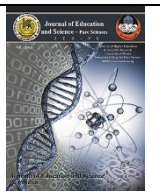
EDUSJ, Vol, 32, No: 2, 2023 (65-75)

Journal of Education and Science (ISSN 1812-125X)

www.edusj.mosuljournals.com



Data Flow Testing and Tools Review

A. H. Ali N. N. Saleem

Department of Software Engineering, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq

Article information

Article history:
Received: January 10, 2023
Accepted: March 01, 2023
Available online: June 01, 2023

Keywords:
Data Flow Testing (DFT)
Test Data
Flow Analysis
Coverage Tracking
Testing tool

Correspondence:
A. H. Ali
abdullah.20csp13@student.uomosul.edu.iq

Abstract

Software engineering always strives to develop and identify software pitfalls and errors before publishing the software product, in testing the software. Bugs can appear during any stage of development or testing, even after the product has been released. This paper describes different methodologies for data flow testing. Since testing is the process of running a program to identify errors, we need to increase the accuracy of the coverage area by including dataflow elements based on aliases and avoiding useless elements that reduce the overall coverage to increase the applicability and effectiveness of the dataflow test. This page looks at data flow testing, which is a type of basic test (white box). Information flow testing is divided into two main points: properties / usage test and a set of tests embedding measurements; And divide the program into parts according to its factors to make testing programming frameworks more straightforward. It also describes the steps for performing data flow testing as well as how to design test suites that take anomalies into account. It also examines and discusses methods used to date to perform data flow testing. These approaches include node-based design, trend-finding coverage, web application comparison, and analytical testing.

DOI:10.33899/edusj.2023.137611.1315, ©Authors, 2023, College of Education for Pure Sciences, University of Mosul.

1. Introduction

Computerized programming testing is a prerequisite for programming hardware that examines, approves, creates, conveys, and assesses the whole programming item. Every time the status of the appropriate software is looked at, the inspection equipment raises the bar for specialized computerized testing. Planning, developing test cases, carrying out the test, validating the results, and debugging are all steps in the testing process. A successful test needs a solid test strategy that concentrates on evaluating new services in a specific testing setting [1]. Regression testing is carried out on a software application after it has gone through revision cycles to make sure that the addition of new features and adjustments hasn't negatively impacted the code that was already in place. To evaluate the software's response and stability, both successful and unsuccessful test cases are used throughout testing.

System testing entails independently checking the interfaces and dependencies between each subsystem before combining them to test the system as a whole to ensure that the functionalities outlined in Software Requirements Specification are met. To ensure that system requirements are met in real-world or simulated settings, integration testing takes into account all system components, including hardware, software, and human interaction. Users typically do acceptance testing while using a BETA version of the program. Repairing software flaws after they have been deployed can cost up to a hundred times more than not letting the problem arise in the first place [2].

Utilizing pre-programmed tools, automated testing maximizes test coverage while requiring the least amount of time and effort [3]. Automation is not a viable option for projects that are running late or over budget. To automate a test, a whole development effort is needed, including objective and strategy planning, requirement creation, alternative analysis, execution, and assessment. Tools for automated testing are programs that can run other programs using test scripts. Manual testing cannot

guarantee that a test script can be performed again with the same inputs and sequence when a potential issue is found. Reusing scripts not only saves time but also promotes program stability.

A class of testing strategies known as DF testing (DFT) picks program pathways to check definition-use relations to information objects. Fully intent on recognizing potential DF abnormalities, it spans the holes between branch/articulation testing the whole way testing testing. At the point when utilized as a test determination rule, DFT can offer a more exhaustive testing strategy to ensure the test reasonableness of a piece of programming and track down blames that less severe standards could not be guaranteed to miss.

2. Related Works

In [5], the suggested algorithm An Improved Round Robin CPU Scheduling Algorithm with Varying Time Quantum outperformed the traditional RR approach in terms of performance. The suggested IRRVQ scheduling algorithm decreased waiting and turnaround times, and thus enhanced system performance. The algorithm was compared considering two sets of processes. The first consists of 5 processes, with burst times in the range of 10-32, at an average of 21-time units for each process. Processes were arranged in ascending order of burst times with a fixed arrival time for all of them equal to zero. The second consists of 5 processes, with an arrival time in the range of 5-36, an average of 18-time units for each process. The processes were arranged according to their arrival times. The comparison was made with the traditional RR, with time slices of 10, 6, and 5 for the first group, and 11 and 7 for the second group.

