FinanceMet – All In one Finance Calculator

Data Flow Coverage Criteria

CS 731: Software Testing

Guided By

Meenakshi D Souza
Professor
International Institute of Information Technology, Bangalore

Under the Assistance of

Mayank Chadha
Teaching Assistant

International Institute of Information Technology, Bangalore

Created by:

Vraj Jatin Naik
MT2023050
Vraj.Naik@iiitb.ac.in

Arjun Gangani
MT2023153
Arjun.Gangani@iiitb.ac.in

ABSTRACT

The project, **FinFlow Tester**, implements **Data Flow Coverage Criteria** to thoroughly test a Java-based terminal application, **FinanceMate**, designed to provide a suite of financial calculators, including EPF, PPF, SIP, SWP, Taxation, Lumpsum, and Gratuity. The testing process employs **Data Flow Graphs (DFGs)** to systematically analyze and validate all possible definition-use (DU) paths of variables in the program. Emphasis is placed on achieving **all-defs** and **all-DU-paths coverage**, particularly in scenarios involving loops and complex control flows.

By leveraging graph-based testing techniques, FinFlow Tester ensures that every variable definition is exercised and its uses verified across all paths. The testing process is automated using the **JUnit framework**, allowing for efficient execution, monitoring, and result analysis. This rigorous approach ensures the reliability, accuracy, and robustness of the application while showcasing the value of data flow testing in delivering error-free, high-quality software solutions.

GitHub Repo Link: https://github.com/VrajNaik/Software-Testing-MiniProject

TABLE OF CONTENTS

LIST OF FIGURES

1. PROBLEM STATEMENT

FinanceMate is a comprehensive Java-based terminal application designed to assist users with financial planning and investment decisions. The program offers a robust suite of calculators tailored to address various financial scenarios, enabling users to make informed choices for their future.

The application provides the following features:

- **Employee Provident Fund (EPF)**
- **Public Provident Fund (PPF)**
- **Systematic Investment Plan (SIP)**
- **Systematic Withdrawal Plan (SWP)**
- **Taxation Calculations**
- **Lumpsum Investment Returns**
- **Gratuity Calculations**
- **Retirement Corpus Calculator**
- **Goal Savings Calculator**
- **Budget Planning Calculator**

By leveraging these functionalities, **FinanceMate** aims to simplify financial planning, ensuring users can efficiently calculate, plan, and track their financial goals in a user-friendly manner. Developed entirely in Java, the application is designed to handle a wide range of financial computations with precision and ease of use.

2. TESTING METHODOLOGY

The testing process for this project is based on Data Flow Graph (DFG) Testing, a graph-based approach that ensures comprehensive coverage of variable definitions and uses within the source code. The methodology revolves around generating DU-paths (Definition-Use paths) for each variable, ensuring that every definition and corresponding usage of variables are thoroughly tested.

2.1 DATA FLOW GRAPH (DFG)

A Data Flow Graph represents the control flow of a function, where each node is labeled with:

Definitions (DEF): Statements where variables are assigned values.

$$\boldsymbol{DEF(S) = \{X \mid \text{statement } S \text{ contains the definition of } X\}}$$

Uses (USE): Statements where these variable values are referenced.

$$\boldsymbol{USE(S) = \{X \mid \text{statement } S \text{ contains the use of } X\}}$$

By analyzing the DFG, we generate test paths (DU-paths) that cover both the points where variables are defined and where they are used.

2.2 DATA FLOW COVERAGE CRITERIA

Two critical testing criteria are employed to ensure thorough analysis:

2.2.1 All-Defs Coverage:

For every definition-path set $\boldsymbol{S = du(n, v)}$ the test suite ($\boldsymbol{TR}$) includes at least one path $\boldsymbol{d}$ in $\boldsymbol{S}$. This ensures that every variable definition is tested at least once.

2.2.2 All-DU-Paths Coverage:

For each definition-use pair set $\boldsymbol{S = du(ni, nj, v)}$, $\boldsymbol{TR}$ includes every possible path $\boldsymbol{d}$ in $\boldsymbol{S}$. This guarantees comprehensive testing of all possible flows from definitions to their respective uses, including paths through loops and complex control structures.

2.3 TEST CASE DESIGN

Test cases are designed to cover every unique DU-path in the program. The process involves:

Identifying all variable definitions (**DEF**) and their corresponding uses (**USE**) in the program.

Generating a DFG for each function and deriving all possible DU-paths.

Writing test cases to exercise each DU-path, ensuring that both all-defs and all-DU-paths coverage criteria are satisfied.

2.4 TOOLS USED FOR TESTING

2.4.1 Data Flow Graph Coverage Web Application:

This tool generates All-Defs Coverage and All-DU-Paths Coverage for the Data Flow Graph of each function. It was instrumental in visualizing and validating the DFGs used for testing.

Tool URL: DFGraphCoverage

2.4.2 JUnit:

A unit testing framework for Java applications, JUnit automates the execution of test cases and verifies their outcomes. It enabled efficient and systematic testing of the DU-paths in the code.

Tool URL: JUnit

By combining these tools with the Data Flow Coverage Criteria, the project achieved rigorous and systematic testing of the source code, ensuring reliability and correctness.

3. TESTING

3.1 EMI CALCULATOR:

Data Flow Graph:

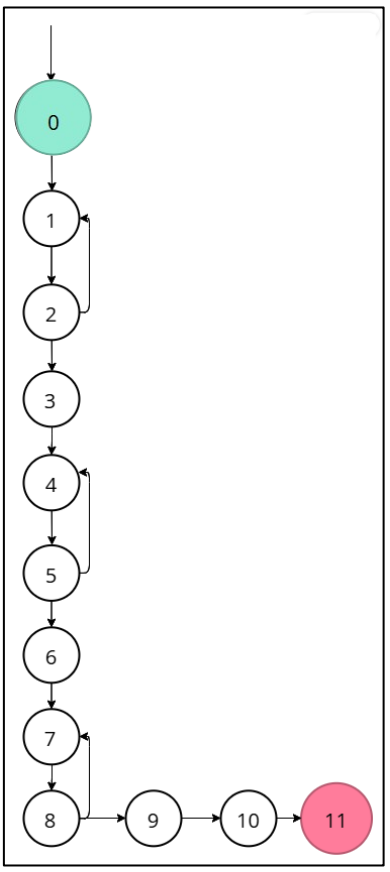

Figure 3-1 EMI Calculator DFG

Variables	Definitions	Uses
val	{2, 5, 8}	{(2, 1), (2, 3), (5, 4), (5, 6), (8, 7), (8, 9)}
loanAmount	{3}	{0}
interestRate	{6}	{10}
loanTenure	{9}	{10}
amount	{10}	{11}

Variables	All DU Path Coverage
val	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11]
loanAmount	[0,1,2,3,4,5,6,7,8,9,10,11]
interestRate	[0,1,2,3,4,5,6,7,8,9,10,11]
loanTenure	[0,1,2,3,4,5,6,7,8,9,10,11]
amount	[0,1,2,3,4,5,6,7,8,9,10,11]

Variables	All Def Coverage
val	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,9,10,11]
loanAmount	[0,1,2,3,4,5,6,7,8,9,10,11]
interestRate	[0,1,2,3,4,5,6,7,8,9,10,11]
loanTenure	[0,1,2,3,4,5,6,7,8,9,10,11]
amount	[0,1,2,3,4,5,6,7,8,9,10,11]

```java
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class EMICalculatorTest {

    String input1 = "1000000\n5.5\n2\n"; // [0,1,2,3,4,5,6,7,8,9,10,11]
    String input2 = "2000000\n-5\n3.5\n2\n"; // [0,1,2,3,4,5,4,5,6,7,8,9,10,11]
    String input3 = "2000000\n3.5\n-2\n2\n"; //  [0,1,2,3,4,5,6,7,8,7,8,9,10,11]
    String input4 = "-10000\n2000000\n3.5\n2\n"; // [0,1,2,1,2,3,4,5,6,7,8,9,10,11]

    public void testing(String input, Long expectedTax){
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);
        EMICalculator emiCalculator = new EMICalculator();
        Long actual = emiCalculator.init();
        Assert.assertEquals(expectedTax,actual);
    }

    @Test
    public void testCase1(){
        testing(input1, 44095L);
    }

    @Test
    public void testCase2(){
        testing(input2, 86405L);
    }
    @Test
    public void testCase3(){
        testing(input3, 86405L);
    }
    @Test
    public void testCase4(){
        testing(input4, 86405L);
    }
}
```

Figure 3-2 EMI Calculator Test Code

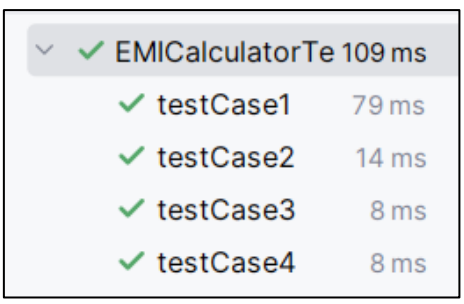

Figure 3-3 EMI Calculator Test Cases

3.2 GRATUITY CALCULATOR

Data Flow Graph:

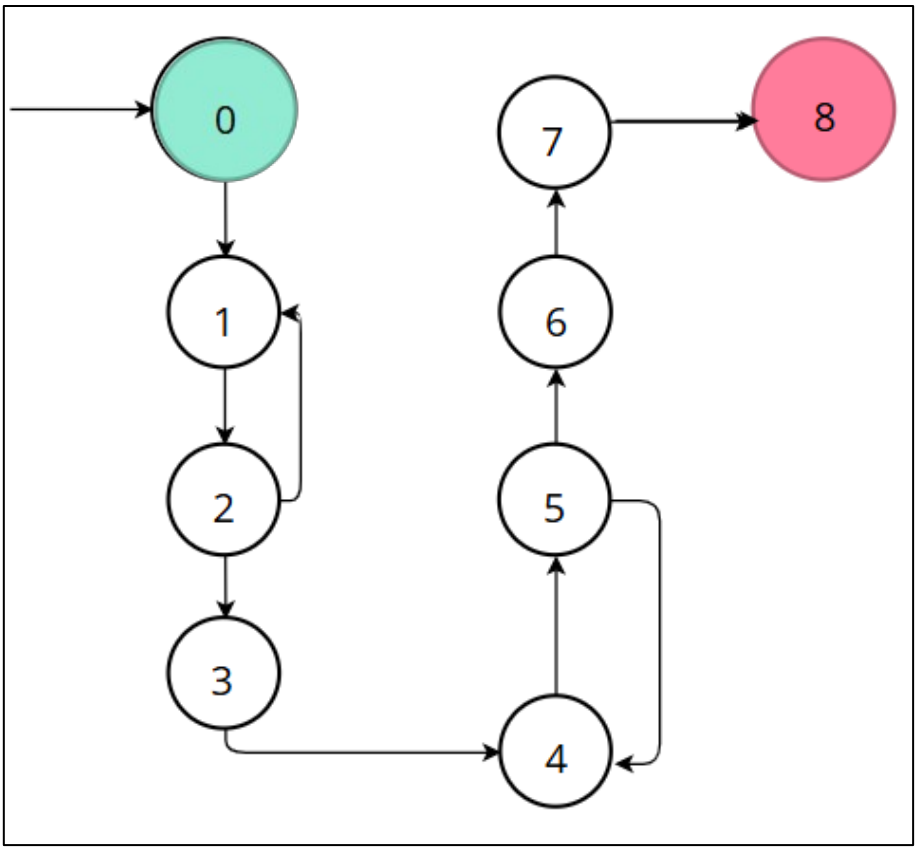

Figure 3-4 GRATUITY Calculator DFG

Variables	Definitions	Uses
ms	{2}	{(2, 1), (2, 3), 3}
monthlySalary	{3}	{7}
yos	{5}	{(5, 4), (5, 6), 6}
yearOfServices	{6}	{7}
amount	{7}	{8}

