ABSTRACT: Software development life cycle is a systematic way of developing software which describes phases of the software development and the sequence in which these phases are executed. Each phase generates deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced according to the design which is called development phase. After coding and development the testing verifies the deliverable of the implementation phase against requirements. Almost all software development models contain software testing as one phase but testing require at each phase of SDLC. Different testing technique can apply on different phases based on software quality attributes. Software testing majorly categorized in to functional and structural testing each of them focus on different aspect of software. Both play vital role in software development life cycle by assuring software quality, each of them can use to measure different quality attributes. Since, both structural and functional testing faces a lot of challenges during manual conduction.

Focus of this research paper is to find out significance of structural testing in SDLC along with challenges of structural testing methodology “DD path testing” in manual environment and suggest suitable solution to face these challenges. Suggested solution describes which steps involve in DD path testing and how they can automate.

Keywords: Software Development Life Cycle, Decision to decision, Quality Assurance, Software Testing, Path Testing

Received: 14 March 2015, Revised 19 April 2015, Accepted 25 April 2015

1. Introduction

This Section will provide introduction about SDLC, Quality Assessment in SDLC & DD path Testing.

1.1 Significance of Software Testing In SDLC

Software Testing is a practice of finding errors while executing a program so that we get a zero defect software. It is aimed at assessing the fitness or usability of a program. Software testing is an important means of accessing quality of software.[14] Quality means meeting requirements or from customer’s point of view “fit for use”[11]. Since testing typically consumes 40~50% of development efforts, and consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering.[1]

Role of software testing in SDLC depends upon software testing type and SDLC phase on which testing technique will be implementing. To assure software quality one should familiar with quality attributes related to each testing type.

Number of quality attributes defined for quality assurance of software product. Each attribute can check by specific type of software testing. Quality attributes broadly categorize in to Static & Dynamic attributes. Static
attributes focus on code while dynamic attributes focus on behavior/functionality of software. Dynamic attributes can be obtained by basic path testing.[12].

1.2 DD-path Testing
Software testing process faces lot of challenges both in manual as well as automatic ways of software testing. Sometimes testers can add complications during testing because of their unskilled way of work.

Modern software engineering paradigm is moving towards complex software architectures and bulky software code. It is not possible to test every piece of code in software in Manual testing. If one tries all such combinations, one can never ship the product in time since the testing is always performed on the sampling frame. Moreover, the selection of test data requires a better understanding of methods for selecting test data. Sometimes one does not pay the necessary attention that processes are defined by the company and they are for What purposes. Maintaining Relationship with developers is also big Challenge. Tester requires skill to manage this positive relationship and even completion being testers. There are simply hundreds of developers and testers can make excuses when they do not agree with some points. For this tester also requires good communication, troubleshooting and analysis skills.

Testers are responsible for communicating with clients to understand the requirements. For this reason tester should have good listening and understanding skills otherwise tester fails to understand the requirements and He will be not able to execute testing of application properly.

2. Background

Software Development Life cycle is sequences of phases for development of software. These phases connects with each other, requirement gathering gather user requirement which is input for analysis & design phase. Design is use to develop code at last phase testing is use to match coding/functionality of software with user requirement gather in requirement gathering phase.

Testing is key to assure quality of software. The software should perform tasks what the user really requires. The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements is testing. So testing is way to assure software quality.

2.1 Software Testing in relation with SDLC
Most of SDLC process model use software testing as one phase in SDLC but problem is to find whether it is appropriate to carry out testing only in one phase or whether it should be part of each phase. Testing can be implementing after coding phase or it could be a V&V process. V & V (Verification and Validation) is a whole life-cycle process and it must be applied at each stage in the software process. It has two principal objectives

- The detection of errors in a system
- The evaluation of whether or not the system is functional in an effective situation.[13]

2.2 Software Testing Types In Relation With Quality Attributes:
Software quality cannot achieve by a single factor or by accessing already complete product. Quality assessment is enduring process. A number of factor present to measure product quality. It is not possible to use single testing technique to measure all of these factors. There are number of software testing techniques meanwhile number of quality attributes (Factors). Then how to identify proper software testing type for each quality attribute.

2.3 Black Box Testing
Black Box testing is type of testing in which a product tested without having the knowledge of the internal workings of the code. For example, when black box testing is applied to software engineering, the tester would only know the “permissible” inputs and expected results should be, but not how the program in fact arrives at those results.