[3] presented an improved dynamic round robin algorithm which is based on a dynamic time quantum. The algorithm was tested on three sets of processes; The first consists of 7 processes arranged in ascending order, with execution times in the range of 20-120, and with an average of 60-time units. The second is of 5 processes arranged according to their arrival times, with an average burst time of 40-time units. And the third is of 7 processes arranged in descending order according to burst time, with burst times in the range of 5-80 time units, at a rate of 31.5-time units for each process. The algorithm was compared with two other algorithms, one was round robin, with a time slice of 40, 25, and 20 for the three groups, respectively. Traditional Round Robin and Self Adjustment Round Robin were compared with the suggested algorithm. Due to shorter average waiting and turnaround times, the experimental findings demonstrate that the suggested algorithm is superior to Round Robin and Self Adjustment Round Robin. Round Robin and SARR, however, performed better than the suggested algorithm when the burst times of the processes were ordered decreasingly.

[6] declared that their proposed algorithm had reduced average waiting time, average turnaround time, and the number of context switches. The algorithm was applied to 3 groups of processes; The first consists of 5 processes arranged in ascending order of burst time, with burst times in the range 14-77 with an average of 46.5-time units per process and a fixed arrival time of zero. The second consists of 5 processes that were arranged in ascending order, with a burst time in the range of 22-74 at a rate of 49.4-time units for each process, and an arrival time in the range of 0-9, with an average arrival time of one process per 5.75-time units. The third consists of 4 processes arranged randomly according to the arrival time with burst times in the range of 15-85, with an average burst time of 51.75-time units, and arrival times in the range of 0-20, with an arrival rate of one process per 13-time units. The algorithm provided better performance than simple RR. Researchers explained that when the quantum had been increased and a process with a small burst time was to arrive in the middle of execution then the algorithm could suffer - the new one has to wait more time than the basic RR.

[7] proposed an algorithm that had improved performance. The algorithm was tested on one set of 5 processes, with execution times in the range of 26-82, with an average execution time of 50-time units. With a fixed access time of zero, the algorithm was compared with five other algorithms. Among the algorithms is the traditional round robin with a time slice of 25-time units. The algorithm considered arranging operations in ascending order according to the execution time. In comparison to the RR method, there were a lot fewer context switch. Additionally, it shortened turnaround and average waiting times.

[8] announced that their Modified Median Round Robin Algorithm (MMRRA) was tested on five processes with burst times in the range of 45–73-time units and compared with five other algorithms. When it was compared to the traditional RR, IRR, IMRRSJF, HLVQTRR, and DRRCP CPU scheduling algorithms, it offered a more effective solution. MMRRA dramatically reduces the number of context switches (NCS), but the average waiting time and turnaround time were not greatly impacted.

[9] presented Smart Round Robin which outperformed Traditional Round Robin and other algorithms in terms of AWT and ATAT. First, the algorithm executes processes with a short remaining burst time and provides a dynamic time quantum for each cycle. The algorithm was tested on four sets of processes; The first consists of four processes with a large burst time in a range of 8-34 and an average of 18-time units for each process, the second of five processes with a short burst time in a range of 2-9 and an average of 5-time units for each process, the third of four processes with burst times in a wide range of 11-82 time units and an average of 46.5-time units for each process, and the fourth of four processes in a range of 5-11, at a rate of 7.5-time units per process. The arrival times for all processes in the first three groups were 0, while the fourth group had different arrival times in a range of 0-7, with an average of one process arriving every three-time units. The algorithm was

compared with the traditional round robin algorithm with time slices of 6, 4, 20, and 2-time units for the four groups, respectively. The algorithm considered arranging processes in ascending order according to the burst time.
In [10], a comparison of calculations was performed on two groups of processes; The first consists of 6 processes, with a burst time in the range of 4-8, at a rate of 6-time units for each process, and an arrival time in a range of 0-13. The second consists of 5 processes with burst times in the range of 7-90 and arrival times in the range of 0-10. The algorithm considered adding the processes to the ready queue as soon as it arrives, but it chooses the shortest one when allocating the CPU to the next process. The algorithm had been compared to Dynamic Quantum Using the Mean Average Round Robin (ANRR), Shortest Remaining Burst RR (SRBRR), An Optimized Round Robin Algorithm (ORR), Adaptive Round Robin Algorithm (ARR), and Simple Round Robin Algorithm (RR) algorithms. The comparative analysis demonstrated that, in terms of average waiting time and average turnaround time, the suggested method outperforms the mentioned algorithm.
[11] presented a new CPU scheduling algorithm with varying dynamic time quantum. Two groups were considered in the test and comparison. The first group consists of five processes with burst times in a range of 10-32 and the second group of five processes also in a range of 10-35 with different arrival times. The processes were arranged in ascending order according to their burst times. The algorithm was compared to the traditional round robin (RR), AN, and IRRVQ. The comparison demonstrated that the suggested RRDTQQ algorithm achieved a lower average waiting time, a lower turnaround time, and a smaller number of context switches.
[12] Described a novel CPU scheduling strategy called An Improved Time Varying Round Robin Algorithm (ITVRR). The method outperformed the traditional RR algorithm. The algorithm was tested on one set of seven processes, with burst times in a range of 7–58-time units, an average burst time of 8.5-time units, and arrival times in a range of 0–6. The algorithm handled the processes in the order of arriving at the ready queue. For CPU scheduling, the ITVRR algorithm was compared against the FCFS, SJF, RR, and RMRR algorithms. According to the results, ITVRR outperformed RR and RMRR algorithms in terms of AVT and CS, although RMRR algorithms only exceeded the suggested model in terms of AWT. Additionally, ITVRR outperformed FCFS in terms of AVT.

