

Design and Implementation of a Testing Tool (DFT-Tool) Based on Data Flow Specification

Abdullah H. Ali*¹ and Nada N. Saleem²

*^{1&2}Department of Software Science, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq

Abstract:

In software development, testing is a crucial step to ensure the quality and reliability of the final project. Data flow testing is a technique that aims to validate the correct flow and transformation of data through a software system. However, data flow testing can be a complex and time-consuming process, especially for large and complex systems. Data flow testing is an important technique used in software testing to detect potential errors and vulnerabilities in the code. It focuses on how data is processed and moves throughout the code, and can help identify issues such as uninitialized variables, code that is never executed, and variables that are never used. This helps to guarantee that the software behaves as expected, operates correctly, and meets user requirements. Data flow testing can help identify and resolve uninitialized variables, dead code, data dependencies, input validation, and user-entered data. It can also help identify potential issues, such as invalid data or uninitialized variables, and ensure that user-entered data is validated and processed correctly. This can help to improve the quality of software and reduce the number of problems experienced by end-users. The proposed DFT tool has been successful by achieving all planned objectives by multiple stages, first stage is drawing flow graph after translate and analyzing the input code into abstract syntax tree by using parser which built from scratch, abstract syntax tree is the suitable model to draw the graph, then the proposed DFT tool break down the input code into blocks each block contain number statements, second stage the proposed DFT tool make a table to detects transformation for each variables over the each line of the block into: D (definition), U (use may be: prediction or computation), DU pair (variable defined and then used), UD pair (variable used after defining). Third stage of the proposed DFT tool is making another table to detects anomalies like variable which has defined and not used, identify variables into: defined (D), used (U), killed (K) along all lines of the tested code, It also identify the paths of each variable into: clear and not clear, fourth stage is making a table for merge the paths which has duplication of the node and running test cases on the paths which produced after merge process, the proposed DFT tool has so high coverage and accuracy of the tested codes, these accuracy may be reach over 95% depending on the behavior of the tested code.

DOI: [10.24297/j.cims.2023.07.1](https://doi.org/10.24297/j.cims.2023.07.1)

1. Introduction

Overview

In most industrial contexts, software testing is the most common software quality assurance activity. It entails running the program with test inputs and checking the outputs for accuracy in order to look for errors and confirm the behaviour of the application. The thoroughness of the test cases that are run on the application determines how well software testing works. Extensive testing, which involves running the software and examining every possible behaviour, is the ideal way to demonstrate an application's ultimate dependability. Exhaustive testing is not feasible due to the large amount of behaviour that software encodes, thus test engineers must sample the application's input space in order to find a limited number of test cases [1]. Finding a manageably limited number of test cases to execute all the critical actions of the applications is a challenging task [2][3]. The input space of an application can be sampled in numerous (usually infinite) ways [4]. The strengths of data flow software testing tools are that test data flow software provides a comprehensive view of the data flow through a software system, allowing for more comprehensive testing of all possible paths. Many aspects of the testing process can be automated, including test case creation and execution. Testing and reporting, making the process more efficient and reducing the time and costs associated with manual testing. Early detection of problems and testing of data flow software can help prevent defects from becoming more significant and difficult to fix later in the development cycle [4]. It also helps ensure that all possible paths through the software system are tested, resulting in better code coverage and a better understanding of the system as a whole. Also, by identifying and repairing defects in the software system, data flow software testing can help improve the overall reliability of the software and reduce the likelihood of system failure. Finally, integration with other testing techniques: Dataflow software testing can be used in conjunction with other testing techniques, such as unit testing and integration testing, to provide a more complete picture of the software system and help identify defects more effectively [4]. This thesis presents a study on the effectiveness of data flow approaches to the constrained effectiveness of standard data aligned with the flow experiment approach, and proposes a new method based entirely on dynamic evaluation (data flow) to overcome the shortcomings of the cutting-edge approach.

Problem statement

The problem that data flow testing tools aim to address is the identification of defects related to the flow of data within a software program. Data flow defects can arise due to a variety of reasons, such as incorrect variable initialization, memory leaks, data dependencies, and race conditions. These defects can cause the program to behave incorrectly or crash, leading to potential security vulnerabilities or loss of data. Data flow testing is preferred over control flow testing because it examines the data flow with respect to the variables used in the code, while control flow testing focuses on the order of statements in which they will be executed. Manual testing of data flow defects can be time-consuming and error-prone, particularly in large and complex software systems. The creation of the proposed data flow testing tools, helps

developers to identify potential defects early in the development cycle and reduce the time and cost associated with manual testing. This can help ensure that software programs are functioning correctly and are less likely to contain security vulnerabilities or bugs that could cause the program to fail or behave incorrectly, Data flow testing tools automate the process of identifying potential defects related to the flow of data within a program. These tools can analyze the source code of a program and identify potential defects such as null pointer dereferences, uninitialized variables, buffer overflows, and race conditions, Data flow testing helps catch different kinds of anomalies in the code, including variables that are declared but never used within the program or variables that are used but never declared.