The reason behind this strategy is that black box testing focus on product specification, no other knowledge of the program is necessary. Black box testing performs by selecting test cases on basis of requirement and design specification.[12]

Method uses to carry out black box testing are unit testing, boundary value analysis, stress testing, Performance testing, Usability Testing, Equivalence Partitioning, comparison testing, Model base testing, Fuzz Testing, Beta Testing, Regression Testing, Ad hoc Testing, End to End testing & syntax Testing.

2.4 White Box Testing
The opposite of Black box testing would be white box (also known as glass box or clear box) testing, where test data are derived from straight inspection of the code to be tested. For white box testing, the test cases cannot be determined until the code has actually been written. White-box testing of software is based on a careful review of procedural details. Logical paths through the software are tested by providing test cases that exercise specific sets of conditions and / or loops. The “program status” can be examined at different points to analyze if the expected status corresponds to the actual situation. Basic methodologies to carry out white box testing are basic path testing, DD (decision to decision) path testing, Control structure Testing.

2.5 Quality Attributes for DD-path Testing
Different quality attributes use to measure software quality, in order to find significance of DD-path graph testing in SDLC, it is important to study different quality attributes so that their relation with testing type can identify which may help to find exact testing type for each phase of
A number of quality attributes use to measure quality e.g.
- Software should be design according to level of user for whom it may design so that target user can easily understand it.
- Software product should be complete in all perspectives e.g. if a software is using external libraries then there should be reference for that library.
- Software should facilitate to change according to changing environment.
- Software should easy to use and should have friendly interface.
- Software should efficient to use i.e. it should fulfill user requirement without wasting extra resources (memory & time)
- Software should able to protect data against unauthorized users.

Along with above mentioned, there are number of other attributes to measure quality of software.

3. Problem Description

This section describes the problem in detail.

3.1 Significance of White Box Testing

In SDLC testing significance can be analyze by counting whether testing carried out throughout the cycle or only during one phase, testing type and its relation with software quality attribute. In order to find role of white box testing in SDLC it is important to know which quality attribute can measure by white box testing.

3.2 Role of Path Testing In Quality Assessment

DD-path testing is type of basic path testing which is testing mechanism purpose by McCabe. Its purpose is to design logical complexity measure of procedural design and use it as a guide to define basic set of executable paths.[B]

DD Path testing is useful as it corroborate that how much piece of code is check for error, but how DD path testing is associated with software quality assurance and during which phase of SDLC DD path testing should perform is question to think about.

3.3 Basic Path Method: DD-PATH TESTING

The core reason that path testing is implemented is to provide code with a level of test coverage; that is, to find out how much of a piece of software has been examined for faults. DD-Paths are chains of nodes in a directed graph that adhere to certain definitions. Each chain can be broken down into a different type of DD-Path, the result of which ends up as being a graph of DD-Paths [2]. Definitions for DD-Path node are given bellow:

1. For DD path testing knowledge of code and internal structure is necessary, therefore a skilled tester is needed to carry out this type of testing due to which cost of the software testing increase.

2. As it is mentioned above that at present software architecture are becoming complex having huge code due to which it is not possible to test each and every path of the loops in program, which may create hurdles, resulting in failure of the testing process.

3. In order to ensure complete coverage of program logic test cases should write after complete understanding of code as well as programming language and logic. This means manual DD path testing is impossible for large systems. However selection of important logical paths and data structure make structural testing possible to conduct practically.

3.4 Testing Automation:

It has already been studied that automated testing as called dynamic testing in which testing team run test cases on testing tool. They divided automated testing in four major types, Correctness testing, Performance testing, Reliability testing, Security testing [3]. Automation of Testing process can be use to overcome issues which occurs during manual implementation of DD path testing. Automation of DD path testing can helpful to conduct testing in smooth manner as compare to manual testing as in it.

• Tester involvement is limited to use the automated tool which will ensure that testing process will not suffer from errors occurs due to execution of testing process by un-experienced testers.

• Automated approach for testing can save time as for complex software containing bulk of code manual testing conduction can take more time to complete.

• DD path testing require clear understanding of code and logic so skilled tester require to fulfill this requirement in manual testing process due to which cost will increase but it can control by automation of testing as, In Automated DD path testing understanding of code and logic is carry out by NLP which can save the cost of testing process.

4. Proposed Methodology

4.1 Main Algorithm Steps

Building DD path graph is sequential process in both manual and automated approaches. Steps need to follow sequentially given bellow:

• Input Acquisition
• Building Control Flow Graph
• Assigning DD path Type to each node of CFG
• Convert CFG in to DD-path graph

To automate construction of DD path graph all steps listed above should done via automated tool. To design such tool there should be some algorithm for performing all these
steps. In purpose solution we present algorithm to automate each step of DD-graph construction.