In [13], a CPU scheduling algorithm called ADRR Scheduling is suggested. The dynamism of time quantum and many rounds, which produced optimal waiting times and numbers of context switches, were some of the key characteristics of ADRR. The comparison is made by applying four examples, each of them on five processes. The burst times of the processes were in the range of 5-22, 10-60-, 17-50-, 4-10-, and 5–35-time units. Other well-known scheduling techniques were compared to the algorithm. The findings demonstrate that the suggested ADRR algorithm outperformed competing algorithms in terms of decreased turnaround times, fewer context switches, and decreased average waiting times.

[4] presented a new Median-Average Round Robin (MARR) scheduling algorithm. The algorithm was tested on three sets of processes. The first included 4 processes with execution times in the range of 6-80, the second included 8 processes in a range of 10-200, and the third included 6 processes in a range of 12-140 with fixed arrival times in each of the three groups. The algorithm adopted the method of entering operations into the ready queue in the form of successive groups. A group is not entered until after the completion of the group before it. The algorithm adopted arranging operations in ascending order according to the burst time in the ready queue. An average turnaround time and average waiting time are decreased using the suggested algorithm.

[1] presented A novel intelligent RR Algorithm. For a large number of processes, clustering the ready queue and building sub-ready queues based on optimal threshold values and standard deviation had been suggested. The algorithm was applied for comparison on one set of processes with burst times in a range of 2-300 and with one fixed arrival of zero. The algorithm adopted the ascending order of processes according to their burst times. Experimental results demonstrated that the suggested algorithm performed better in terms of average waiting time (AWT), average turnaround time (AWT), and context switches (NCS).

Software Testing

3.1 Software testing levels.

The difference between levels of testing is illustrated in Table 1.

Table 1. Software testing levels compared [9]

Criteria	Unit	Integration	System	Acceptance
Purpose	Correct operation of unit/module	proper operation of integrated modules	When coordinated, the entire framework operates admirably. All modules cooperate and work as one.	Programming is operating in accordance with the specified specifics, satisfying the client's expectations
Focus	Smallest testable component	Point of connection and module communication	When coordinated, the entire framework operates admirably. All modules cooperate and work as one.	Programming is operating in accordance with the specified specifics, satisfying the client's expectations
Testing time	When another code is composed	When new parts are added	When the product is finished	When the product is functionally prepared
Performed by	Engineer	Advancement group	Testing group	Improvement group and End-clients
Testing techniques	Generally Whitebox, and Greybox	Whitebox, and Blackbox	Generally Blackbox, and Greybox	Black-box testing
Automation	Automatable with tools like JUnit, PHPUnit, TestNG, etc.	Automatable utilizing Cleanser UI,	Automatable utilizing Webdriver	Automatable utilizing Cucumber
Scaffolding	Complex (require drivers and additionally nails)	Rest Client, and so on.	No drivers/nails required	No drivers/hits required

2.2 Software Testing Teqniques

These are a change of procedures that are utilized in looking at test program code to make positive it proceeds true to form. Testing procedures determine the technique utilized in creating take a show up at cases for leading the endeavoring out and in examining test results[10] Even as creating test protection plan (since thorough endeavoring out is currently at this point not feasible) to accomplish seriously astounding testing. They help distinguish test limitations that are in any unique case hard to perceive. There are really a couple of making an endeavor out techniques with every single strategy veiling exceptional components of the product program application to uncover its quality. Using all the endeavoring out strategies in making an endeavor out a given programming program programming is as of now not conceivable, On the other hand, the analyzer can select and utilize more than one strategy depending on the looking at necessities, test program code type, spending plan, and time imperative. The higher the level of trying out strategies joined, the more noticeable the making an undertaking out result, incorporation, and marvelous There are three fundamental checking frameworks out [11]: White-box, Black-box, and Dark box testing.