2. Background

Data stream software testing can help to solve a variety of issues with software quality and reliability, in terms of identifying defects by examining the data stream through a software system, data stream software testing can help identify defects or problems that are difficult to detect using other testing techniques. Dataflow software testing also helps ensure that all possible paths through the software system are tested, which can lead to better code coverage and a better understanding of the system as a whole. Data flow software testing can help prevent defects from becoming more significant and difficult to fix later in the development cycle. It also gives improved software reliability by identifying and repairing defects in the software system, making the development process more efficient and cost-effective [5], Building test suites, running the program against which it is being tested, and observing the behaviour of the program to assess the health of the program are all steps in the program testing process. Testing each possible input value, or exhaustive testing, allows observing all program behaviours [10], Data flow approaches study how information is created, consumed, and flows from one part of a program to another. Data flow analysis is used in a variety of contexts, including compiler optimization, security analysis, and software testing. Focusing on how they apply to software systems, this chapter defines basic terms and concepts for data flow analysis and software testing [24], Software Testing is Finding and fixing issues to raise quality is the goal of testing. A typical software development budget includes 40% for software testing [18] (See Figure 2.1). The four main goals of software testing are as follows:

- **Demonstration:** It indicates that items are ready for integration or use by demonstrating functionalities under unique circumstances.
- **Detection:** It identifies flaws, mistakes, and shortcomings. It establishes the system's capabilities and constraints, as well as the component and work product quality.
- **Prevention:** It gives advice on how to avoid or minimize errors. Clarify system performance and specifications. Determine how to reduce risk and problems in the future.
- **Improving Quality:** By doing high quality testing, we can limit blunders and consequently enhance the fantastic of software.

Software Testing Techniques There are different testing techniques such as (Black Box Testing, White Box Testing, and Grey Box Testing).

A. Black Box Testing

There is no need to look at the code because black box testing is based on the requirements specifications. Only the tester is aware of the set of inputs and expected results; everything else is done solely from the perspective of the client[19].

B. White Box Testing

White box testing mostly concentrates on the code's internal logic and organization. When a programmer has complete understanding of the program structure, white-box testing is performed. This method makes it possible to test each program branch and choice [20].

C. Grey Box Testing

In general, gray-box testing is successful in combining the advantages of both black-box and white-box testing. The straightforward approach of black-box testing is used in gray-box testing, which additionally makes use of certain restricted application-specific knowledge. Black box plus white box equals grey box [18].

It is a method for testing an application while having a basic understanding of how the system functions and just a limited understanding of how the application is internally structured [21]. As a result, a tester is able to validate both the user interface's output and the steps that led to it. The majority of testing phases can benefit from gray-box testing, but integration testing is where it is most frequently used [22].

Table (1) represent comparison between these three techniques of software testing, Manual testing of data flow defects can be time-consuming and error-prone, particularly in large and complex software systems. The creation of the proposed data flow testing tools, helps developers to identify potential defects early in the development cycle and reduce the time and cost associated with manual testing. This can help ensure that software programs are functioning correctly and are less likely to contain security vulnerabilities or bugs that could cause the program to fail or behave incorrectly, Data flow testing tools automate the process of identifying potential defects related to the flow of data within a program. These tools can analyze the source code of a program and identify potential defects such as null pointer dereferences, uninitialized variables, buffer overflows, and race conditions, Data flow testing helps catch different kinds of anomalies in the code, including variables that are declared but never used within the program or variables that are used but never declared.

Table (1) comparison of three testing techniques forms

No	Black Box	White Box	Grey Box
1	Analyses simply the core components, without understanding how things work inside.	Full understanding of internal processes.	partial understanding of internal processes.

2	It takes the least amount of time and effort.	possibly the most time-consuming and intensive	It lies in the middle of the two.
3	Not appropriate for testing algorithms.	It is appropriate for testing algorithms (suited for all).	Not appropriate for testing algorithms.
4	performed by testers, developers, and end users (user acceptance testing).	Developers and testers carry it out.	Performed by testers, developers, and end users (user acceptance testing).

3. Methodology

The Proposed Data flow testing tool used hand writing parser which to be explain instructions code (if \ for \ while \ break and sequence statement) , by using this parser to build the flow graph and convert it to table of identification parameters into (Def , Use , Killed) , and tracing parameters along all the produced paths and detect which one of these paths clear or not clear , at last execute design test cases automatically generated values .

Use Case Diagrams of the UML are used for declaring and analyzing the dft -tool , An UML conduct or dynamic outline is a utilization case graph. Utilize case charts use entertainers and use cases to address the usefulness of a framework. Use cases are an assortment of errands, contributions, and tasks that framework should deal with. A "framework" is something that is being made or run. The "entertainers" are people or things that do specific jobs for the framework. Figure (1) shows the first level of Use case diagram of the proposed dft-tool.

The figure shows the general view of the DFT- tool , An actor (user) interact with the tool use cases .At first an actor select the code or program to be tested, then the process of analyzing the code will be start and at the end choice the use case for running testing process .