4.1.1 Input Acquisition
Piece of code that needs to test will be taken as input. Input acquisition will do by tester who enter the code (need to test) in to automated system for further processing. Input code file will be first read by system by giving path, this file will be read by system and each statement will assign a line number and then it will save in to another file.

Algorithm for Reading Input and Assigning Line Number:
Following is the algorithm used to read input source code file and assign line numbers to the input source code:

1. Initialize streamReader by giving path of source code
2. Initialize streamWriter by giving path of new file for source code
3. Declare line as string = streamReader.ReadLine()
4. Declare LineNumber as integer = 1
5. While(line!=null)
   Display lineNumber + line
   streamWriter.WriteLine(line)
   Line = streamReader.readLine()
   Increment in LineNumber
6. Close streamReader
7. Close streamWriter

Example of Input Acquisition: Following is the example of input acquisition:
Compute Factorial ()
int num;
int fact=1;
int i=1;
Console.Write("Enter Number:");
num=Convert.ToInt32(Console.ReadLine());
While (i<=num)
{
fact=fact * i;
i=i+1;
}
Console.WriteLine("Factorial is "+ fact);
Console.ReadLine();

4.1.2 Assigning Line Numbers:
In this step, each line of the source code is assigned a line number.
1. Dim n As Integer
2. Dim k as Integer = 2
3. Dim prime As Boolean = True

4. Console.Write("Enter num:"")
5. n=Convert.ToInt32(Console.ReadLine())
6. While (k < n - 1)
7. If (n Mod k = 0) Then
8. prime = False
End If
9. k = k + 1
End While
10. Console.WriteLine(prime)

4.1.3 Parsing and Building CFG
After input acquisition next step is to build the program graph or control flow graph. This stage is very important and need to execute carefully as it provide complete understanding of code and logic which will use as a basis for DD graph construction.

- Control flow graph describe the internal code and its logical structures. In control flow graph every statement will be converted in to a node and relationship between nodes or flow of control between nodes will represent by edges. Node will assign number usually line number of statement can use as node number.
- Parsing is use to understand each statement structure simple, branch or looping. It will help to understand syntax of code in order to identify possible paths for each statement.

Purpose Algorithm to build Control Flow graph:
Following is the algorithm used to build a CFG from source code.
1. Read Input File with line number obtained from algorithm1.
2. Identify type of statement of each line by tokenization.
3. if (statement = Assignment) or (statement = copy statement)
   • Build node for statement with single edge
4. else if (statement=branch structure) or (statement = looping structure)
   • Build node for statement with multiple execution edges depend upon code path
5. End if

4. Console.Write("Enter num:"")
5. n=Convert.ToInt32(Console.ReadLine())
6. While (k < n - 1)
7. If (n Mod k = 0) Then
8. prime = False
End If
9. k = k + 1
End While
10. Console.WriteLine(prime)
4.1.4 Computing In–Degree of Each node in control Flow Graph
After building CFG (control flow graph) in-degree of each node in CFG will compute, so that they could assign DD-Path type according to definition.

· In-Degree: can define as number of edges (path) a node contain as terminal node (end node).

Purposed Algorithm to find In-Degree of each node in CFG: Following is algorithm use to compute in-degree for each node of Control Flow Graph.
1. float in-Degree
2. In-Degree= Num of edges pointed toward node
3. Return in-degree
4.1.5 Computing Out –Degree of Each node in control Flow Graph
After building CFG (control flow graph) out-degree of each node in CFG will compute, so that they could assign DD-Path type according to definition.

· Out-Degree: can define as number of edges (path) a node contain as start node.

Purposed Algorithm to find Out-Degree of each node in CFG: Following is algorithm use to compute out-degree of each node in Control Flow Graph.
1. float out-Degree
2. out-Degree= Num of edges pointed out from node
3. Return out-degree

<table>
<thead>
<tr>
<th>Node Number</th>
<th>In-degree</th>
<th>Out-Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. In-degree and out-Degree

4.1.6 Assign DD-Path Type:
Base on in-degree and out-degree of each node all node in CFG will assign DD path type according to DD path definition.

1. Node with in degree = 0 considered as DD-path type 1.
2. Node with outdegree = 0 considered as DD-path type 2.
3. Node with Indegree >= 2 or Outdegree >= 2 considered as DD-path type 3.
4. Node with Indegree = 1 and Outdegree = 1 considered as DD-path type 4.
5. Consecutive nodes with in-degree=1 and out-degree=1 combined in to one single node and considered as DD-path type 5.