3.2.1 White-box testing.

This test looking for approach wherein within shape and execution of the testing program being inspected are perceived by the analyzer. In white-box testing, full data of supply code is expected because the reality check occasions goal is grounded on the execution of the product program substance; inside perspective on the machine and analyzer's abilities to customize are utilized to chart investigate cases [12]. Analyzer chooses contributions to exercise programming ways and assesses the result with the anticipated result. White-box giving a shot is moreover known as Primary, Straightforward Box, Glass Box, Clear Box, Rationale Driven, and Open Box Testing. White-box testing, however ordinarily cultivated at the unit level, is furthermore done the coordination and device phases of the product program evaluating framework. Some white-box evaluating sorts include: Control Stream, Information stream, Branch, Circle, and Way Testing [13]. Table (2) shows the advantages and disadvantages of white-box testing.

Table 2. Advantages and disadvantages of white-box testing

Advantages	Disadvantages
Code advancement can be performed	Specific apparatuses are required, for example, investigating instruments and code analyzers.
Simple to recognize information and cover more experiments because of the analyzer's information on the code.	It's frequently costly and hard to keep up with
Blunders in secret codes are uncovered	Difficult to track down and test all the secret blunders and manage them without leaving time

3.2.2 Black Box testing.

This is a method of software testing where the tester is not aware of the inner workings or implementation of the software being tested. Even while it is often functional, it can be purposeful (like integration testing) or non-functional (like overall performance testing). The creation of test instances follows requirement specifications. Black-box testing places a focus on assessing fundamental aspects of software programs through the use of in-depth test cases and, generally, on maintaining the integrity of external data. [14]. The tester confirms correct input and output production for a given check case in contrast to the check oracle. Despite being primarily focused on System testing and Integration testing, this testing can be applied at all levels of software testing procedures, including Unit, Integration, System, and Acceptance Testing levels. Additionally, black-box testing is also referred to as practical, detail-based, close-box, conduct, and information result testing. Equality Dividing, Cause-Impact Chart, Fluffing, Limit Worth Examination, Choice Table, State Change, Symmetrical Cluster, and All Pair Testing are some examples of Black-box evaluation sorts [14]. Table (3) shows the advantages and disadvantages of black-box testing

Table 3. Advantages and disadvantages of black-box testing

Advantages	Disadvantages
Code information isn't needed, the analyzer's insight is exceptionally straightforward	Restricted inclusion, scarcely any test situations are planned/performed.
The client's and engineer's views are plainly discrete	A few pieces of the backend are not tried by any means.
Admittance to code is unrequired, and speedier experiment advancement	Wasteful testing because of the restricted information on code by an analyzer.
Effective and appropriate for enormous pieces of code	Experiments are challenging to plan without clear particular

3.2.3 Grey Box testing

Grey-box (clear) looking at an approach that adopts the straightforward strategy of black-box looking at and joins it with the code-designated structures in white-box testing. Some comprehension of within working of the product program is required (for the most part of the stage to be tried) in planning evaluations at the black-box level. More handle of the internals of programming program is expected in dark box looking at than in black-box testing, whatever amount of less as opposed to white holder giving a shot. Dark field giving a shot is significantly more prominent fine in reconciliation testing and is the phenomenal technique for deliberate or regional testing, furthermore an ideal suit for Electronic purposes. Some dark box evaluating sorts include: Symmetrical Cluster, Relapse, Example, and Network Testing as shown in Table (4).

Table 4. Advantages and disadvantages of grey-box testing

Advantages	Disadvantages
Gives consolidated advantages of both white-box and black-box testing	Complete white-box testing isn't possible due to incest
Can deal with the plan of intricate test situations all the more cleverly	0sible source code/parallels
Keep up with the limit between autonomous analyzers and engineers	Deformity affiliation is troublesome in appropriated frameworks.