Variables	All DU Path Coverage
ms	[0,1,2,3,4,5,6,7,8], [0,1,2,1,2,3,4,5,6,7,8]
monthlySalary	[0,1,2,3,4,5,6,7,8]
yos	[0,1,2,3,4,5,6,7,8], [0,1,2,3,4,5,4,5,6,7,8]
yearOfServices	[0,1,2,3,4,5,6,7,8]
amount	[0,1,2,3,4,5,6,7,8]

Variables	All Def Coverage
ms	[0,1,2,3,4,5,6,7,8]
monthlySalary	[0,1,2,3,4,5,6,7,8]
yos	[0,1,2,3,4,5,6,7,8]
yearOfServices	[0,1,2,3,4,5,6,7,8]
amount	[0,1,2,3,4,5,6,7,8]

```java
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class GratuityCalculatorTest {
    String input1 = "3400\n12\n"; // [0,1,2,3,4,5,6,7,8]
    String input2 = "-10000\n2300\n12\n"; // [0,1,2,1,2,3,4,5,6,7,8]
    String input3 = "2300\n-5\n12\n"; // [0,1,2,3,4,5,4,5,6,7,8]

    public void testing(String input, Long expectedTax){
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);
        GratuityCalculator gratuityCalculator = new GratuityCalculator();
        Long actual = gratuityCalculator.init();
        Assert.assertEquals(expectedTax,actual);
    }

    @Test
    public void testCase1(){
        testing(input1, 23538L);
    }

    @Test
    public void testCase2(){
        testing(input2, 15923L);
    }
    @Test
    public void testCase3(){
        testing(input3, 15923L);
    }
}
```

Figure 3-5 GRATUITY Calculator Test Code

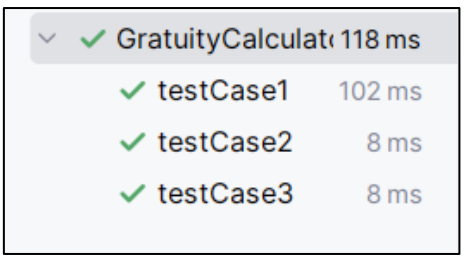

Figure 3-6 GRATUITY Calculator Test Cases

3.3 LUMPSUM CALCULATOR

Data Flow Graph:

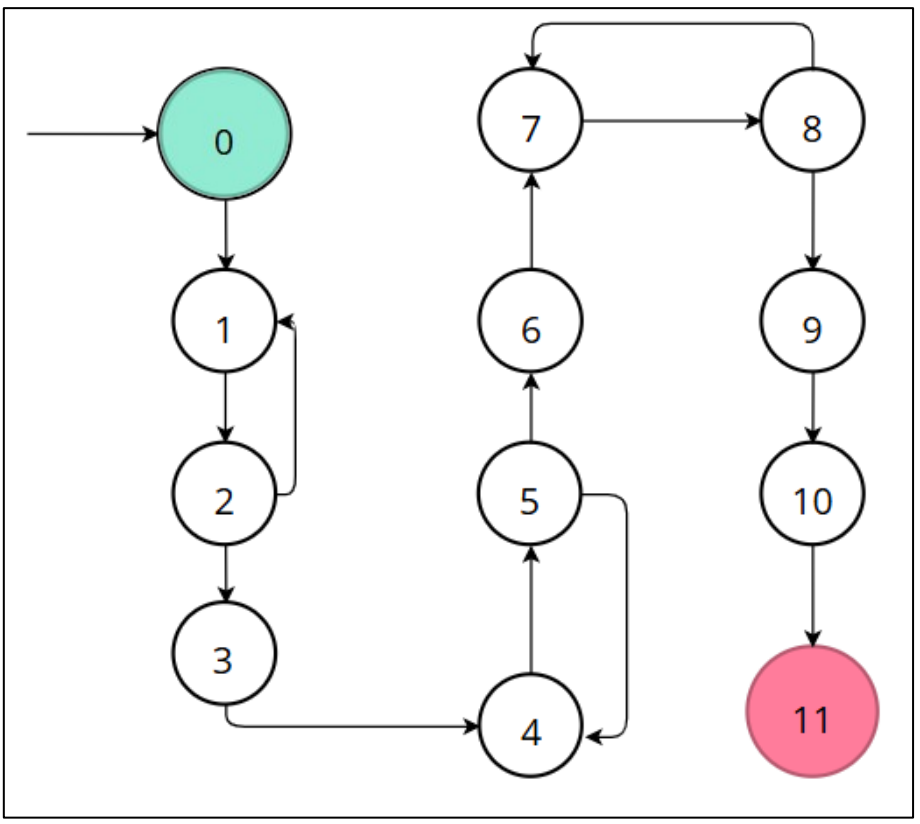

Figure 3-7 LUMPSUM Calculator DFG

Variables	Definitions	Uses
Val	{2, 5, 8}	{(2, 1), (2, 3), (5, 4), (5, 6), (8, 7), 9}
principleAmount	{3}	{10}
interestRate	{6}	{10}
timePeriod	{9}	{10}
amount	{10}	{11}

Variables	All DU Path Coverage
val	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11]
principleAmount	[0,1,2,3,4,5,6,7,8,9,10,11]
interestRate	[0,1,2,3,4,5,6,7,8,9,10,11]
timePeriod	[0,1,2,3,4,5,6,7,8,9,10,11]
amount	[0,1,2,3,4,5,6,7,8,9,10,11]

Variables	All Def Coverage
val	[0,1,2,3,4,5,6,7,8,9,10,11]
principleAmount	[0,1,2,3,4,5,6,7,8,9,10,11]
interestRate	[0,1,2,3,4,5,6,7,8,9,10,11]
timePeriod	[0,1,2,3,4,5,6,7,8,9,10,11]
amount	[0,1,2,3,4,5,6,7,8,9,10,11]

```java
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class LumpsumCalculatorTest {
    String input1 = "1000000\n5.5\n2\n"; // [0,1,2,3,4,5,6,7,8,9,10,11]
    String input2 = "2000000\n-5\n3.5\n2\n"; // [0,1,2,3,4,5,4,5,6,7,8,9,10,11]
    String input3 = "2000000\n3.5\n-2\n2\n"; //  [0,1,2,3,4,5,6,7,8,7,8,9,10,11]
    String input4 = "-10000\n2000000\n3.5\n2\n"; // [0,1,2,1,2,3,4,5,6,7,8,9,10,11]

    public void testing(String input, Long expectedTax){
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);
        LumpsumCalculator lumpsumCalculator = new LumpsumCalculator();
        Long actual = lumpsumCalculator.init();
        Assert.assertEquals(expectedTax,actual);
    }

    @Test
    public void testCase1(){
        testing(input1, 1113025L);
    }

    @Test
    public void testCase2(){
        testing(input2, 2142449L);
    }
    @Test
    public void testCase3(){
        testing(input3, 2142449L);
    }

    @Test
    public void testCase4(){
        testing(input4, 2142449L);
    }
}
```

Figure 3-8 GRATUITY Calculator Test Code

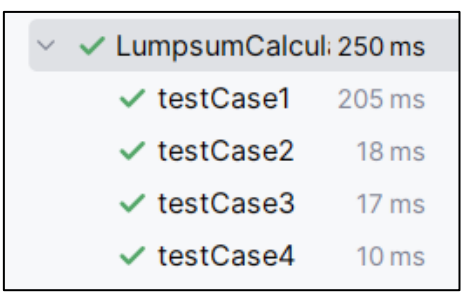

Figure 3-9 GRATUITY Calculator Test Cases

3.4 PPF CALCULATOR

Data Flow Graph:

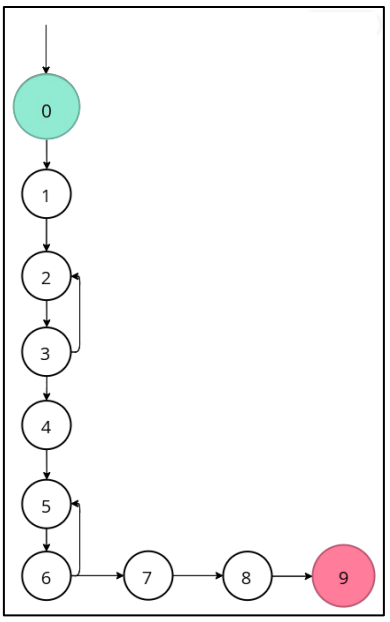

Figure 3-10 PPF Calculator DFG

Variables	Definitions	Uses
yi	{3}	{(3, 2), (3, 4), 4}
tp	{6}	{(6, 5), (6, 7), 7}
yearlyInvestment	{4}	{8}
timePeriod	{7}	{8}
rateOfInterest	{1}	{8}
amount	{8}	{9}

Variables	All DU Path Coverage
yi	[0,1,2,3,4,5,6,7,8,9], [0,1,2,3,2,3,4,5,6,7,8,9]
tp	[0,1,2,3,4,5,6,7,8,9], [0,1,2,3,4,5,6,5,6,7,8,9]
yearlyInvestment	[0,1,2,3,4,5,6,7,8,9]
timePeriod	[0,1,2,3,4,5,6,7,8,9]
rateOfInterest	[0,1,2,3,4,5,6,7,8,9]
amount	[0,1,2,3,4,5,6,7,8,9]

