Automated DD-path Testing and its Significance in SDLC Phases

Hina Sattar[1], Imran Sarwar Bajwa[2], Umar Farooq Shafi[3]
Department Of Computer Sciences & IT
The Islamia University of Bahawalpur
Pakistan
hinasattar@yahoo[1].com,imran.sarwar@iub.edu.pk[2], umarshafi_527@yahoo.com[3]

Journal of Digital Information Management

ABSTRACT: *Software development life cycle is 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.*

Subject Categories and Descriptors
D.2.5 [SoftwareTesting and Debugging] D.2.9 [Management]: Software Life cycle

General Terms:
Software Testing, SLDC

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 attribute can 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 inthe 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)
Diaply lineNumber +line
streamWriter.Writeline(line)
Line= streamReader.read
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

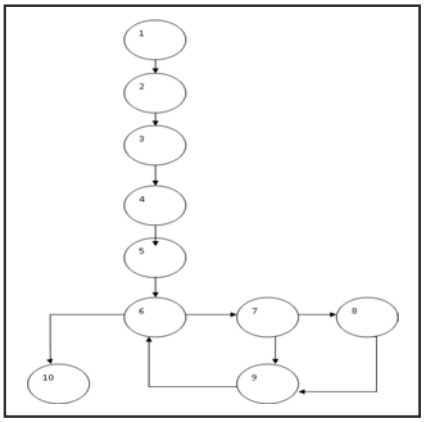


Figure 1. CFG of Example

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

Node Number	In-degree	Out-Degree
1	0	1
2	1	1
3	1	1
4	1	1
5	1	1
6	2	2
7	1	2
8	1	1
9	2	1
10	1	0

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 type1.

2. Node with outdegree = 0 considered as DD-path type2.

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 type4.

5. Consecutive nodes with in-degree=1 and out-degree=1 combined in to one single node and considered as DD-path type5.

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)
    Type-Array[i] =2
    DD-Path-Array [i] =char(j)
12. elseif(indegree>=2)or(out-degree>=2)
    Type-Array [i]=3
    DD-Path-Array [i] =char(j)
13. Elseif(indegree= =1)and(out-degree==1)
14. if(Type-Array [i-1]==4)
15. i - - ;
16. Type-Array [i]=5
17. else
18. Type-Array [i]= 4;
19. DD-Path-Array [i] =char(j)
20. 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

Node Number	In-degree and out-Degree	DD-Type
1	In-degree=0	Type1
2-5	Consecutive nodes with Type 4	Type5
6	In-Degree>=2/Out-Degree>=2	Type3
7	In-Degree>=2/Out-Degree>=2	Type 3
8	In-degree=1 and out-degree=1	Type4
9	In-Degree>=2/Out-Degree>=2	Type 3
10	Out-Degree=0	Type2

