proposed Techniques, the Existing Technique (Get Split Algorithm technique). The following areas are used to compare with existing techniques:
1) Number of test cases
2) Reduction percentage of test cases
3) Compilation time
The evaluation is described using two examples

A. Example
The function value takes three marks as input such as mark1, mark2, mark3 and returns some total mark for student depending upon the performance.

1. Source code

```vbnet
int Result(mark1,mark2,mark3)
{
    int total;
    Total=0;
    If(mark1<mark2)
    {
        Mark3=mark3+5;
        If (mark1<mark3)
        sum=mark1+10;
        Else
        Total=mark1+5;
    }
    If (mark1-mark3)
    sum=mark1+10;
    Else
    Total=mark1+5;
    Else
    {markdown
        mark3=mark3+10;
        sum=mark1+mark2+mark3;
    }
    return (sum);
}
```

Proposed technique for increasing efficiency
In this study a Test Execution technique is adopted for making test case efficient by feeding data from database directly. Advantage of this is once we store test data in database we can use it for many test run and save the overall time & effort and for this purpose a piece of code is developed as follows

```vbnet
Option Explicit
Dim con,rs
Set
con=createobject("adodb.connection")
Set rs=createobject("adodb.recordset")
con.provider="(Microsoft.jet.oledb.4.0)"
con.open "c:\document ans settings\mydocuments\test.mdb:"
rs.open "select * from test","con db until rs.eof=true
invokeapplication "c:\program files\test.exe"
Diolog("Result").Activate
Diolog("Result").WinEdit("mark1:").Set
rs.Fields("m1")
Diolog("Result").WinEdit("mark2:").Set
rs.Fields("m2")
Diolog("Result").WinEdit("mark3:").Set
rs.Fields("m3")
W in dow("testapplication").close
Rs.movenext
loop
```

Another approach used is first feeding all test case data into data table and then using it from the data table of excel by using following

```vbnet
proposed code
m1=datatable("mark1",1)
m2=datatable("mark2",1)
m3=datatable("mark3",1)
invokeapplication "c:\program files\test.exe"
Diolog("Result").Activate
Diolog("Result").WinEdit("mark1:").Set m1
Diolog("Result").WinEdit("mark2:").Set m2
Diolog("Result").WinEdit("mark3:").Set m3
W in dow("testapplication").close
```

EXECUTION Code Used For Calculating Test Case Time =>

```vbnet
Service.startTransaction T= Result(m1,m2,m3)
Dim m1,m2,m3,ST,ET,TT
TT=ST-ET
ST=timer() msgbox ("Totaltime:="&TT)
ET=timer() Service.EndTransaction
```

By using this proposed technique time required and effort required to run number of test cases are reduced to maximum extent and will contribute in increasing efficiency. Execution code for calculating test case time is implemented to find total time required.
3. No of independent path:
Path1: 1, 2, 4, 6, 8
Path2: 1, 2, 4, 7, 8
Path3: 1, 3, 5, 8

4. Evaluation result for proposed method:
Assume that the path 1-2-4-6-8 is elected and the initial domains of the input variables are
<0 to 30>, <10 to 50>, <0 to 40>
A step follows:
1) Finding all possible constraints from start to finish nodes.
   ma1 < ma2, ma1 = ma3
2) Find minimum values in the path, if any.
   From the above conditions, it is possible to identify ma3 as the Variable with the minimum value and ma2 as the variable with Maximum value. In accordance to the finding, a value of zero, the lowest value within the range of variable ma3, can then be assigned to ma3 while the value of ma2 can be set at 50, the highest value of the variable.
3) Finding constant values in the path, if any.
   Ma1 constant value for variable ma3 found on node 2 of the path has been used to replace the fix value of ma3 (10) at the node.
4) Using all of the above-mentioned values to create a table to Present all possible test cases:
   ma1 value is 10..30, ma2 as the Variable with maximum value = 50, ma3 as the variable with Minimum value = 10.
Reduced test cases:

Evaluation result for existing method:
Assume that the path 1-2-4-6-8 is elected and the initial Domains of the input variables are
<0 to 30>, <10 to 50>, <0 to 40>
A step follows:
1. Finding all possible constraints from start to finish nodes.
   ma1 < ma2, ma1 = ma3 = 10
2. Calculate split value and splitting intervals for all constraints.
   (i) For constraints ma1 < ma2
      Splitting values are 8, 10, 11, 13, 15. We choose the split value = 15 from above mentioned values. Then divided the input domain into two intervals.
VI. EVALUATION RESULTS

Total possible test case came from number values on each variable $31^*41^*41$.

Saving (%) = $100-((100*Reduced \text{ Test Case})/All \text{ Possible Test Case})$.

VIII. CONCLUSION

The new implemented technique has achieved higher reduction Percentage of the test cases by fetching data directly from the data table or D.B and running as many times as needed. Furthermore, because It retrieves test data directly from data table or D.B it takes less time Among the one existing technique. Based on the observation done, The proposed method can be considered a superior technique From all others available in current literatures. and it diminishes no of executions The proposed Technique Lies In its requirement for Identification Of Fix values For All variables, either as Maximum, minimum or constant values.

IX. REFERENCES