Variables	All Def Coverage
yi	[0,1,2,3,4,5,6,7,8,9]
tp	[0,1,2,3,4,5,6,7,8,9]
yearlyInvestment	[0,1,2,3,4,5,6,7,8,9]
timePeriod	[0,1,2,3,4,5,6,7,8,9]
rateOfInterest	[0,1,2,3,4,5,6,7,8,9]
amount	[0,1,2,3,4,5,6,7,8,9]

```java
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class PPFCalculatorTest {

    String input1 = "100000\n2\n"; // [0,1,2,3,4,5,6,7,8,9]
    String input2 = "-10000\n200000\n2\n"; // [0,1,2,3,2,3,4,5,6,7,8,9]
    String input3 = "200000\n-2\n2\n"; // [0,1,2,3,4,5,6,5,6,7,8,9]

    public void testing(String input, Long expectedTax){
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);
        PPFCalculator ppfCalculator = new PPFCalculator();
        Long actual = ppfCalculator.init();
        Assert.assertEquals(expectedTax,actual);
    }

    @Test
    public void testCase1(){
        testing(input1, 207099L);
    }

    @Test
    public void testCase2(){
        testing(input2, 414199L);
    }

    @Test
    public void testCase3(){
        testing(input3, 414199L);
    }

}
```

Figure 3-11 PPF Calculator Test Code

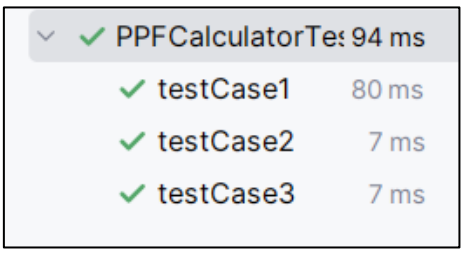

Figure 3-12 PPF Calculator Test Cases

3.5 SIP CALCULATOR

Data Flow Graph:

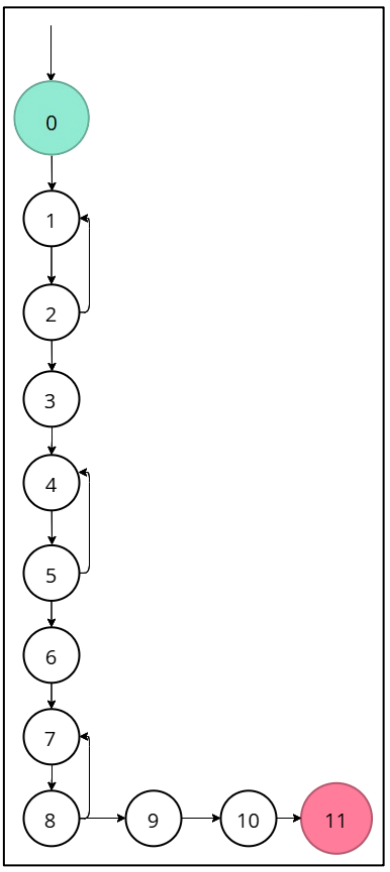

Figure 3-13 SIP Calculator DFG

Variable	Definitions	Uses
val	{2, 5, 8}	{(2, 1), (2, 3), 3, (5, 4), (5, 6), 6, (8, 7), (8, 9), 9}
monthlyInvestment	{3}	{10}
expectedReturnRateInPercentage	{6}	{10}
timePeriodInYear	{9}	{10}
amount	{10}	{11}

Variable	All DU-Path Coverage
Val	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,9,10,11]
monthlyInvestment	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
expectedReturnRateInPercentage	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
timePeriodInYear	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
Amount	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Variable	All-Def Path Coverage
val	[0,1,2,3,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,4,5,6,7,8,9,10,11], [0,1,2,3,4,5,6,7,8,7,8,9,10,11]
monthlyInvestment	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
expectedReturnRateInPercentage	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
timePeriodInYear	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
amount	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

```java
package org.example;

import org.junit.Assert;
import org.junit.Test;

import java.io.ByteArrayInputStream;

public class SIPCalculatorTest {

    String input1 = "3500\n5.5\n2\n"; // [0,1,2,3,4,5,6,7,8,9,10,11]
    String input2 = "5000\n-5\n5.5\n2\n"; // [0,1,2,3,4,5,4,5,6,7,8,9,10,11]
    String input3 = "5000\n5.5\n-2\n2\n"; //  [0,1,2,3,4,5,6,7,8,7,8,9,10,11]
    String input4 = "-10000\n5000\n5.5\n2\n"; // [0,1,2,1,2,3,4,5,6,7,8,9,10,11]


    public void testing(String input, Long expectedTax){
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);
        SIPCalculator sipCalculator = new SIPCalculator();
        Long actual = sipCalculator.init();
        Assert.assertEquals(expectedTax,actual);
    }

    @Test
    public void testCase1(){
        testing(input1, 88985L);
    }

    @Test
    public void testCase2(){
        testing(input2, 127122L);
    }

    @Test
    public void testCase3(){
        testing(input3, 127122L);
    }

    @Test
    public void testCase4(){
        testing(input4, 127122L);
    }

}
```

Figure 3-14 SIP Calculator Test Code

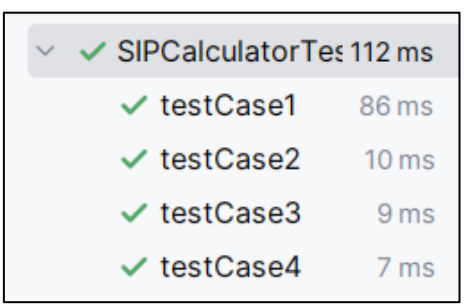

Figure 3-15 SIP Calculator Test Cases

3.6 SWP CALCULATOR

Data Flow Graph:

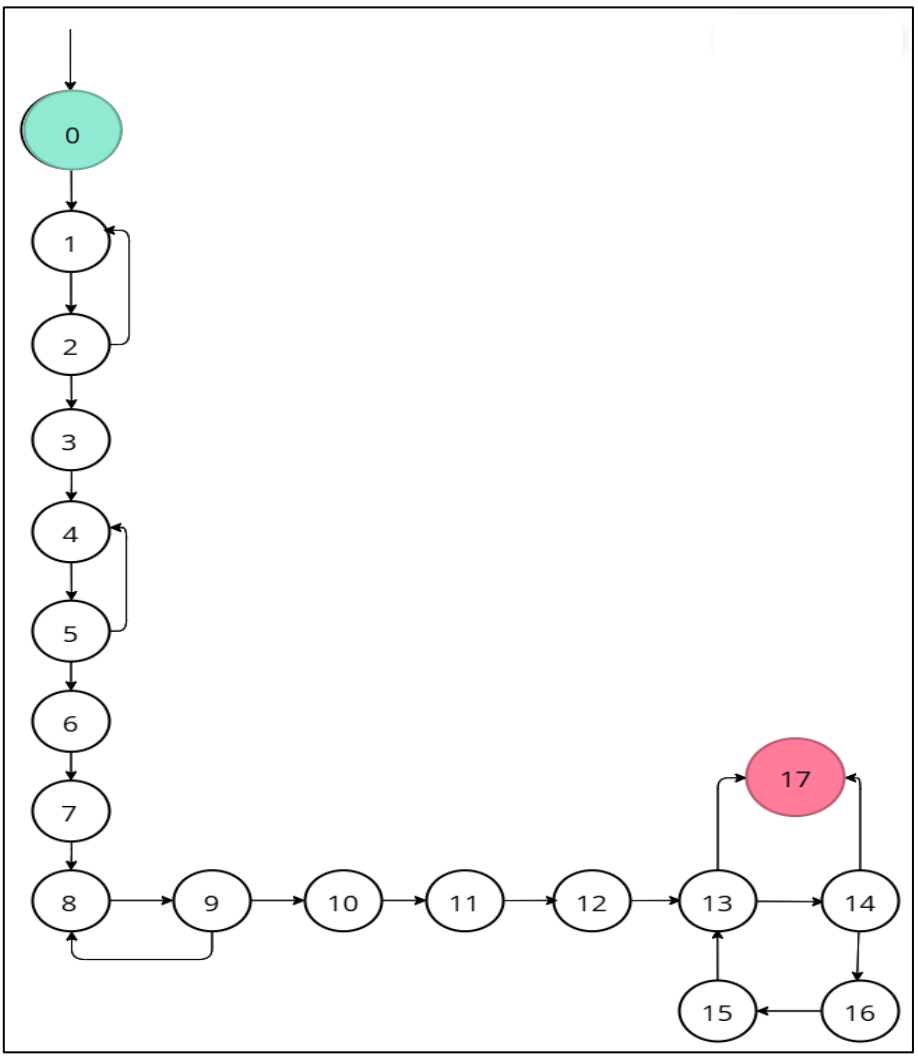

Figure 3-16 SWP Calculator DFG

Variable	Definitions	Uses
val	{2, 5, 7, 9}	{(2, 1), (2, 3), (5, 4), (5, 6), (6, 7), (9, 8), (9, 10), 10}
totalInvestment	{3}	{11}
withdrawalAmount	{6}	{11}
expectedReturnRate	{7}	{11}
timePeriod	{10}	{11}
deduct	{11}	{14}
val1	{11, 14, 15}	{14, (14, 17), (14, 15), 15}
gain	{11, 15}	{15}
N	{11}	{(13, 14), (13, 17)}
I	{12, 16}	{(13, 14), (13, 17), 16}
returnAmnt	{17}	{17}
tmp	{15}	{15}

Variable	All DU-Path Coverage
val	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,6,7,8,9,8,9,10,11,12,13,17]
totalInvestment	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17]
withdrawalAmount	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17]
expectedReturnRate	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17]
timePeriod	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17]
deduct	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17]
val1	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17]
gain	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]
n	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
i	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]
returnAmnt	No Path Needed
tmp	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]