3.3 Some common White- Box Testing Types.

3.3.1 Control-flow testing.

Control float taking a gander at is a kind of white-encase assessment which supervises float plan Control Flow Graph (CFG) ways, centers, and restrictions are picked, examine events are created for executing these ways, and every way, center or clarifications are crossed in some action when to explore the bank of control and pick the solicitation for execution. By looking at the control structure, the analyzer can choose and arrange research models [12][8]. Normally, a check case is an entire course from passage to leave hubs of the CFG. The picked set of ways is utilized to get a definite confirmation of looking at carefulness. Control-stream giving a shot is generally pertinent to new programming program for unit giving a shot [12]. Table (5) shows the Advantages and Disadvantages of Control-Flow Testing.

Table 5. Advantages and disadvantages of control-flow testing

Advantages	Disadvantages
Gets half of all bugs discovered during unit testing	cannot notice decision errors as well as communication errors and misunderstandings at the point of contact
Extremely powerful testing strategy for code that follows unstructured programming	Can't get all the introduction botches
Empower experienced analyzers to sidestep drawing CFG by doing way determination on the source	Tedious

3.3.2 Data Flow Testing.

Unlike the check oracle, the tester verifies that the input and output production for a specific check case is correct. All levels of software testing procedures, including Unit, Integration, System, and Acceptance Testing levels, can use this testing even though it is primarily focused on System and Integration testing. [2]:

- Consideration for all definitions (Advanced): Has a progression from each term to about one use of that definition.
- Consideration for all uses (AU): There is roughly one heading from a variable's definition to its use for each use of the variable.
- There is a direction from everything about definition to everything about cause for each and every c-use (ACU) incorporation. Any shown variable that doesn't have a c-use after that is removed from the inquiry.
- Each and every inclusion of c-uses and certain p-uses: There are courses available for each factor, covering everything from definition through c-use. P-use is defined as any portrayed variable that cannot be understood by c-use.
- All-p-uses (APU) incorporation: There is a course for each and every variable, covering everything from definitions to p-use. Any depicted variable without a corresponding p-use is removed from the analysis. APU+C inclusion: For each variable, there is a relationship between everything pertaining to definition and everything pertaining to p-use. Any depicted variable that doesn't have a p-utilize after it is considered to be in c-use.
- (ADUP) inclusion: For each pair of definitions and uses, all connections between the two should be covered. Given that it is a superset of all distinct data float evaluating techniques, it is the most reliable information stream analysis technique. Additionally, this strategy calls for the best variety of testing methods. Table (6) shows the Advantages and Disadvantages of DF Testing.

Table 6. Advantages and disadvantages of df testing

Advantage	Disadvantage
Can characterize go-between Control stream examination models between all-hubs and all-ways testing	Unscalable Information Stream Investigation calculation for enormous true projects
Handles variable definition and use	Experiment plan hardships contrasted and control stream testing.
It traverses the hole between all ways and branch testing	Unrealistic test objectives that could result in endless testing time wasting
Recognize numerous variable statements	Can have an endless number of ways because of circles

3.4 Comparison of software testing techniques

There isn't a certain approach that works better, but depending on how needs and requirements are assessed, one approach may have a few benefits over others and bad habits. Investigating and combining multiple inspection techniques while inspecting any product aids in removing more flaws, improving the program's overall quality rather than sticking to a single

method. The workspace below displays connections between the three mentioned approaches to using particular norms As shown in Table (7).

Table 7. Comparison of testing techniques

Criteria	White-box	Black-box	Grey-box
Required knowledge	Full information on the inner working of the product.	Information on the interior working of programming isn't needed.	Restricted information on the interior activities of the product.
Performed by	Normally	End-clients, designers, and analyzers	End-clients, designers, and analyzers
Testing focus	Inner operations, coding design, and stream of information and control.	Assessing	Significant level information base graphs and information stream charts.
Granularity	High	parts of the product	Medium
Time consumption	Extremely comprehensive and tedious	Low	Halfway tedious and comprehensive.
Data domain Testing	Information spaces and inside limits can be better tried.	Thorough and the least tedious.	Should be possible on recognized Information areas and inside limits
Algorithm testing	Appropriate	Can be performed through an experimentation strategy.	Unseemly
Also known as	Straightforward box, Open-box, Rationale driven, or code-based testing.	Unacceptable	Clear testing

3. Data flow testing overview

Control flow testing has been improved with Data Flow Testing. DFT and Data Flow Diagram (DFD) diagrams are unrelated. It is centered on keeping track of the variables and how they affect how the program runs. It looks at the life cycle of a variable, including how it is defined, how it is used in the program, how it is computed, and how it is destroyed. DFT is a static and dynamic process in and of itself [15].

Static DFT involves inspecting the program rather than running the code. When the code is studied, it can be seen where a variable is defined in a program and where it is used later on. It is an anomaly if the variable is used by the program after it has been terminated [15].