Table 2. DD path type of nodes

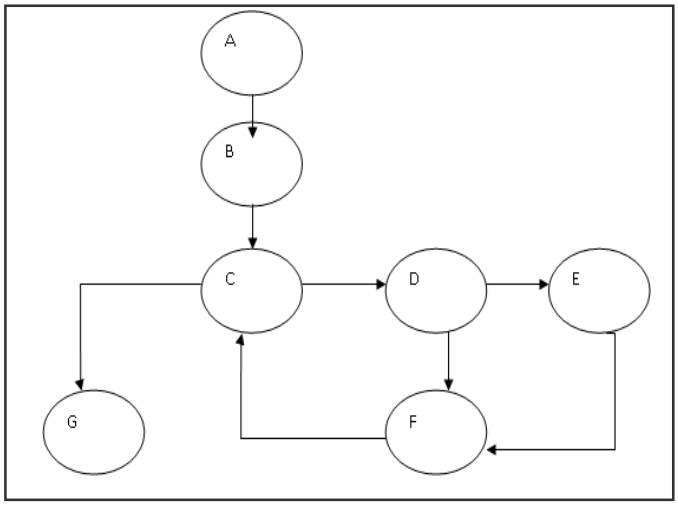


Figure 2. DD-path graph

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. Measure=2(R*P)/R+P

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

Table 3. Results of case study

6. Related Work

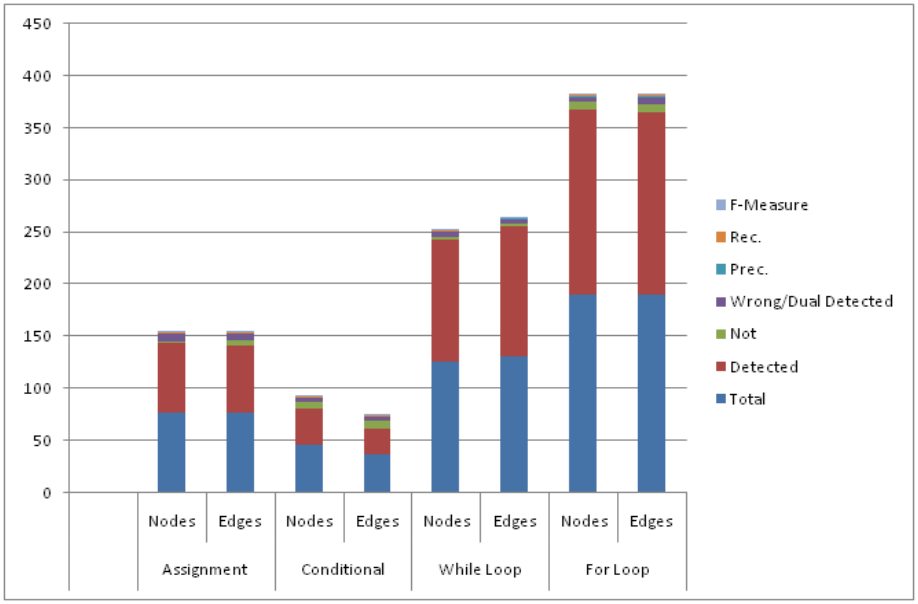
The main role of software testing holds that there should be no inconsistency in the software development process. Each software development life cycle has passed through a set of common phases one or more times. So starting activities early means we can latch small glitches before they become big problems 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

When to start testing is big question to answer, it is



Graph 1. Results of Case Study

suggested to make testing a part of each SDLC phase. It is also concluded that if testing is done at later stages of SDLC it may fails to find errors in early stages i.e. requirement gathering so it is better to start testing at early stages of SDLC to avoid big issues. Software testing perform to misuse quality but quality cannot measure by single attribute it could analyze by measuring collection of attributes ,different quality attribute need to measure at different phases of SDLC using different Testing techniques. It has already been investigated that testing can perform manually as well as automatically via some tool, manually testing totally rely on testers knowledge. Both manual and automated testing have their own benefits and limitations manual testing good to test in detail while automated testing not suitable to test in detail[4]. It is discussed that different types of faults which can cause product failure and effect quality of product, there are number of methods to find out these different kind of faults/ bugs in code. Manual and automated approaches are well known to use but major issue is that which approach should adopt to find and fix which type of fault. Errors can be detected by human tester in case of manual testing and could also detect by end user and by using some automate approach like random testing, they identify different kind of errors find by all these different method. They show that all these methods have their own advantages. Their experiments show that in manual testing, human tester can find bugs in depth or he/she can find the bugs which an automated tool may be left. But with this approach major issue is time duration as finding bugs by hand take much time to complete. This issue could resolve by automated testing tool like auto test in which random testing can perform using complete automated approach. This tool can find bugs in minimum time interval which can save time and cost of testing process, but testing tool may leave some bugs to uncover which may lead to improper working of product. They perform experiment to compare results of both testing approaches and they conclude that manual testing need which may lead to improper working of product. They perform experiment to compare results of both testing approaches and they conclude that manual testing need long time duration to complete while automated approach take minimum time therefore automated approach should integrate with manual approach to get maximum benefits from both approaches.[5-9].