Variable	All-Def Path Coverage
val	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,4,5,6,7,8,9,10,11,12,13,17], [0,1,2,3,4,5,6,7,8,9,8,9,10,11,12,13,17]
totalInvestment	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17]
withdrawalAmount	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17]
expectedReturnRate	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17]
timePeriod	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17]
deduct	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17]
val1	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17]
gain	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]
n	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
i	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17], [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17]
returnAmnt	No Path Needed
tmp	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15, 16,13,17]

```java
package org.example;

import org.junit.Assert;
import org.junit.Test;

import java.io.ByteArrayInputStream;

public class SWPCalculatorTest {
    String input1 = "100000\n5000\n5.5\n0\n"; // [0,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
    String input2 = "100000\n-5000\n5000\n5.5\n3\n"; // [0,1,2,3,4,5,4,5,6,7,8,9,10,11,12,13,17]
    String input3 = "100000\n5000\n5.5\n-2\n3\n"; // [0,1,2,3,4,5,6,7,8,9,8,9,10,11,12,13,17]
    String input4 = "-3500\n100000\n5000\n5.5\n3\n"; // [0,1,2,1,2,3,4,5,6,7,8,9,10,11,12,13,17]
    String input5 = "500000\n500000\n5.5\n2\n"; // [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17]
    String input6 = "500000\n250000\n5.5\n2\n"; // [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,17]
    String input7 = "600000\n200000\n5.5\n2\n"; // [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,13,14,15,16,13,17]

    public void testing(String input, Long expectedTax){
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);
        SWPCalculator swpCalculator = new SWPCalculator();
        Long actual = swpCalculator.init();
        Assert.assertEquals(expectedTax,actual);
    }

    @Test
    public void testCase1(){
        testing(input1, 0L);
    }

    @Test
    public void testCase2(){
        testing(input2, 4621L);
    }

    @Test
    public void testCase3(){
        testing(input3, 4621L);
    }

    @Test
    public void testCase4(){
        testing(input4, 4621L);
    }

    @Test
    public void testCase5(){
        testing(input5, 4621L);
    }

    @Test
    public void testCase6(){
        testing(input6, 4621L);
    }

    @Test
    public void testCase7(){
        testing(input7, 4621L);
    }

}
```

Figure 3-17 SWP Calculator Test Code

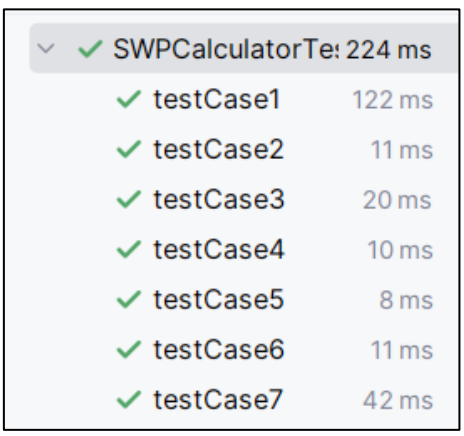

Figure 3-18 SWP Calculator Test Cases

3.7 TAX CALCULATOR

Data Flow Graph:

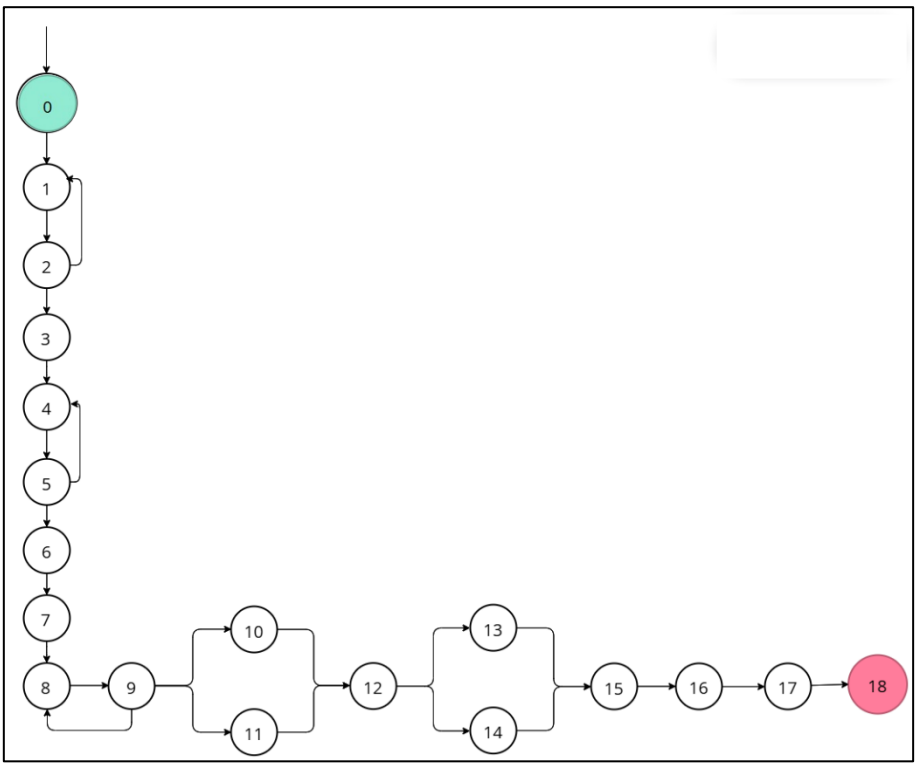

Figure 3-19 GRATUITY Calculator DFG

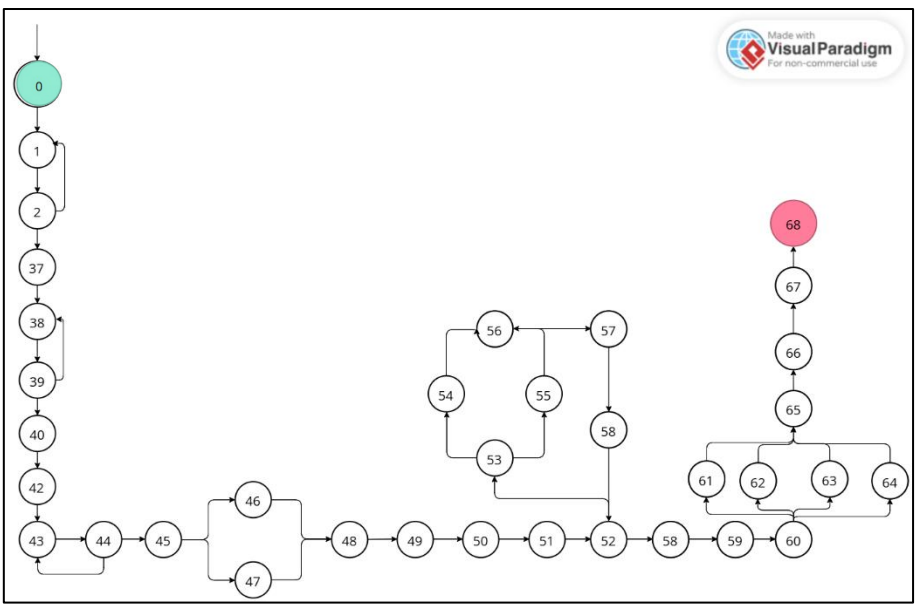

Figure 3-20 GRATUITY Calculator DFG

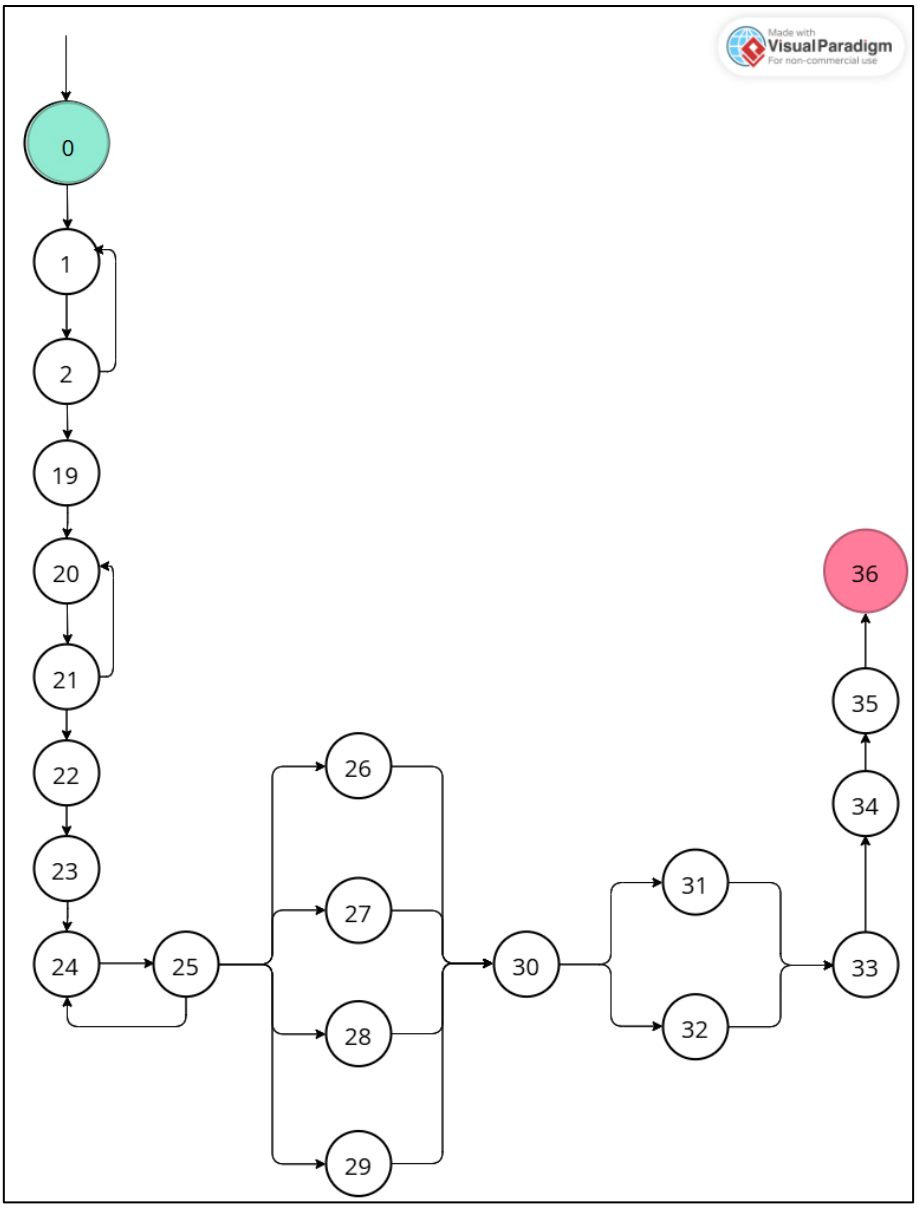

Figure 3-21 GRATUITY Calculator DFG

Variable	Definitions	Uses
slabs	{1}	{(52, 52), (53, 64), (53, 55), (56)}
slab	{1}	{51}
type	{2}	{(2, 1), (2, 3), (2, 19), (2, 37)}
income	{3, 19, 44}	{4, 7, 20, 23, 50}
ngoDonation	{4, 20}	{5, 21}
goDonation	{4, 20}	{5, 21}
netQualifyingLimit	{5, 21}	{5, 21}
deduction	{5, 21, 48}	{6, 22, 49}
deductedAmt	{5, 22, 49}	{7, 23, 50}
taxableAmt	{7, 23, 50, 57}	{10, 11, (12,13), (12,14), 26, 27, 28, 29, (30,31), (30,32), (52,53), 54, 55, 57, (60,61), (60,62), (60,63), (60,64)}
id	{9, 25}	{(9,8), (9,10), (9,11), (25,24), (25,26), (25,27), (25,28), (25,29)}
ta	{10, 11, 26, 27, 28, 29, 56}	{12, 30, 56, 59}
taxAmt	{12, 30, 59}	{ 13, 14, 16, 18, 31, 32, 34, 36, 61, 62, 63, 64, 66, 68 }
sc	{13, 14, 61, 62, 63, 64, 31, 32}	{15, 33, 65}
surcharge	{15, 33, 65}	{18, 36, 68}
charge	{16, 34, 60}	{17, 35, 67}
healthAndEduCess	{17, 35, 67}	{18, 36, 68}
netTax	{18, 36, 68}	{}
schemeID	{39}	{(39, 38), 40}
regimeID	{40}	{(41, 42), (45, 46), (41, 43), (45, 47)}
ageGrp	{40, 42}	{(42, 41), 43}
ageGrpID	{43}	{51}
80c	{45, 46}	{48}
80ccd1b	{45, 46, 47}	{48}
i	{51, 58}	{(52,53), (52,59), (53,54), (53,55), 55, 56, 58}
minA	{54, 55}	{56, 57}