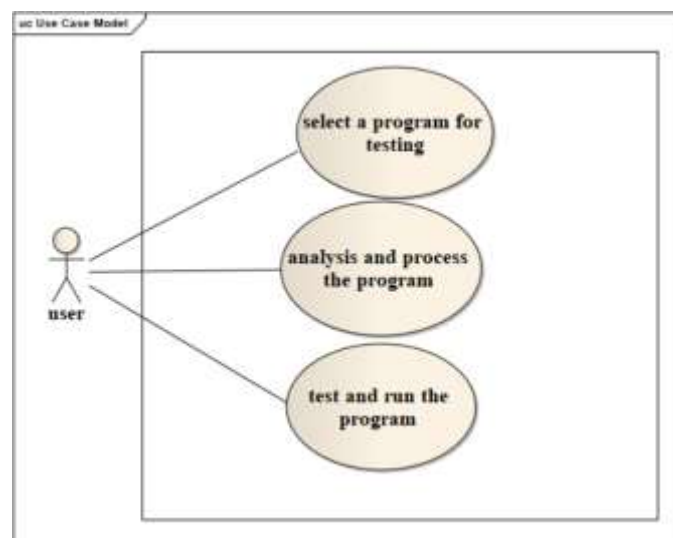


Figure (1): First analysis use case level of DFT-Tool

Figure (2) shows the second level of use case diagram which represent the analysis mechanism of DFT tool working by details, from the beginning point is selecting the tested code or program down to end point which is generate input data for each path.

After analysing the tool, the design phase will explain using activity diagrams, The general flowchart of the proposed DFT-Tool represented in figure (3).