7. Conclusion

7.1 Automated DD path Testing

DD-path testing is method of structural testing in which a tester take source code of software product, construct CFG from source code and then assign DD-path type to each node in CFG. Building of source code in to CFG needs clear understating of code, therefore experienced tester need to perform DD-path graph construction which increases cost of structural testing moreover experienced tester may not sure about completion of structural testing in case of complex software.

In, this paper we aim to improve the throughput of structural testing process and achieve the goal of more accurate and efficient software testing. To attain above mention goal we present an automated approach to generate DD-path graph. The presented approach was based on algorithm which will divided in to sub algorithms. First algorithm accepts source of software product as input and generate CFG by analyzing statement type it converted statement in to node and possible execution path in to edges. This CFG will pass as input to second sub algorithm which will analyze DD-path type of each node up to type 4 and identify consecutive nodes with DD-path type 4 and combine them in to single node with DD-path type 5.

Decision to Decision path testing .Solution contain sequences of steps initially code need to provide as input

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 realted 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 eaxamined 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

[1] Nguyen, B. N., Robbins, B., Banerjee, I., Memon, A. (2014). GUITAR: an innovative tool for automated testing of GUI-driven software. *Automated Software Engineering*, *21* (1), 65-105.

[2] Yu, T., Srisa-an, W., & Rothermel, G. (2014, May). SimRT: an automated framework to support regression testing for data races. In *Proceedings of the 36th International Conference on Software Engineering* (pp. 48-59). ACM.

[3] Sawant, A. A., Bari, H.P. and Chawan, M.P.(2012). Software Testing Techniques and Strategies. International *Journal of Engineering Research and Applications (IJERA)*, 2 (3) 980-986.

[4] Leitner, A., Ciupa, I., Meyer, B., Howard, M. (2007). Reconciling manual and automated testing: The autotest experience. In System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on (261a-261a). IEEE.

[5] Ciupa, I., Meyer, B., Oriol, M., Pretschner, A. (2008, November). Finding faults: Manual testing vs. random+ testing vs. user reports. In Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on(157-166). IEEE.

[6] Srivastava, P. R., Baby, K. (2010). Automated software testing using metahurestic technique based on an ant colony optimization. *In: Electronic System Design (ISED)*, 2010 International Symposium on (235-240). IEEE.

[7] Just, R., Jalali, D., Inozemtseva, L., Ernst, M. D., Holmes, R., Fraser, G. (2014, November). Are mutants a valid substitute for real faults in' software testing?. *In:* Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (654-665). ACM.

[8] Alzabidi, M., Kumar, A., Shaligram, A. D. (2009). Automatic Software structural testing by using Evolutionary Algorithms for test data generations.IJCNS *International Journal of Computer Science and Network Security,* 9 (4) 390-395.

[9] Vanmali, M., Last, M., Kandel, A. (2002). Using a neural network in the software testing process. *International Journal of Intelligent Systems,* 17(1) 45-62.

[10] Last, M., Friedman, M., Kandel, A. (2003). The data mining approach to automated software testing. *In:* Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (p. 388-396). ACM.

[11] Khan,M. Khan,F.(2014) Importance of Software Testing in Software Development Life Cycle .IJCSI *International Journal of Computer Science Issues,* 11 (2) March 2014.

[12] Harman, M., Jia, Y., Zhang, Y. (2015, April). Achievements, open problems and challenges for search based software testing. *In:* Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on (1-12). IEEE.

[13] Agrawal, H. (2015). *U.S. Patent No. 20,150, 234, 734*. Washington, DC: U.S. Patent and Trademark Office.

[14] Chauhan, R. K., Singh, I. (2014). Latest Research and Development on Software Testing Techniques and Tools. *International Journal of Current Engineering and Technology*, 4 (4) (Aug 2014).