Variable	All DU-Path Coverage

slabs	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

slab	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

type	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68],
	[0, 1, 2],
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36]
	}

income	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

ngoDonation	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36]
	}

goDonation	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36]
	}

netQualifyingLimit	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36]
	}

deduction	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

deductedAmt	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

taxableAmt	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 16, 17, 18],
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 27, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 27, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 28, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 28, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 29, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 29, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 62, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 59, 60, 64, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 59, 60, 63, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 59, 60, 62, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 59, 60, 61, 65, 66, 67, 68]
	}

id	{
	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 27, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 28, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 29, 30, 31, 33, 34, 35, 36]
	}

ta	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 27, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 28, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 29, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 55, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

taxAmt	{ [0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18], [0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 16, 17, 18], [0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 31, 33, 34, 35, 36], [0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36], [0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68], [0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 62, 65, 66, 67, 68], [0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 63, 65, 66, 67, 68], [0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]}
sc	{[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18], [0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 16, 17, 18], [0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 31, 33, 34, 35, 36], [0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36], [0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68], [0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 62, 65, 66, 67, 68], [0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 63, 65, 66, 67, 68], [0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]}

surcharge	{ [0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18], [0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36], [0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]}

charge	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

healthAndEduCess	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

netTax	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

schemeID	{
	[0, 1, 2, 37, 38, 39, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

regimeID	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

ageGrp	{ [0, 1, 2, 37, 38, 39, 40, 41, 42, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68], [0, 1, 2, 37, 38, 39, 40, 41, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]}

ageGrpID	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

80c	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]

80ccd1b	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

i	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 55, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 59, 60, 64, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

minA	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 53, 55, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

Variable	All Def Coverage

slabs	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

slab	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

type	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

income	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

ngoDonation	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36]
	}

goDonation	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36]
	}

netQualifyingLimit	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

deduction	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

deductedAmt	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

taxableAmt	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18]
	}

id	{
	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 33, 34, 35, 36]
	}

ta	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 27, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 28, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 29, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

taxAmt	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68]
	}

sc	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 31, 33, 34, 35, 36],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 61, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 62, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 63, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

surcharge	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

charge	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

healthAndEduCess	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

netTax	{
	[0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18],
	[0, 1, 2, 19, 20, 21, 22, 23, 24, 25, 24, 25, 26, 30, 32, 33, 34, 35, 36],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

schemeID	{
	[0, 1, 2, 37, 38, 39, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

regimeID	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

ageGrp	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

ageGrpID	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
80c	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]

80ccd1b	{
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68],
	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 54, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
	}

i	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 52, 53, 55, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]
minA	[0, 1, 2, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 52, 53, 55, 56, 57, 58, 52, 59, 60, 64, 65, 66, 67, 68]

```java
package org.example;

import org.junit.Assert;
import org.junit.Test;

import java.io.ByteArrayInputStream;

public class TaxCalculatorTest {

    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
    String input1 = "1\n1\n1\n8000000\n10000\n25000\n1000\n2000\n6788\n";
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,59,60,61,65,66,67,68]
    String input2 = "1\n1\n1\n8000000\n10000\n25000\n1000\n2000\n6788\n";
    //  [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
    String input3 = "1\n1\n1\n80000000\n10000\n25000\n1000\n2000\n6788\n";
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,61,65,66,67,68]
    String input4 = "1\n1\n1\n8000000\n10000\n25000\n1000\n2000\n6788\n";
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,62,65,66,67,68]
    String input5 = "1\n1\n1\n15000000\n10000\n25000\n1000\n2000\n6788\n";
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,63,65,66,67,68]
    String input6 = "1\n1\n1\n40000000\n10000\n25000\n1000\n2000\n6788\n";
    // [0,1,2,37,38,39,40,41,43,44,45,47,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
    String input7 = "1\n2\n80000000\n10000";
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,53,55,56,57,58,52,59,60,64,65,66,67,68]
    String input8 = "1\n1\n1\n70000000\n10000\n25000\n1000\n2000\n6788\n";
    // [0,1,2,37,38,39,40,41,42,41,42,43,44,45,46,48,49,50,51,52,53,55,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
    String input9 = "1\n1\n-1\n1\n70000000\n10000\n25000\n1000\n2000\n6788\n";
    // Following test-paths are not possible practically based on business logic, although they are possible theoretically
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,52,53,54,56,57,58,52,59,60,64,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,64,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,64,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,63,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,62,65,66,67,68]
    // [0,1,2,37,38,39,40,41,42,43,44,45,46,48,49,50,51,52,59,60,61,65,66,67,68]
    // [0,1,2,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18]
    String input10 = "-1\n3\n80000000\n200000\n300000\n1\n";
    // [0,1,2,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18]
    String input11 = "-1\n3\n80000000\n200000\n300000\n0\n"
    // [0,1,2,3,4,5,6,7,8,9,10,12,13,15,16,17,18]
    String input12 = "3\n80000000\n200000\n300000\n1\n";
    // [0,1,2,3,4,5,6,7,8,9,8,9,10,12,13,15,16,17,18]
    String input13 = "3\n80000000\n200000\n300000\n-1\n1\n";
    // [0,1,2,3,4,5,6,7,8,9,11,12,13,15,16,17,18]
    String input14 = "3\n80000000\n200000\n300000\n0\n";
    // [0,1,2,1,2,3,4,5,6,7,8,9,11,12,14,15,16,17,18]
    String input15 = "-1\n3\n150000000\n200000\n300000\n1\n";
    // [0,1,2,1,2,3,4,5,6,7,8,9,10,12,14,15,16,17,18]
    // [0,1,2,19,20,21,22,23,24,25,26,30,31,33,34,35,36]
    String input16 = "2\n70000000\n200000\n300000\n1\n";
    // [0,1,2,19,20,21,22,23,24,25,27,30,31,33,34,35,36]
    String input17 = "2\n70000000\n200000\n300000\n2\n";
    // [0,1,2,19,20,21,22,23,24,25,24,25,26,30,31,33,34,35,36]
    String input18 = "2\n70000000\n200000\n300000\n-1\n1\n";
    // [0,1,2,19,20,21,22,23,24,25,28,30,31,33,34,35,36]
    String input19 = "2\n70000000\n200000\n300000\n3\n";
    // [0,1,2,19,20,21,22,23,24,25,24,25,26,30,32,33,34,35,36]
    String input20 = "2\n150000000\n200000\n300000\n-1\n1\n";
    // [0,1,2,19,20,21,22,23,24,25,29,30,31,33,34,35,36]
    String input21 = "2\n70000000\n200000\n300000\n4\n";
    // [0,1,2,19,20,21,22,23,24,25,26,30,32,33,34,35,36]
    String input22 = "2\n250000000\n200000\n300000\n1\n";
    // [0,1,2,19,20,21,22,23,24,25,27,30,32,33,34,35,36]
    String input23 = "2\n250000000\n200000\n300000\n2\n";
    // [0,1,2,19,20,21,22,23,24,25,28,30,32,33,34,35,36]
    String input24 = "2\n250000000\n200000\n300000\n3\n";
    // [0,1,2,19,20,21,22,23,24,25,29,30,32,33,34,35,36]
    String input25 = "2\n250000000\n200000\n300000\n4\n";
    // Objective : finding bug in the code through test case
    // providing negative value in the deduction, wrongfully increases the taxable amount
    // because the statement min(deduction,150000), is not directly appropriate in this case
    // following test case should give tax amount 75400, but instead gives 84715
    // as negative deduction amount should be ignored
    String input26 =  "1\n1\n1\n800000\n-10000\n-25000\n-1000\n-2000\n-6788\n";
    public void testing(String input, int expectedTax){
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);
        TaxCalculator taxCalculator = new TaxCalculator();
        int netTax = taxCalculator.init().intValue();
        Assert.assertEquals(expectedTax,netTax);
    }
    @Test
    public void testCase1(){
        testing(input1, 2506932);
    }
    @Test
    public void testCase2(){
        testing(input2, 2506932);
    }
    @Test
    public void testCase3(){
        testing(input3, 33556679);
    }
    @Test
    public void testCase4(){
        testing(input4, 2506932);
    }
```

Figure 3-22 TAX Calculator Test Code - I

```
    @Test
    public void testCase4(){
        testing(input4, 2506932);
    }
    @Test
    public void testCase5(){
        testing(input5, 5115885);
    }
    @Test
    public void testCase6(){
        testing(input6, 15220792);
    }
    @Test
    public void testCase7(){
        testing(input7, 33465645);
    }
    @Test
    public void testCase8(){
        testing(input8, 29326679);
    }
    @Test
    public void testCase9(){
        testing(input9, 29326679);
    }
    @Test
    public void testCase10(){
        testing(input10, 42135000);
    }
    @Test
    public void testCase11(){
        testing(input11, 33708000);
    }
    @Test
    public void testCase12(){
        testing(input12, 42135000);
    }
    @Test
    public void testCase13(){
        testing(input13, 42135000);
    }
    @Test
    public void testCase14(){
        testing(input14, 33708000);
    }
    @Test
    public void testCase15(){
        testing(input15, 81477500);
    }
    @Test
    public void testCase16(){
        testing(input16, 19286250);
    }
    @Test
    public void testCase17(){
        testing(input17, 16971900);
    }
    @Test
    public void testCase18(){
        testing(input18, 19286250);
    }
    @Test
    public void testCase19(){
        testing(input19, 11571750);
    }
    @Test
    public void testCase20(){
        testing(input20, 43355000);
    }
    @Test
    public void testCase21(){
        testing(input21, 23143500);
    }
    @Test
    public void testCase22(){
        testing(input22, 72355000);
    }
    @Test
    public void testCase23(){
        testing(input23, 63672400);
    }
    @Test
    public void testCase24(){
        testing(input24, 43413000);
    }
    @Test
    public void testCase25(){
        testing(input25, 86826000);
    }
    @Test
    public void testCase26(){
        testing(input26, 75400);
    }
}
```