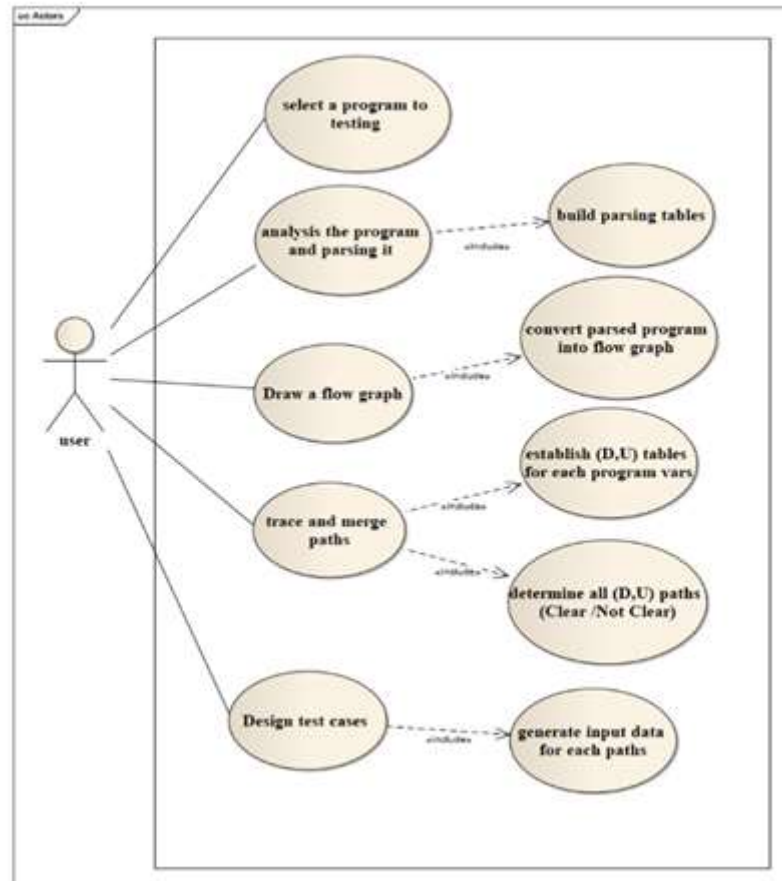


Figure (2): second level of proposed DFT tool

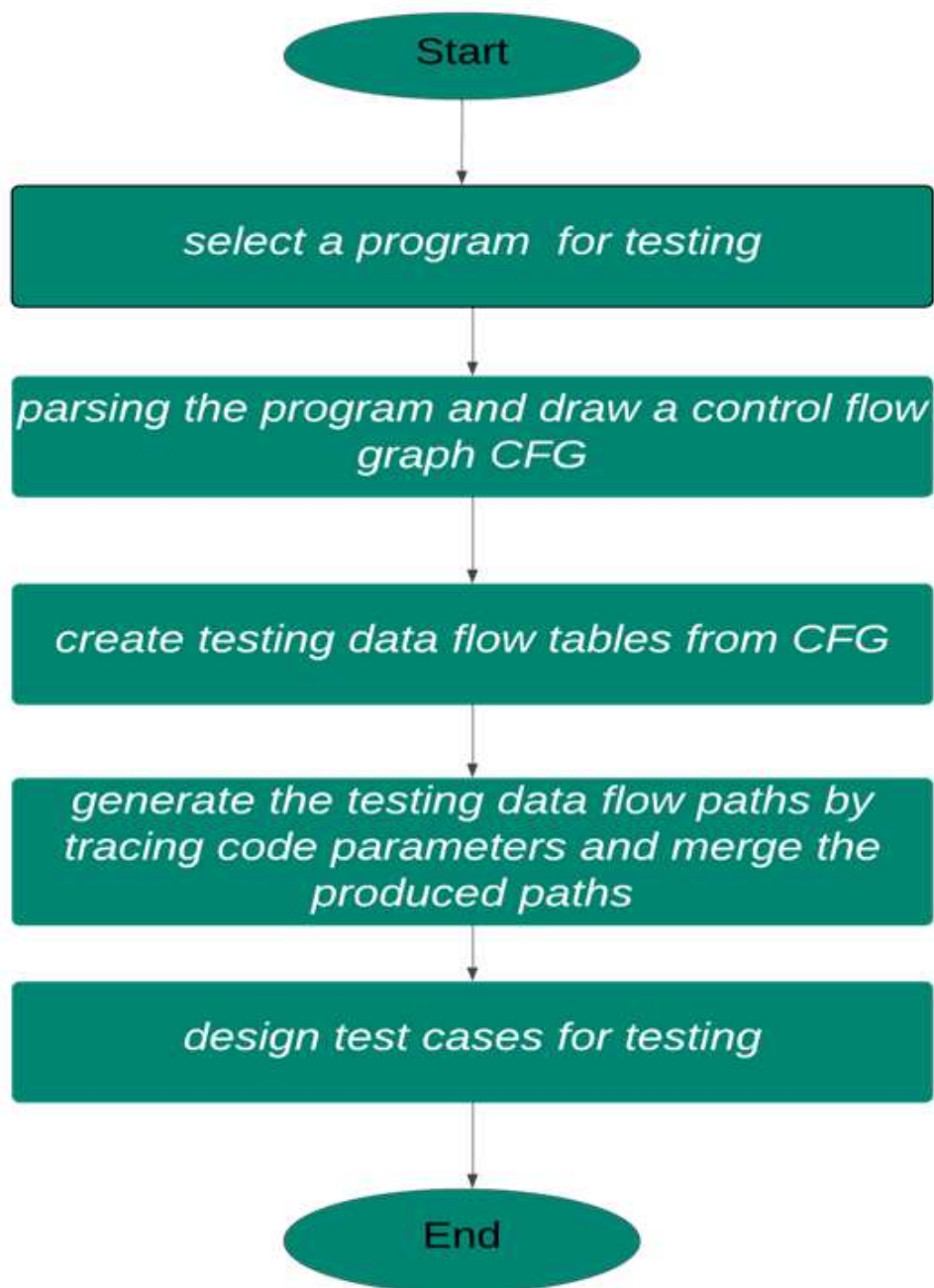


Figure (3): flowchart of DFT-Tool

4. Discussion The Result of DFT- Tool

The proposed DFT- tool has high flexibility , it will receive any code file with (.py) python extension and that mean testing code must be written by python language exclusively and it must be executable individually .after running and executing the proposed DFT –tool by using editor pycharm , for the following explanations we talking about one example as sample code te be test , the main GUI screen is display multiple options used for testing the code and display results as shown in figure (4) .

According to the figure (4), DFT-tool has multiple options to be selected sequentially, these options to be explained now simply and then the researcher will talking about them deeply and these options are: first option is a list of code samples which to be selected to test and which must written by python language, second option is a button its title is draw graph which is responsible of converting the input code into control flow graph consist of nodes and edges, third option is a button its title is establish table of data action which responsible for providing table of divided input code into blocks and the description of each variable into: D (defined) \ U (use) \ DU (defined then used) inside these blocks, fourth option is a button its title tracing and merge paths which responsible for tracing each variable along its paths and identify these variables into: DU pair and killed and identification of paths into: clear and not clear, fifth option is a button its title is design test case which is responsible for apply the values that generated automatically on the variables of the input code.

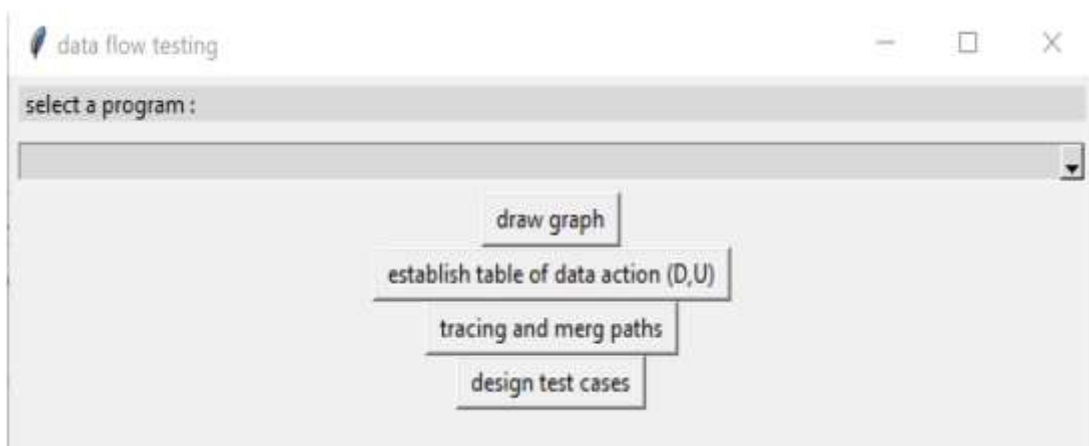


Figure (4): shows the main GUI of proposed DFT-tool

The proposed DFT-tool as shown in figure (5) start apply the first option of the GUI of the proposed DFT- tool which is the list titled (select a program) which refer to selection program to be test and this will be done after putting the files of codes to be tested in the folder (input) of the DFT-tool's folders manually for preparing them to be chosen by the proposed DFT- Tool.