Three fundamental phases make up DF testing: DF analysis, test data production, and coverage tracking, as shown in Figure (1). These phases take up a significant portion of the research work [16].

- A DF analysis algorithm uses the program P under test as input to determine the test objectives during the DF analysis phase (i.e., def-use pairs).
- The phase of creating test data. To create a test input t that satisfies a target def-use pair du, a testing strategy is used.
- The phase of coverage tracking. For covering the pair du, the test input t is run against the program P. T is added to the test suite if du is covered and not redefined.

Until all pairs are satisfied or the testing budgets (such as testing time) are used up, the entire testing procedure is carried out. In order to verify that the created test suite T is correct using test oracles, it will finally be replayed against against program P.

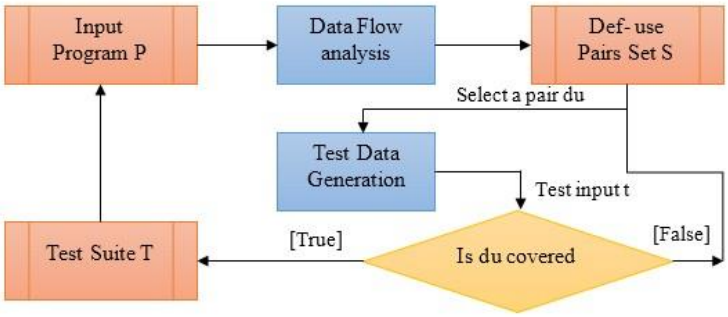


Figure 1. DF testing process [17]

Although DFT is capable of detecting DF defects, a number of obstacles prohibit it from being widely used in industrial practice [17].

- Analyzing DF Inefficiently. In DFT, a DF analysis technique is required to find def-use pairs in the program being tested. However, scaling a DF analysis method to large real-world programs is difficult, especially when all program aspects are taken into account (e.g., aliases, arrays, structs, and class objects). To balance precision and scalability, a reasonable approximation must be used.
- DF Test Design Complexity. In a program, there are many more test objectives for DF criteria than for straightforward control-flow criteria. A DF test case must also cover a variable definition and all of its related usage without requiring any variable redefinitions, which takes more time and effort than simply testing a statement or branch.
- Unrealistic test goals. Def-use pairs may have impractical test targets because static DF analysis approaches are conservative when used to define test targets. If there is an execution path that can pass through a pair, it is viable. If not, it is impossible to apply. The test approach may waste a lot of time covering a non-viable pair if it is not known in advance whether the target pair is feasible or not.

In this case, it is impossible to determine which test objectives are infeasible, and no method can provide a reliable answer to that question. It also appears in structural testing, therefore it is not specific to DFT. Despite the aforementioned limitations, DFT can be automated and these problems can be reduced with the use of current methodologies and recent advancements, as this survey will show.

4. Improving DFT through contract design

A technique for creating high-quality software is called Design by Contract (DBC). This technique makes use of contract specification, which entails the pre- and post-conditions of class method calls as well as class invariants in a class's design. This methodology combines DFT with design by contract to create a testing method known as "DFT Design By Contract." This method generates a class flow graph from the contract specification and then uses traditional DFT to identify test cases [1].

A technique in an experiment is possibly performed if its precondition and class invariant are fulfilled; if they are not, it brings about an unimaginable succession. This is conceivable on the grounds that agreement particulars are executable. The system has additionally been built incorrectly if the postcondition isn't fulfilled after it has been finished. Coming up next are the parts of an agreement particular [15]:

- Precondition of method: It outlines the requirements that must be completed before a class method may be executed. The class's methods each have a prerequisite.
- The postcondition of a method specifies the requirements that must hold after the class method has been executed. The class's methods each have a postcondition.
- Class Invariant: A class's invariant conditions are those that must apply to each and every item in the class. If an invariant was satisfied prior to the method's execution, it must also be satisfied following the method's completion. An invariant constraint verifies an object's consistency in a state across the course of its life. After each class object is created, the invariant must be true.

Contract-based DF testing: A class implementation is tested in relation to the class's contract definition. The method entails the following steps [18]:

- A class flow graph is produced from the contract definition of a class created using DBC. The contract specification outlines the prerequisites and requirements for the class's methods as well as its invariants.
- Definitions and uses (p-uses and c-uses) of each data member of the class are established according to the contract specification, and impossible du pairings are eliminated.
- Through the use of traditional DFT criteria, test cases are generated from CFG.