Figure 3-23 TAX Calculator Test Code - II

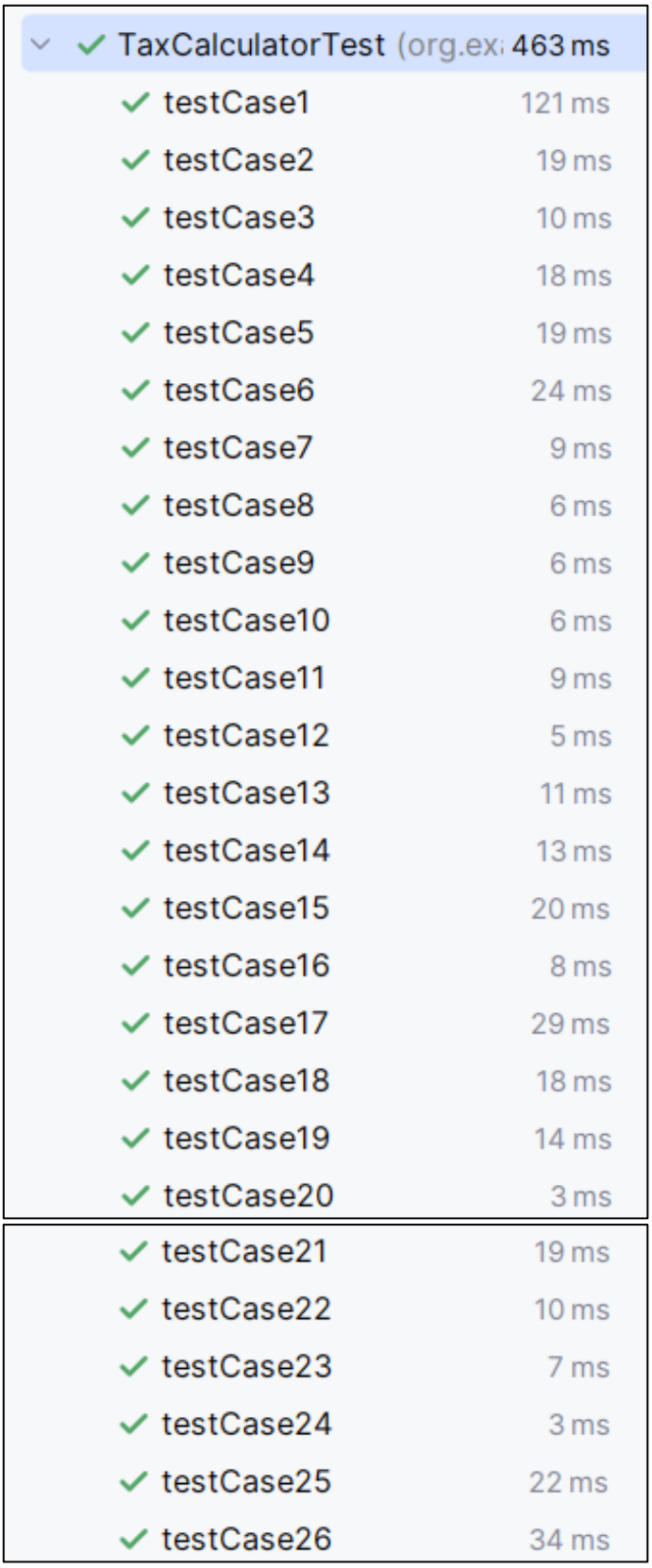

Figure 3-24 TAX Calculator Test Cases

3.8 GOALSAVINGS CALCULATOR

Data Flow Graph:

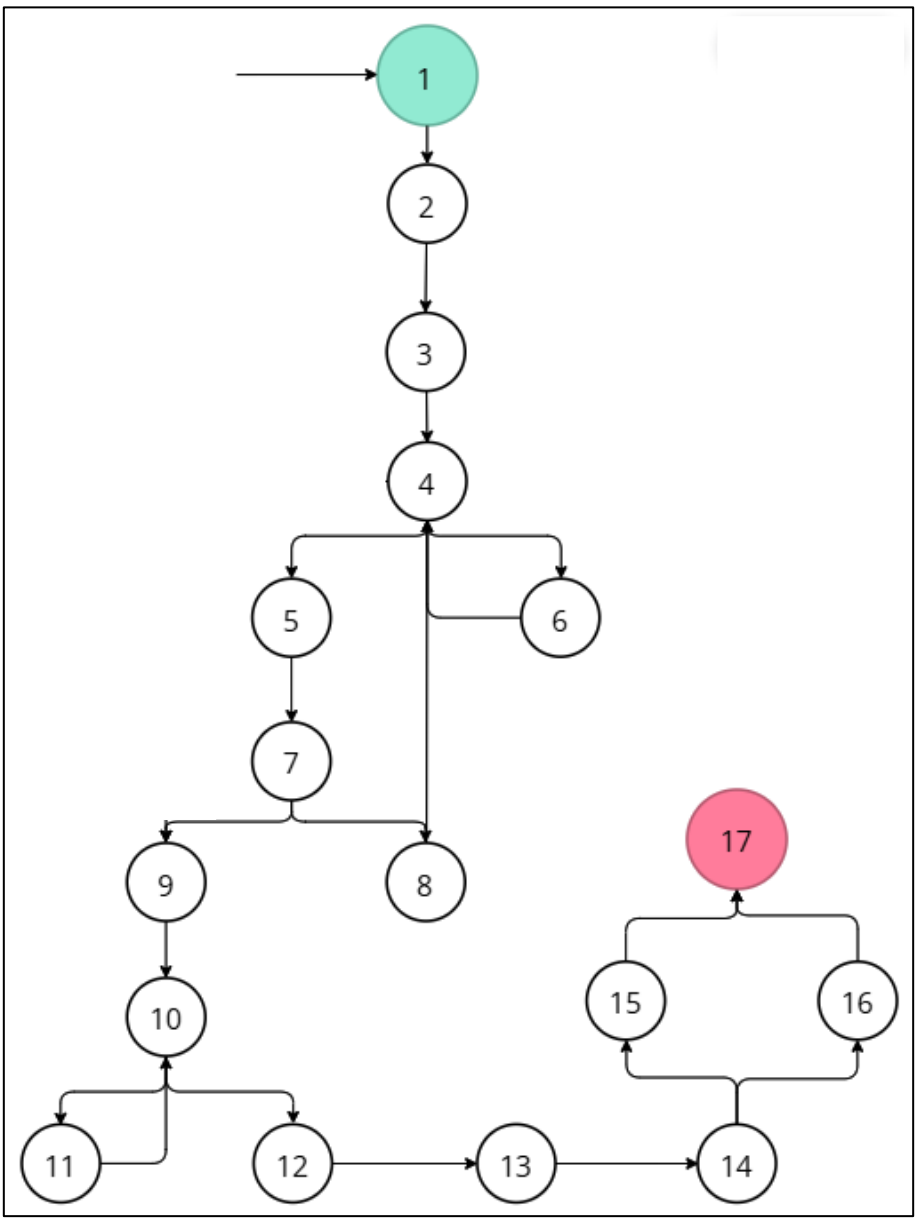

Figure 3-25 GOALSAVINGS Calculator DFG

Variable	Definitions	Uses
goalAmt	{2, 4}	{14, 16, (4, 6)}
initalSavings	{2, 5}	{14, (7, 8)}
retRate	{2, 5}	{14, 16, (7, 8)}
years	{2, 4}	{14, 16, (4, 6)}
freq	{2, 10}	{16, (10, 12)}

Variable	All DU-Path Coverage

goalAmt	{
	[1,2,3,4,6,4,5,7,9,10,12,13,14,16,17],
	[1,2,3,4,5,7,9,10,12,13,14,16,17]
	}

initalSavings	{
	[1,2,3,4,5,7,8,4,5,7,9,10,12,13,14,16,17],
	[1,2,3,4,5,7,9,10,12,13,14,16,17]
	}

retRate	{
	[1,2,3,4,5,7,8,4,5,7,9,10,12,13,14,16,17],
	[1,2,3,4,5,7,9,10,12,13,14,16,17]
	}

years	{
	[1,2,3,4,6,4,5,7,9,10,12,13,14,16,17],
	[1,2,3,4,5,7,9,10,12,13,14,16,17]
	}

freq	{
	[1,2,3,4,5,7,9,10,11,10,12,13,14,15,17],
	[1,2,3,4,5,7,9,10,12,13,14,15,17],
	[1,2,3,4,5,7,9,10,12,13,14,16,17]
	}

Variable	All Def Coverage
goalAmt	[1,2,3,4,6,4,9,10,12,13,14,16,17]
initalSavings	[1,2,3,4,5,7,8,4,5,7,9,10,12,13,14,16,17]
retRate	[1,2,3,4,5,7,8,4,5,7,9,10,12,13,14,16,17]
years	[1,2,3,4,6,4,9,10,12,13,14,16,17]
freq	[1,2,3,4,5,7,9,10,12,13,14,15,17]

```java
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class GoalSavingsCalculatorTest {

    String input1 = "1000000\n10\n8\n100000\nannual\n"; // [1, 2, 3, 4, 5, 7, 9, 10, 12, 13, 14, 16, 17]
    String input2 = "100000\n8\n5\n80000\nannual\n"; // [1, 2, 3, 4, 5, 7, 9, 10, 12, 13, 14, 15, 17]
    String input3 = "-1000000\n5\n100000\n8\n5\n10000\nannual\n"; // [1,2,3,4,6,4,5,7,9,10,12,13,14,16,17]
    String input4 = "1000000\n5\n0\n200000\n1000000\n5\n8\n200000\nmonthly\n"; // [1,2,3,4,5,7,8,4,5,7,9,10,12,13,14,16,17]
    String input5 = "1000000\n10\n8\n100000\nannua\nannual"; // [1,2,3,4,5,7,9,10,11,10,12,13,14,15,17],


    public void testing(String input, String expectedMessage) {
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);

        GoalSavingsCalculator calculator = new GoalSavingsCalculator();
        calculator.init();

        // Assuming output is verified by matching part of the expected message in the console output
        // You might need additional helpers to capture and verify console outputs
    }

    @Test
    public void testCase1() {
        testing(input1, "To achieve your goal of ₹1000000.00, you need to save ₹54126.54 per year");
    }

    @Test
    public void testCase2() {
        testing(input2, "Congratulations! Your current savings are sufficient to achieve your goal.");
    }

    @Test
    public void testCase3() {
        testing(input3, "Goal amount and time frame must be positive.");
    }

    @Test
    public void testCase4() {
        testing(input4, "Goal amount and time frame must be positive.");
    }
    @Test
    public void testCase5() {
        testing(input5, "To achieve your goal of ₹1000000.00, you need to save ₹54126.54 per year");
    }
}
```