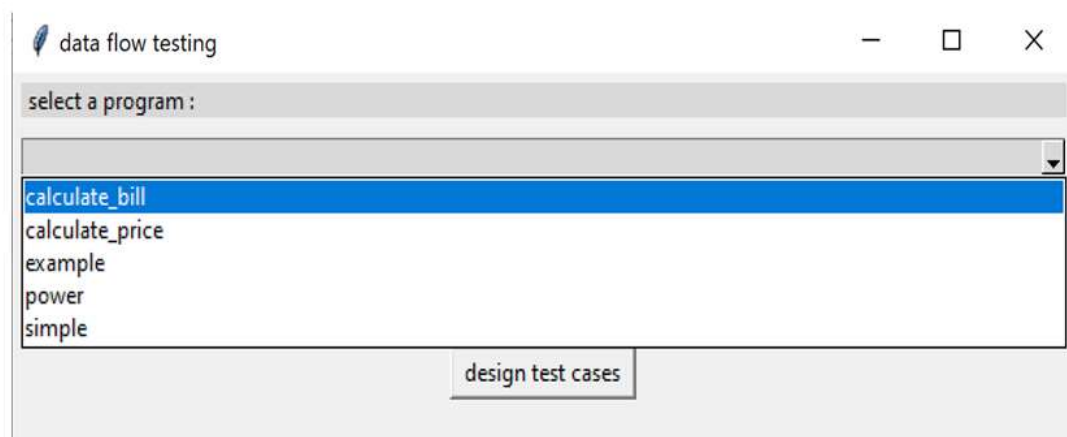


Figure (5): choosing the sample code to be test

The proposed DFT-tool as shown in figure (6) start apply the second option of the GUI of the proposed DFT- tool which consist of multiple items whereas the produced control flow graph CFG represent the input code and it is located at the left of the GUI screen and the main buttons at the center of the GUI screen and image of text input code will be shown under all buttons of the main GUI, after explaining the content of GUI screen of third option, we will continue explaining the role of third option which is the button titled (Draw graph) which responsible for drawing the CFG graph of the selected program and that will be achieved after parsing the input code using (AST python model) by breaking the input code into sequence of tokens and then translate each one of these tokens into nodes and connect these nodes by the edges which connect between each two nodes , this option of GUI was the essential and preparing option and consider the base for the next options of the main GUI .