A coordinated diagram G=(N,E) with N being a bunch of hubs is known as a (CFG). Hub addresses the class' strategies. There are edges in E. Edges interface the technique hubs. The class agreement particularly decides if there is an edge between two technique hubs. On the off chance that the post states of a technique hub, suppose M1, fulfill the preconditions of M2, and the class invariant is valid, then an edge is attracted from M1 to M2. As per the accompanying standard, each datum individual from a class is either characterized, computationally used (c-u), or predicate utilized (p-use). Rule: Expect that d is an individual from its information class and that G is a CFG [19].

- If a value is assigned to d by this method, then d is defined at the method nodded is said c-used at the method node if this method references d.
- If d is used in a method condition, it is said to have been "p-used" at the method node.

In order to evaluate the class DFT criteria, a technique for class testing that uses a design by contract and DFT has been devised. Although there are alternative class testing methods that employ flow diagrams and DFT criteria to produce test cases, this method differs from those methods in the following ways:

- It makes makes use of a flow graph that is built from a contract-based definition.
- Impossible sequences are removed.
- Errors at the implementation level can be detected.

The image illustrates the main elements of DFC and how they interact with the Eclipse environment. The module knowledge base analyzes the Java source code (SRC), which is the input for DFC, and generates a list of classes and the corresponding methods. The tester notes on the list which methods use or modify the object state. The module instrumentation provides extra instructions to find dataflow coverage. Additionally, it generates a DUG that includes details on each node's control flow, variable definitions, and usage. DUG is the input used to create the graph, view the module, and determine every pair of def-u.

So, the conclusion is that opportunities to locate problems that may not be discovered by black-box testing are available by allowing DFT of Java classes. Other tools, such as JUnit, EcEmma, or TPTP, are available in the Eclipse environment for testing Java programs using various methodologies. Def-uses or all-uses coverage criteria, which also ensure instruction coverage, can be achieved in DFC tester designs tests for.

5. Conclusion

With the help of data flow testing tools, users can quickly and efficiently test any component code built in any programming language. With these tools, white box testing can be performed successfully and defects can be found in the product. The automated DFT tool is made to completely reduce physical labor and time requirements and is effectively expandable to make new programming test strategies. Programming testing is tedious, expensive, and a significant part of the cost of enhancing a product framework. Assuming that the test system can be automated, this will completely reduce the cost of programming creation.

6. Acknowledgments

The authors would like to thank the University of Mosul / College of Computer Science and Mathematics for their facilities, which have helped to enhance the quality of this work.

7. References

[1] D. S. N, S. D. S, D. Vijayasree, N. S. Roopa, and A. Arun, “A Review on the Process of Automated Software Testing,” no. September, 2022, doi: 10.48550/arXiv.2209.03069.

[2] L. Mei, W. K. Chan, and T. H. Tse, “Data flow testing of service-oriented workflow applications,” *Proc. - Int. Conf. Softw. Eng.*, no. June, pp. 371–380, 2008, doi: 10.1145/1368088.1368139.

[3] D. S. N;, S. D. S;, D. Vijayasree;, N. S. Roopa;, and A. Arun;, "A Review on the Process of Automated Software Testing," 2022.

[4] P. Tonella and F. Ricca, "A 2-layer model for the white-box testing of web applications," *Proc. - Sixth IEEE Int. Work. Web Site Evol. WSE 2004*, pp. 11–19, 2004, doi: 10.1109/WSE.2004.10012.

[5] N. Mansour and M. Houri, "Testing web applications," *Inf. Softw. Technol.*, vol. 48, no. 1, pp. 31–42, 2006, doi: 10.1016/j.infsof.2005.02.007.

[6] H. K. Dubey, P. Kumar, R. Singh, S. K. Yadav, and R. S. Yadav, "Automated data flow testing," *2012 Students Conf. Eng. Syst. SCES 2012*, 2012, doi: 10.1109/SCES.2012.6199072.

[7] I. Elgendy, M. R. Girgis, and and A. A. Sewisy, "An Automated Tool for Data Flow Testing of ASP.NET Web Applications," *AppliedMathematics Inf. Sci. An Int. J.*, vol. 14, no. 4, pp. 679–691, 2020.

[8] S. Winzinger and G. Wirtz, "Data Flow Testing of Serverless Functions," *Int. Conf. Cloud Comput. Serv. Sci. CLOSER - Proc.*, vol. 2021-April, pp. 56–64, 2021, doi: 10.5220/0010439600560064.