Figure 3-26 GOALSAVINGS Calculator Test Code

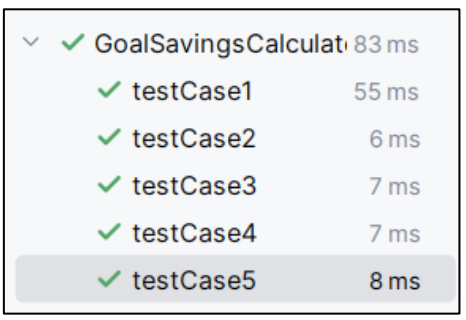

Figure 3-27 GOALSAVINGS Calculator Test Cases

3.9 BUDGETPLANNING CALCULATOR

Data Flow Graph:

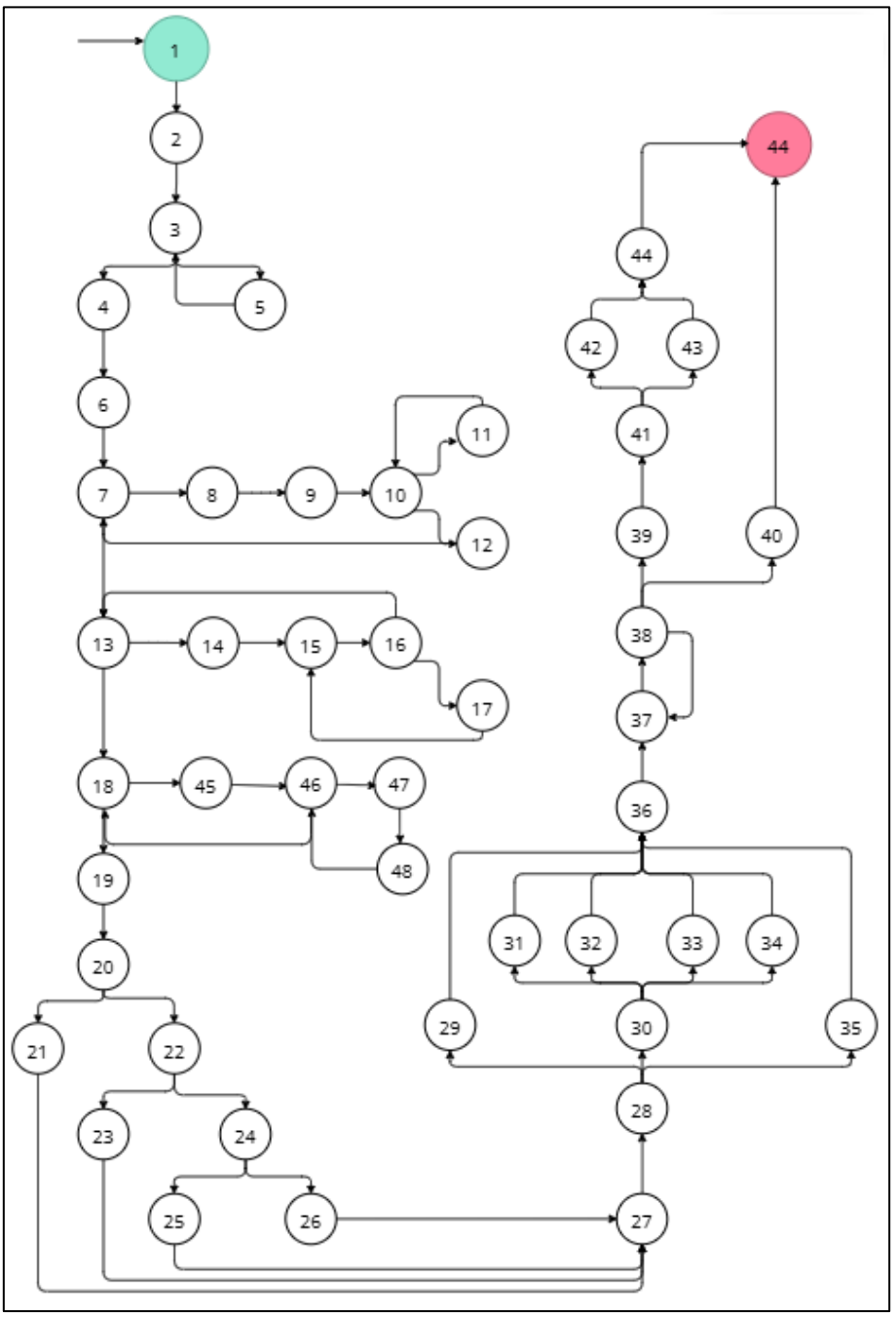

Figure 3-28 BUDGETPLANNING Calculator DFG

Variable	Definitions	Uses
monthlyIncome	{1, 3, 2}	{22, 24, 26, 29, (20, 22), (22, 24), (24, 26), (28,29)}
emergencyFund	{1, 7}	{22, (30, 31), (30, 32), (30, 33)}
savingGoal	{1, 7}	{24, (30, 31), (30, 32), (30, 33)}
fixedEx	{1}	{13, 18}
varibleEx	{1}	{13, 18}
totalFixedEx	{19}	{19, 28}
totalVarEx	{19}	{19}
totalEx	{19}	{24, 26, 28, (20, 22), (22, 24), (24, 26), (30,31)}
emergencyAllocation	{19}	{22, 24, 26, 28, (22, 24), (24, 26), (30, 31), (30, 32), (30, 33)}
savingAllocation	{19}	{24, 26, 28, (24, 26), (30, 31), (30, 32)}
descriptionFunds	{19,23,25}	{26, 28, (28, 30)}

Variable	DU Path

monthlyIncome	{
	[3,4,6,7,13,18,19,20,22],
	[3,4,6,7,13,18,19,20,22,24],
	[3,4,6,7,13,18,19,20,22,24,26],
	[3,4,6,7,13,18,19,20,21,27,28,29],
	[3,4,6,7,13,18,19,20,22,23,27,28,29],
	[3,4,6,7,13,18,19,20,22,24,26,27,28,29],
	[3,4,6,7,13,18,19,20,22,24,25,27,28,29]
	}

emergencyFund	{
	[7,13,18,19,20,22],
	[7,13,18,19,20,21,27,28,30,31],
	[7,13,18,19,20,21,27,28,30,32],
	[7,13,18,19,20,21,27,28,30,33],
	[7,13,18,19,20,22,23,27,28,30,31],
	[7,13,18,19,20,22,23,27,28,30,32],
	[7,13,18,19,20,22,23,27,28,30,33],
	[7,13,18,19,20,22,24,26,27,28,30,31],
	[7,13,18,19,20,22,24,26,27,28,30,32],
	[7,13,18,19,20,22,24,26,27,28,30,33],
	[7,13,18,19,20,22,24,25,27,28,30,31],
	[7,13,18,19,20,22,24,25,27,28,30,32],
	[7,13,18,19,20,22,24,25,27,28,30,33],
	}

savingGoal	{ [7,13,18,19,20,22,24] [7,13,18,19,20,21,27,28,30,31], [7,13,18,19,20,21,27,28,30,32], [7,13,18,19,20,21,27,28,30,33], [7,13,18,19,20,22,23,27,28,30,31], [7,13,18,19,20,22,23,27,28,30,32], [7,13,18,19,20,22,23,27,28,30,33], [7,13,18,19,20,22,24,26,27,28,30,31], [7,13,18,19,20,22,24,26,27,28,30,32], [7,13,18,19,20,22,24,26,27,28,30,33], [7,13,18,19,20,22,24,25,27,28,30,31],

	[7,13,18,19,20,22,24,25,27,28,30,32], [7,13,18,19,20,22,24,25,27,28,30,33] }

fixedEx	{
	[1,2,3,4,6,7,13],
	[1,2,3,4,6,7,13,18]
	}

varibleEx	{
	[1,2,3,4,6,7,13],
	[1,2,3,4,6,7,13,18]
	}

totalFixedEx	{
	[19,20,21,27,28],
	[19,20,22,23,27,28],
	[19,20,22,24,26,27,28],
	[19,20,22,24,25,27,28]
	}

totalVarEx	No Path

totalEx	{
	[19,20,22],
	[19,20,22,24],
	[19,20,22,24,26],
	[19,20,21,27,28],
	[19,20,22,23,27,28],
	[19,20,22,24,26,27,28],
	[19,20,22,24,25,27,28],
	[19,20,21,27,28,30,31],
	[19,20,22,23,27,28,30,31],
	[19,20,22,24,26,27,28,30,31],
	[19,20,22,24,25,27,28,30,31]
	}

emergencyAllocation	{
	[19,20,22],
	[19,20,22,24],
	[19,20,22,24,26],
	[19,20,21,27,28],
	[19,20,22,23,27,28],
	[19,20,22,24,26,27,28],
	[19,20,22,24,25,27,28],
	[19,20,21,27,28,30,31],
	[19,20,21,27,28,30,32],
	[19,20,21,27,28,30,33],
	[19,20,22,23,27,28,30,33],
	[19,20,22,23,27,28,30,31],
	[19,20,22,23,27,28,30,32],
	[19,20,22,24,26,27,28,30,33],
	[19,20,22,24,26,27,28,30,31],
	[19,20,22,24,26,27,28,30,32],
	[19,20,22,24,25,27,28,30,33],
	[19,20,22,24,25,27,28,30,31],
	[19,20,22,24,25,27,28,30,32]
	}

savingAllocation	{
	[19,20,22,24],
	[19,20,22,24,26],
	[19,20,21,27,28],
	[19,20,22,23,27,28],
	[19,20,22,24,26,27,28],
	[19,20,22,24,25,27,28],
	[19,20,21,27,28,30,31],
	[19,20,21,27,28,30,32],
	[19,20,21,27,28,30,33],
	[19,20,22,23,27,28,30,33],
	[19,20,22,23,27,28,30,31],
	[19,20,22,23,27,28,30,32],
	[19,20,22,24,26,27,28,30,33],
	[19,20,22,24,26,27,28,30,31],
	[19,20,22,24,26,27,28,30,32],
	[19,20,22,24,25,27,28,30,33],
	[19,20,22,24,25,27,28,30,31],
	[19,20,22,24,25,27,28,30,32]
	}

descriptionFunds	{
	[19,20,22,24,26],
	[19,20,21,27,28],
	[19,20,21,27,28,30],
	[19,20,22,24,26,27,28],
	[19,20,22,24,26,27,28,30],
	[23,27,28],
	[23,27,28,30],
	[25,27,28],
	[25,27,28,30]
	}