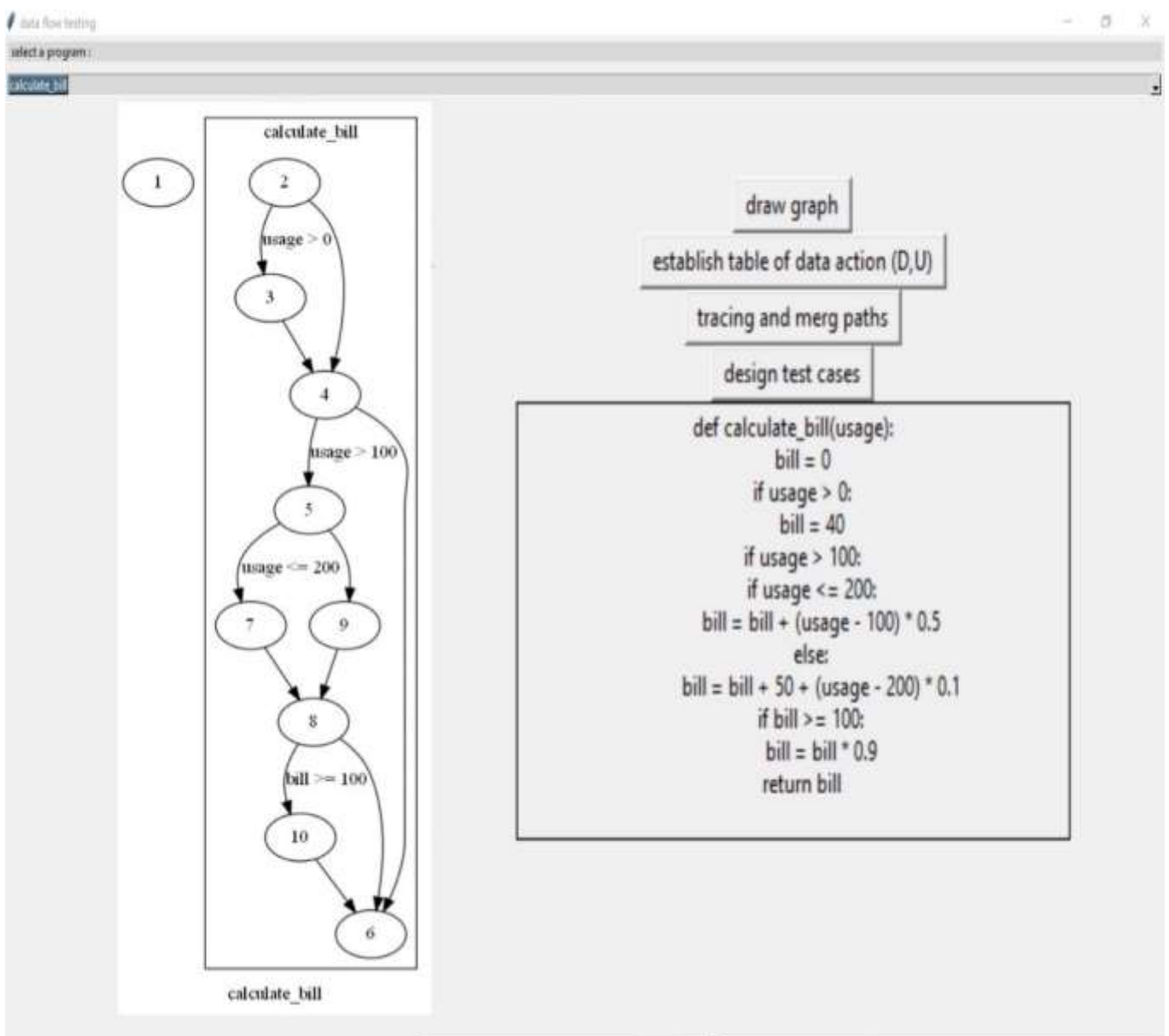



Figure (6) : drawing CFG graph of the testing code

The proposed DFT-tool as shown in figure (7) start apply the third option of the GUI screen of the proposed DFT- tool which consist of a table whereas this table include four column titled by the following headers sequentially :(Block) include block number , (source code) include blocks of the input code after breaking it, (variable name) for each variable include describe the transformations of variables while they exist in the corresponding block into : D (defined) \ U (use) \ UD (used then defined) ,



Block	Source code	usage	bill
2	def calculate_bill(usage): bill = 0 if usage > 0:	D U	D
3	bill = 40		D
4	if usage > 100:	U	
5	if usage <= 200:	U	
7	bill = bill + (usage - 100)	U	UD
9	else: bill = bill + 50 + (usage - 20	U	UD
8	if bill >= 100:		U
10	bill = bill * 0.9		UD
6	return bill		

figure (7): establishment table of data action (D,U) for the testing code

after explaining the content of GUI screen for the fourth option, we will continue explaining the role of fourth option which is the button titled (establish table of data action D, U) whereas the program has been converted into the control flow graph (CFG) at the previous stage , we will see the process of revers translating for data variables from the CFG and extracting it from the graph and tracing and locating it into the table of data action (D,U) for each variable in the code.