[9] M. Albarka Umar, "Comprehensive study of software testing: Categories, levels, techniques, and types Software Testing View project System Analysis View project Comprehensive Study of Software Testing: Categories, Levels, Techniques, and Types," *ResearchGate*, pp. 1–15, 2020,

[10] Q. Control, "Quality Assurance , Quality Control and Testing — the Basics of Software Quality Management".

[11] V. Khajuria, N. Taneja, and F. Detection, "Software Testing : A Error Finding Technique Software Testing : A Error Finding Technique," *Univers. Rev.*, no. November, 2018.

[12] S. Nidhra and J. Dondeti, "BLACK BOX AND WHITE BOX TESTING TECHNIQUES - A LITREATURE REVIEW," *Int. J. Embed. Syst. Appl. Vol.2, No.2*, vol. 2, no. 2, pp. 29–50, 2012.

[13] S. Nidhra, "Black Box and White Box Testing Techniques - A Literature Review," *Int. J. Embed. Syst. Appl.*, vol. 2, no. 2, pp. 29–50, 2012, doi: 10.5121/ijesa.2012.2204.

[14] S. Nidhra, "Black Box and White Box Testing Techniques - A Literature Review B LACK BOX AND W HITE B OX T ESTING T ECHNIQUES – A L ITERATURE R EVIEW," *Int. J. Embed. Syst. Appl.*, no. April, 2016, doi: 10.5121/ijesa.2012.2204.

[15] S. Sarraf and F. Deebahasan, "A comparative study of data flow testing techniques," *Int. J. Latest Trends Eng. Technol.*, vol. 2, no. 2, pp. 280–285, 2013.

[16] A. Bansal, "A Comparative Study of Software Testing Techniques," *Int. J. Comput. Sci. Mob. Comput. A*, vol. 3, no. 6, pp. 579–584, 2019.

[17] T. Su, K. E. Wu, W. Miao, G. Pu, and J. He, *A Survey on Data-Flow Testing*, vol. 50, no. 1. 2017.

[18] M. Vivanti, "Dynamic data-flow testing," *36th Int. Conf. Softw. Eng. ICSE Companion 2014 - Proc.*, no. November, pp. 682–685, 2014, doi: 10.1145/2591062.2591079.

[19] H. Wu *et al.*, "Peculiar: Smart Contract Vulnerability Detection Based on Crucial Data Flow Graph and Pre-training Techniques," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, vol. 2021-Octob, pp. 378–389, 2021, doi: 10.1109/ISSRE52982.2021.00047.

EDUSJ, Vol, 32, No: 2, 2023 (65-75)

مراجعة تدفق البيانات وادوات الاختبار

عبد الله حازم علي، ندى نعمت سليم

قسم هندسة البرامجيات، كلية علوم الحاسوب والرياضيات، جامعة الموصل، الموصل، العراق

الخلاصة

تسعى هندسة البرامجيات الى التطوير دوما وتحديد العثرات والاخطاء البرمجية قبل نشر المنتج البرمجي، في اختبار البرنامج. يمكن أن تظهر الأخطاء أثناء أي مرحلة من مراحل التطوير أو الاختبار ، حتى بعد طرح المنتج. تصف هذه الورقة منهجيات مختلفة لاختبار تدفق البيانات. نظرًا لأن الاختبار هو عملية تشغيل برنامج بهدف تحديد الأخطاء ، فنحن بحاجة إلى زيادة دقة منطقة التغطية من خلال تضمين عناصر تدفق البيانات استنادًا إلى الأسماء المستعارة وتجنب العناصر غير المفيدة التي تقلل التغطية الإجمالية من أجل زيادة قابلية التطبيق والفعالية من اختبار تدفق البيانات. تبحث هذه الصفحة في اختبار تدفق البيانات، وهو نوع من الاختبار الأساسي (المربع الأبيض). ينقسم اختبار تدفق المعلومات إلى نقطتين رئيسيتين: الخصائص / اختبار الاستخدام ومجموعة من قياسات تضمين الاختبار ؛ وتقسيم البرنامج إلى أجزاء وفقًا لعوامله لجعل اختبار أطر البرمجة أكثر وضوحًا. يصف أيضًا خطوات إجراء اختبار تدفق البيانات وكذلك كيفية تصميم مجموعات اختبار تأخذ الحالات الشاذة في الاعتبار. كما يفحص ويناقش الطرق المستخدمة حتى الآن لإجراء اختبار تدفق البيانات. تشمل هذه الأساليب التصميم المستند إلى العقدة وتغطية تحديد الاتجاه ومقارنة تطبيقات الويب واختبار التحليل.