Propose Algorithm to Assign DD-path Types: Following is algorithm use to assign DD-path type to each node of CFG.

Building DD-path type (Num)
1. Declare Type-Array as integer array with size number
2. Declare DD-Path-Array as char with size number
3. Declare in-degree and out-degree as float
4. For(i=1;i<=num; i++)
5. For(int j=65;j<=90;j++)
6. In-degree=Calculate Compute-in-Degree(i)
7. Out-degree=Calculate Compute-out-Degree(i)
8. if (in-degree==0)
9. Type-Array[i]=1
10. DD-Path-Array[i] =char(j)
11. Elseif(out-degree==0)
12. Type-Array[i] =2
13. DD-Path-Array [i] =char(j)
14. elseif(indegree>=2)or(out-degree>=2)
15. Type-Array [i]=3
16. DD-Path-Array [i] =char(j)
17. Elseif(indegree==1)and(out-degree==1)
18. if(Type-Array [i-1]==4)
19. i - - ;
20. Type-Array [i]=5
21. else
22. Type-Array [i]= 4;
23. DD-Path-Array [i] =char(j)
24. End if

4.1.7 Building DD-Graph:
When each node get its DD path type CFG can converted in to DD path graph by representing all information in following pattern (Refer Figure)

5. Experiments & Results
In order to check working of purposed methodology we had perform it on case study “Lexical analyzer”. Results of discussed case study given bellow according to statements types which were divided in following type:

1. Simple Statements: A simple statement contains
<table>
<thead>
<tr>
<th>Node Number</th>
<th>In-degree and out-Degree</th>
<th>DD-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In-degree=0</td>
<td>Type1</td>
</tr>
<tr>
<td>2-5</td>
<td>Consecutive nodes with Type 4</td>
<td>Type5</td>
</tr>
<tr>
<td>6</td>
<td>In-Degree&gt;=2/Out-Degree&gt;=2</td>
<td>Type3</td>
</tr>
<tr>
<td>7</td>
<td>In-Degree&gt;=2/Out-Degree&gt;=2</td>
<td>Type3</td>
</tr>
<tr>
<td>8</td>
<td>In-degree=1 and out-degree=1</td>
<td>Type4</td>
</tr>
<tr>
<td>9</td>
<td>In-Degree&gt;=2/Out-Degree&gt;=2</td>
<td>Type3</td>
</tr>
<tr>
<td>10</td>
<td>Out-Degree=0</td>
<td>Type2</td>
</tr>
</tbody>
</table>

Table 2. DD path type of nodes

Figure 2. DD-path graph

Figure 3. Architecture of Solution
assignment statements and variable declaration and initialization statements.

2. **Conditional Statements**: In conditional statements if-else, if-else if and switch statement included.

3. **Loop statements**: In looping statements while, do-while, for and for-each loop included.

Results are shown by organizing obtained information in given groups.

1. **Detected**: Nodes and edges of each statement which retrieved by applied algorithm organized in detected group.

2. **Not Detected**: Nodes and edges of each statement which are not retrieved by applied algorithm placed in not detected group.

3. **Wrong Detected/Dual Detected**: Nodes and edges which are retrieved by applied algorithm but in wrong statement type are placed in wrong detected group & which are retrieved by applied algorithm in two statement type are placed in dual detected group

**Result Measurement:**

1. **Precision**: precision has ability to measure correct number of results produced by the algorithm. Precision=Detected/ Detected+ Wrong/Dual Detected*100

2. **Recall**: recall has ability to measure the completeness of the results produced by algorithm. Following formula can be used to calculate the recall.

   Recall: Detected/Detected+ Not Detected*100

3. **F-measure**: The traditional F-measure is a mean of Precision and Recall. F-measure is the harmonic mean or the "standard" average of total, correct, and incorrect results. Following formula can be used to calculate the F-measure