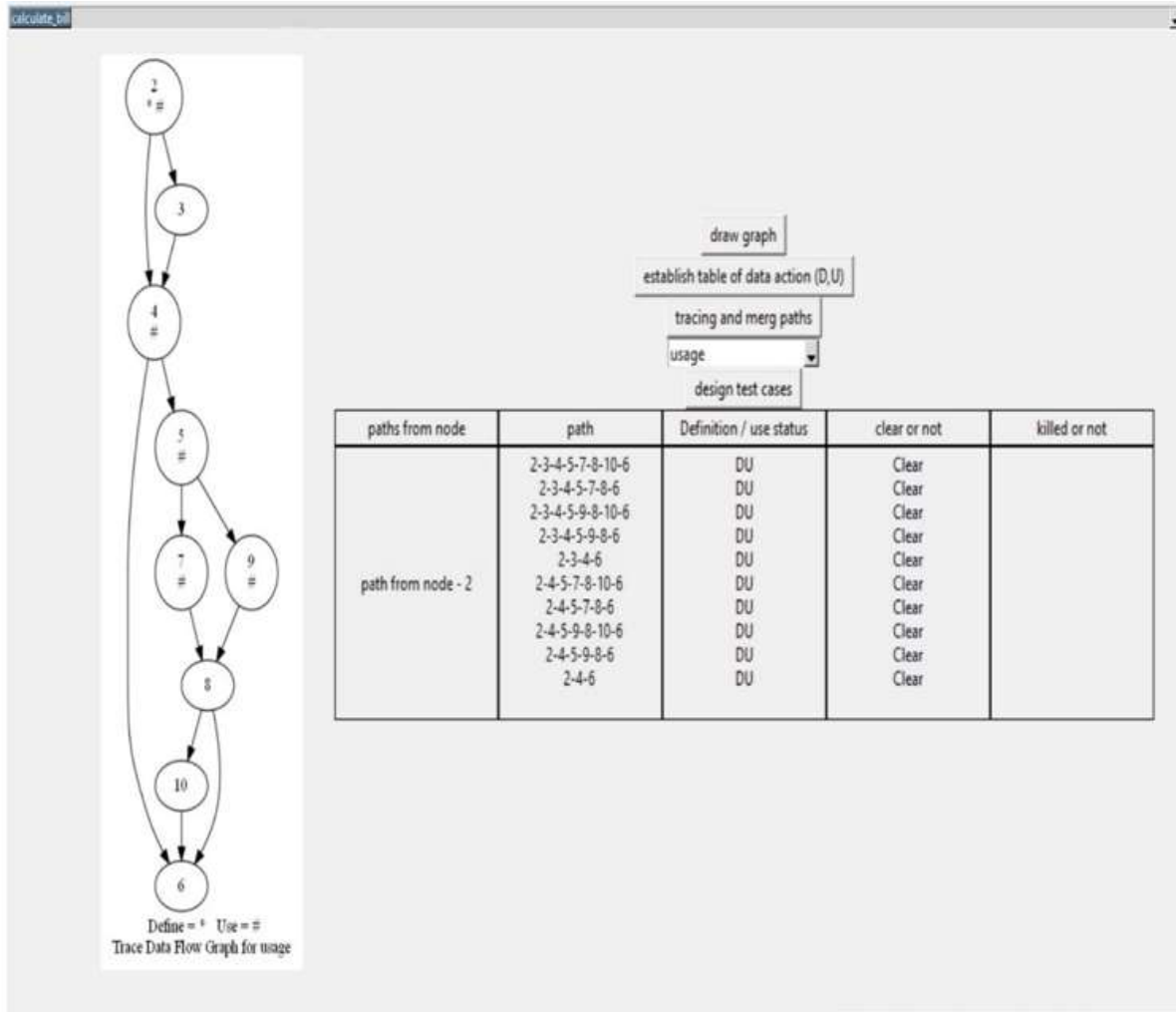


Figure (7): Draw data flow graph and Tracing of parameter usage

The proposed DFT-tool as shown in figures (7) , (8) start apply the fourth option of the GUI of the proposed DFT- tool which consist of multiple items whereas the produced data flow graph DFG represent the behaviour of each variable along the paths of input code and located at the left of the GUI screen and the main buttons at the center of the GUI screen , it also contain of a table that include five columns titled by the following headers sequentially :(path from node) include node number which the tracing of variable and paths generation processes starting from it , path) include produced paths, (Definition/ use status) include description the transformations of each variable while they exist in the corresponding path into : D (defined) \ U (use) \ UD (used then defined) , categorization of each path into : (clear or not) whereas the

clear status include description of the variable inside this path has definition then use and not clear status has double definition without use , ,(killed or not) include identify the variables into killed for variable defined and not used along its path and not for variable that defined and use along its path at least once.

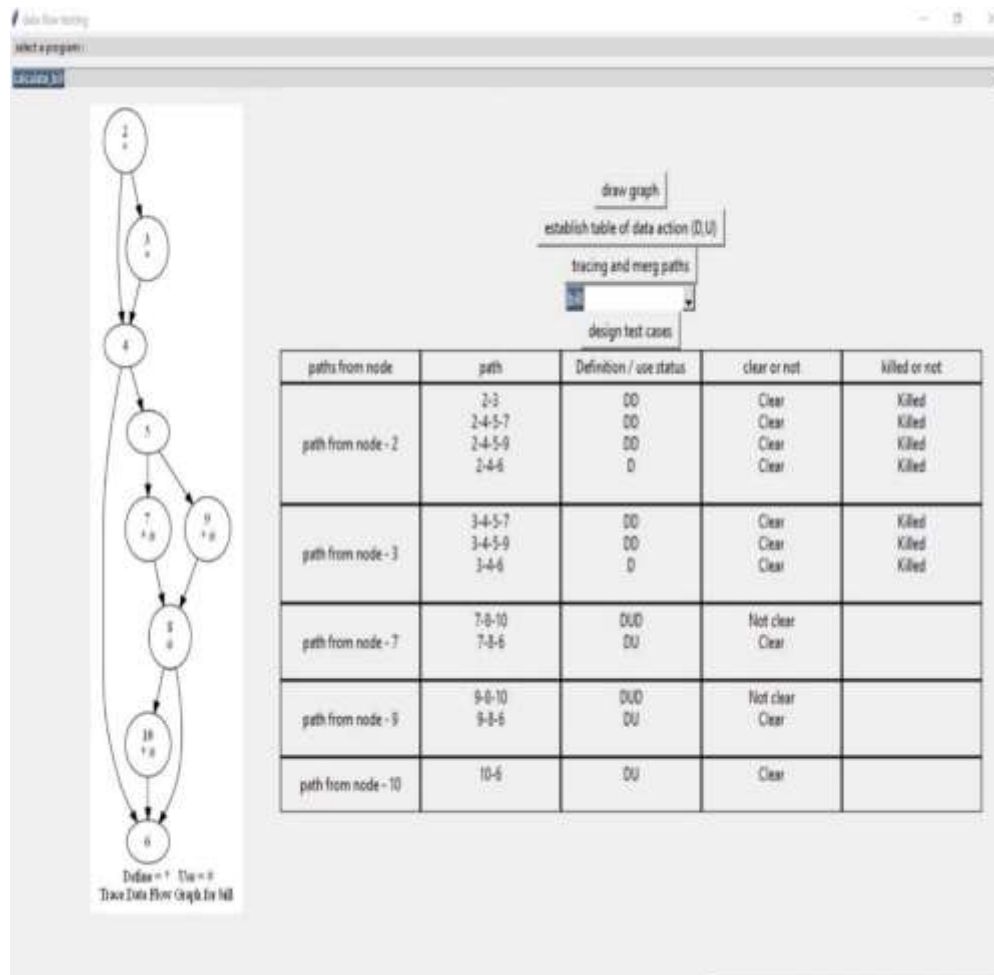


Figure (8): Draw data flow graph Tracing of parameter bill

The proposed DFT-tool as shown in figure (9) start apply the fifth option of the GUI screen of the proposed DFT- tool which consist of a table whereas this table include four columns titled by the following headers sequentially :(Path sequence number) include path number , (Node visited) include nodes which discovered along its merged paths, (input test data usage) include apply the values which to be generated automatically according specific range on each variable of the input code to be tested by the DFT-Tool, (output) include the output results values after applying the test case on the each variable of the input code .