Observations:

- The **Budget Planning Calculator CFG** consists of **49 nodes**, making manual identification of **All Def-Use (DU) paths** and **All Def paths** highly labor-intensive.
- There are **11 variables** in the function, each requiring test paths, which further increases the complexity of manual path enumeration.
- The tool used (**http://cs.gmu.edu:8080/offutt/coverage/DFGraphCoverage**) provided **DU paths**, but it did not generate **All-DU path Coverage** and **All-Def paths Coverage** for larger graphs.
- For smaller graphs, the tool performed better.
- Due to **tool limitations**, **All-DU paths** and **All-Def path coverage** were not performed for this particular function.

3.10 RETIREMENT CORPUS CALCULATOR

Data Flow Graph:

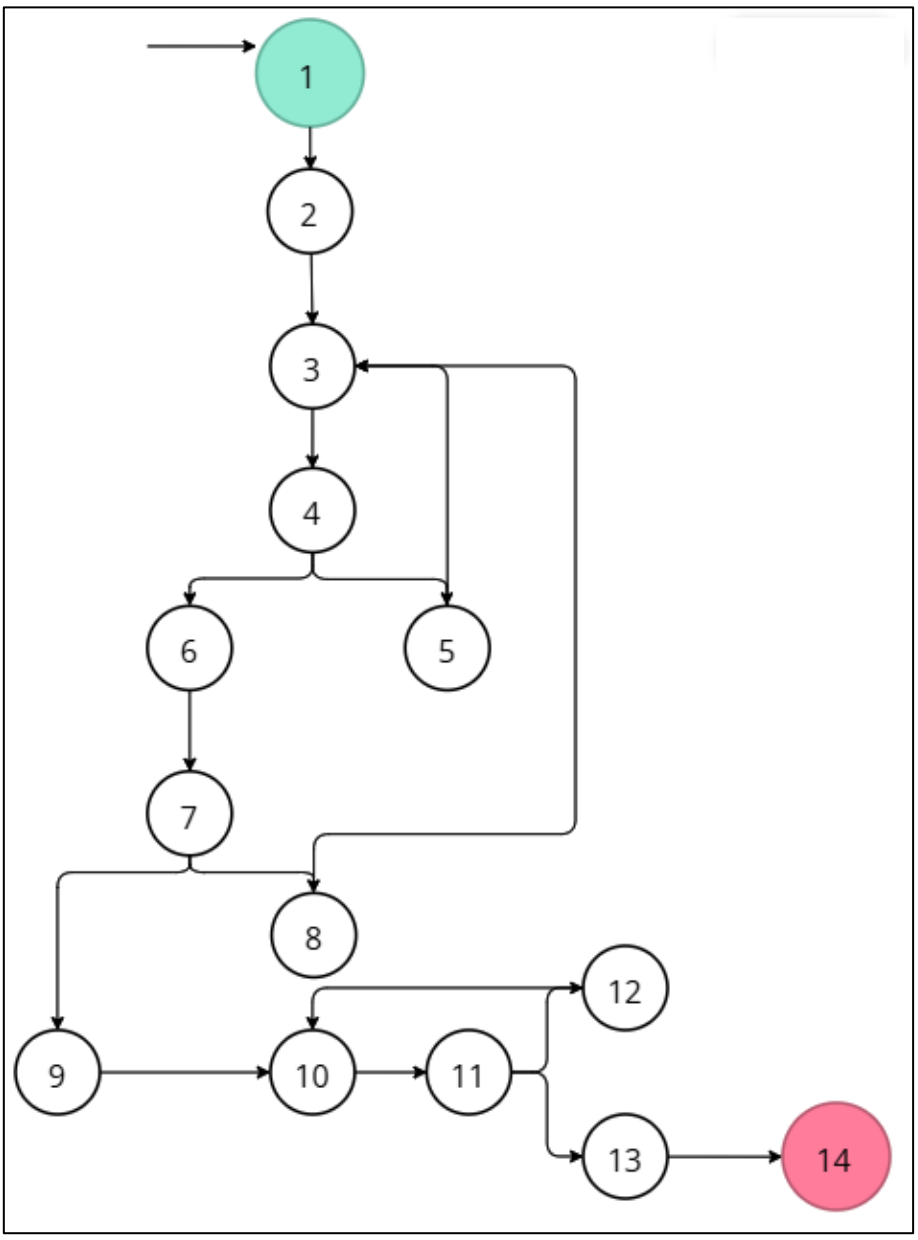

Figure 3-29 RETIREMENT CORPUS Calculator DFG

Variable	Definitions	Uses
currentAge	{2, 4}	{14, (14, 5)}
retirementAge	{2, 4}	{14, (4, 5), (7, 8)}
lifeExpectancy	{2, 6}	{14, (7, 8)}
currentExpenses	{2, 11}	{14, (11, 12)}
inflationRate	{2, 11}	{14, (11, 12)}
returnRate	{2, 11}	{14, (11, 12)}

Variable	All DU-Path Coverage

currentAge	{
	[1,2,3,4,5,3,4,6,7,9,10,11,13,14],
	[1,2,3,4,6,7,9,10,11,13,14]
	}

retirementAge	{
	[1,2,3,4,5,3,4,6,7,9,10,11,13,14],
	[1,2,3,4,6,7,8,3,4,6,7,9,10,11,13,14],
	[1,2,3,4,6,7,9,10,11,13,14]
	}

lifeExpectancy	{
	[1,2,3,4,6,7,8,3,4,6,7,9,10,11,13,14],
	[1,2,3,4,6,7,9,10,11,13,14]
	}

currentExpenses	{
	[1,2,3,4,6,7,9,10,11,12,10,11,13,14],
	[1,2,3,4,6,7,9,10,11,13,14]
	}

inflationRate	{
	[1,2,3,4,6,7,9,10,11,12,10,11,13,14],
	[1,2,3,4,6,7,9,10,11,13,14]
	}

returnRate	{
	[1,2,3,4,6,7,9,10,11,12,10,11,13,14],
	[1,2,3,4,6,7,9,10,11,13,14]
	}

Variable	All Def Coverage
currentAge	[1,2,3,4,5,3,4,6,7,9,10,11,13,14]
retirementAge	[1,2,3,4,5,3,4,6,7,9,10,11,13,14]
lifeExpectancy	[1,2,3,4,6,7,8,3,4,6,7,9,10,11,13,14]
currentExpenses	[1,2,3,4,6,7,9,10,11,12,10,11,13,14]
inflationRate	[1,2,3,4,6,7,9,10,11,12,10,11,13,14]
returnRate	[1,2,3,4,6,7,9,10,11,12,10,11,13,14]

```java
package org.example;

import org.junit.Assert;
import org.junit.Test;
import java.io.ByteArrayInputStream;

public class GoalSavingsCalculatorTest {

    String input1 = "1000000\n10\n8\n100000\nannual\n"; // [1, 2, 3, 4, 5, 7, 9, 10, 12, 13, 14, 16, 17]
    String input2 = "100000\n8\n5\n80000\nannual\n"; // [1, 2, 3, 4, 5, 7, 9, 10, 12, 13, 14, 15, 17]
    String input3 = "-1000000\n5\n100000\n8\n5\n10000\nannual\n"; // [1,2,3,4,6,4,5,7,9,10,12,13,14,16,17]
    String input4 = "1000000\n5\n0\n200000\n1000000\n5\n8\n200000\nmonthly\n"; // [1,2,3,4,5,7,8,4,5,7,9,10,12,13,14,16,17]
    String input5 = "1000000\n10\n8\n100000\nannua\nannual"; // [1,2,3,4,5,7,9,10,11,10,12,13,14,15,17],


    public void testing(String input, String expectedMessage) {
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(input.getBytes());
        System.setIn(byteArrayInputStream);

        GoalSavingsCalculator calculator = new GoalSavingsCalculator();
        calculator.init();

        // Assuming output is verified by matching part of the expected message in the console output
        // You might need additional helpers to capture and verify console outputs
    }

    @Test
    public void testCase1() {
        testing(input1, "To achieve your goal of ₹1000000.00, you need to save ₹54126.54 per year");
    }

    @Test
    public void testCase2() {
        testing(input2, "Congratulations! Your current savings are sufficient to achieve your goal.");
    }

    @Test
    public void testCase3() {
        testing(input3, "Goal amount and time frame must be positive.");
    }

    @Test
    public void testCase4() {
        testing(input4, "Goal amount and time frame must be positive.");
    }
    @Test
    public void testCase5() {
        testing(input5, "To achieve your goal of ₹1000000.00, you need to save ₹54126.54 per year");
    }
}
```

Figure 3-29 RETIREMENT CORPUS Calculator Test Code

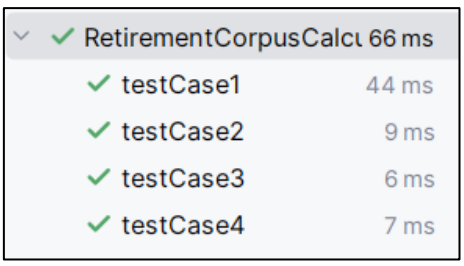

Figure 3-30 RETIREMENT CORPUS Calculator Test Cases

REFERENCES

- **DFGraphCoverage:** https://dfgraphcoverage.com
- **JUnit:** https://junit.org
- **Income Tax Department Site:** https://incometaxindia.gov.in
- **ChatGPT:** https://chat.openai.com
- **DFG:** https://online.visual-paradigm.com/
- **Code to Image:** https://10015.io/tools/code-to-image-converter
- **Software Testing PPT:** https://shorturl.at/PDJ9Q