   $F\text{-}\text{Measure} = \frac{2(R \times P)}{R + P}$

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Total</th>
<th>Detected</th>
<th>Not Detected</th>
<th>Wrong/Dual Detected</th>
<th>Prec.</th>
<th>rec.</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>Nodes 76</td>
<td>68</td>
<td>1</td>
<td>7</td>
<td>90%</td>
<td>98%</td>
<td>93%</td>
</tr>
<tr>
<td></td>
<td>Edges 76</td>
<td>64</td>
<td>6</td>
<td>6</td>
<td>91%</td>
<td>91%</td>
<td>91%</td>
</tr>
<tr>
<td>Conditional</td>
<td>Nodes 45</td>
<td>35</td>
<td>7</td>
<td>3</td>
<td>92%</td>
<td>83%</td>
<td>87%</td>
</tr>
<tr>
<td></td>
<td>Edges 36</td>
<td>25</td>
<td>7</td>
<td>4</td>
<td>86%</td>
<td>78%</td>
<td>81%</td>
</tr>
<tr>
<td>While Loop</td>
<td>Nodes 125</td>
<td>118</td>
<td>2</td>
<td>5</td>
<td>95%</td>
<td>98%</td>
<td>96%</td>
</tr>
<tr>
<td></td>
<td>Edges 130</td>
<td>125</td>
<td>3</td>
<td>4</td>
<td>96%</td>
<td>97%</td>
<td>96%</td>
</tr>
<tr>
<td>For Loop</td>
<td>Nodes 190</td>
<td>178</td>
<td>7</td>
<td>5</td>
<td>97%</td>
<td>96%</td>
<td>96%</td>
</tr>
<tr>
<td></td>
<td>Edges 190</td>
<td>175</td>
<td>7</td>
<td>8</td>
<td>95%</td>
<td>96%</td>
<td>95%</td>
</tr>
</tbody>
</table>

Table 3. Results of case study

6. **Related Work**

The main role of software testing holds that there should be no inconsistency in the software development process. Each software development life cycle has passed through a set of common phases one or more times. So starting activities early means we can latch small glitches before they become big problems later on. Starting testing activities timely also provides the chance to review requirements for important quality attributes, to ask questions and to resolve issues. There are three different testing phases in SDLC are [11]:

i. **Test Analysis**: tester tries to know about the project.

ii. **Test Design**: tester design the test cases based on user requirement.

iii. **Test Execution**: tester execute the test cases and
7. Conclusion

7.1 Automated DD path Testing

DD-path testing is a method of structural testing in which a tester takes the source code of the software product, constructs CFG from the source code, and then assigns DD-path types to each node in CFG. Building of the source code into CFG requires a clear understanding of the code, so experienced testers need to perform DD-path graph construction which increases the cost of structural testing. Moreover, experienced testers may not be certain about completing structural testing in complex software.

In this paper, we aim to improve the throughput of structural testing and achieve the goal of more accurate and efficient software testing. To attain this above-mentioned goal, we present an automated approach to generate DD-path graphs. The presented approach is based on an algorithm that divides the problem into sub-algorithms. The first sub-algorithm accepts the source code of the software product as input and generates CFG by analyzing the statement type, converting statements into nodes and possible execution paths into edges. The generated CFG is then passed as input to the second sub-algorithm, which analyzes the DD-path type of each node up to type 4 and identifies consecutive nodes with DD-path type 4 and combines them into a single node with DD-path type 5.

Decision to Decision path testing. Solution contains sequences of steps initially code need to provide as input which may lead to improper working of product. They perform experiments to compare the results of both testing approaches and conclude that manual testing may take a long time to complete while automated testing approaches are more efficient. Therefore, automated approaches should be integrated with manual approaches to get the maximum benefits from both approaches[5-9].
which will then convert in to CFG, conversion in to CFG can be automate with help of NLP (Natural language processing),this CFG help to compute in-degree and out-degree of nodes, all nodes will assign DD path type according to their in-degree and out-degree finally DD path graph will obtain. All describe steps can be automate which can reduce hurdles occurs during manual processing of these steps, as in manual processing tester should be skilled to build proper CFG to identify possible path for each statement automation of this step will eliminate need of skilled tester which will ensure low cost testing as well as less human interruption in overall testing process.

7.2 DD path Testing Significance in SDLC
In SDLC Process Models Software testing is single phase activity i.e. it is perform at later stages of SDLC but this strategy need to change as testing should perform throughout the SDLC to gain maximum quality.

Software Quality could define by number of attributes e.g. Efficiency, reliability & completeness. It is not practically possible to check all these attributes with single testing technique, as each testing type aim to find bugs/error at different level. Therefore one testing type may not sufficient to assure quality measurement. Majorly testing types categorize in Black box, White box & Gray Box Testing. This paper aim to find quality attributes related with white box testing technique DD path Testing from above discussion it is clear that DD path testing use to generate test coverage matrix which helps to analyze how much piece of code examined for error, therefore with automated approach of DD path graph Construction of Test Coverage matrix consume less time.

DD-path testing can use to measure Completeness of software as Quality Attribute Completeness is use to check that all parts of program or code present and each part is working properly[12], DD-path provide path graph which give details of each path present in code so if there exist missing code/path it could analyze by DD-path graph.

References