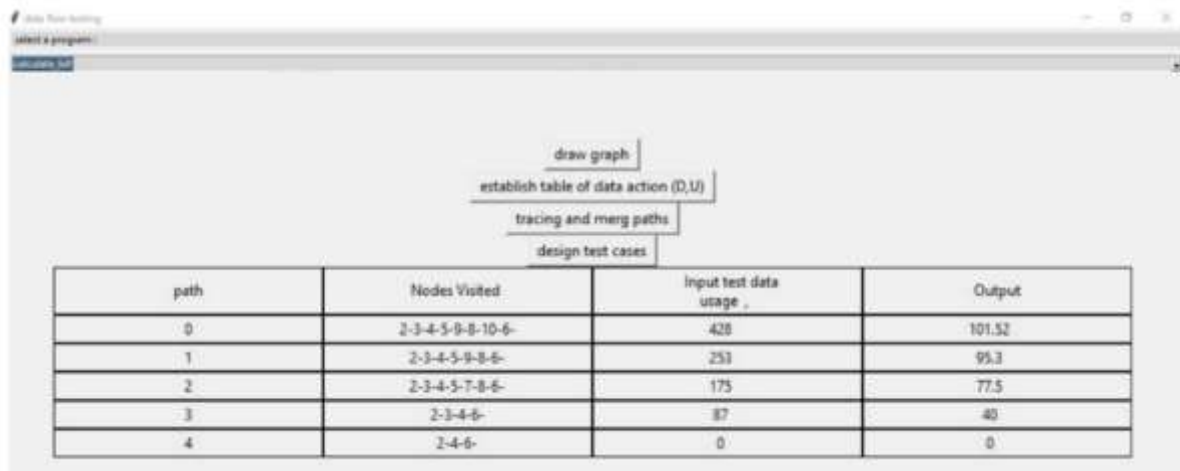


Figure (9):Design Test case

5. Conclusions

Data flow testing is a white-box testing technique that examines the data flow with respect to the variables used in the code. It examines the initialization of variables and checks their values at each instance. The data flow testing tool produces two types of graphs: a data flow graph and a control flow graph. The data flow graph shows the flow of data through the program, while the control flow graph shows the order of statements in which they will be executed. The following steps are involved in tracing code parameters for producing paths and data flow graph for each variable independently, determining if these paths are clear or not, and detecting killed variables along each path as a step of data flow testing tool: Create a control flow graph of the program, Identify the variables and their values in the program, Determine the locations of definitions and uses of variables in the program, Create a data flow graph that shows the flow of data through the program, Trace the variables through the data flow graph to ensure that they are used and defined correctly, Identify the independent variables and put them as column headers in a table, For each independent variable, identify the number of possible values, Test each row in the table, Determine if the paths produced by the data flow testing tool are clear or not, Detect killed variables along each path, Write test cases that ensure complete coverage of the program logic, Test the program using the selected test cases, Analyze the results of the test to identify any issues related to data flow, such as uninitialized variables or dead code, Repeat the process as necessary to ensure the accuracy and completeness of the testing.

References

1. A. Taňkoš, "Dataflow extraction tool for Cobol programming language," pp. 37–42, 2018.
2. M. Young, "Software testing and analysis: process, principles, and techniques," *Choice Rev. Online*, vol. 46, no. 02, pp. 46-0935-46-0935, 2008, doi: 10.5860/choice.46-0935.
3. 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.

4. M. L. Chaim, K. Baral, J. Offutt, M. Concilio, and R. P. A. Araujo, "Efficiently Finding Data Flow Subsumptions," Proc. - 2021 IEEE 14th Int. Conf. Softw. Testing, Verif. Validation, ICST 2021, pp. 94–104, 2021, doi: 10.1109/ICST49551.2021.00021.
5. And E. J. W. Sandra Rapps, "Data Flow Analysis Techniques for Test Data Selection," IEEE, 1982.
6. M. R. Girgis, A. S. Ghiduk, and E. H. Abd-Elkawy, "Automatic Data Flow Test Paths Generation using the Genetical Swarm Optimization Technique," Int. J. Comput. Appl., vol. 116, 2015.
7. A. Bansal, "A Comparative Study of Software Testing Techniques," Int. J. Comput. Sci. Mob. Comput. A, vol. 3, no. 6, pp. 579–584, 2019.
8. M. E. Khan, "Different approaches to white box testing technique for finding errors," Int. J. Softw. Eng. its Appl., vol. 5, no. 3, pp. 1–14, 2011, doi: 10.5121/ijsea.2011.2404.
9. K. Mohd. Ehmer and K. Farneena, "A Comparative Study of White Box , Black Box and Grey Box Testing Techniques," Int. J. Adv. Comput. Sci. Appl., vol. 3, no. 6, pp. 12–15, 2012.
10. M. TanLi, Y. Zhang, Y. Wang, and Y. Jiang, "Grey-box technique of software integration testing based on message," J. Phys. Conf. Ser., vol. 2025, no. 1, 2021, doi: 10.1088/1742-6596/2025/1/012096.
11. K. Sangeetha and P. P. Dalal, "A Review paper on Software Effort Estimation Methods," JETIR, vol. 5, no. 3, pp. 163–169, 2015.
12. D. Almog, D. O. V. B. Sohacheski, M. L. Gillenson, R. Poston, and S. Mark, "THE UNIT TEST: FACING CICD – ARE THEY ELUSIVE DEFINITIONS?," J. Inf. Technol. Manag., vol. XXIX, no. 2, 2018..