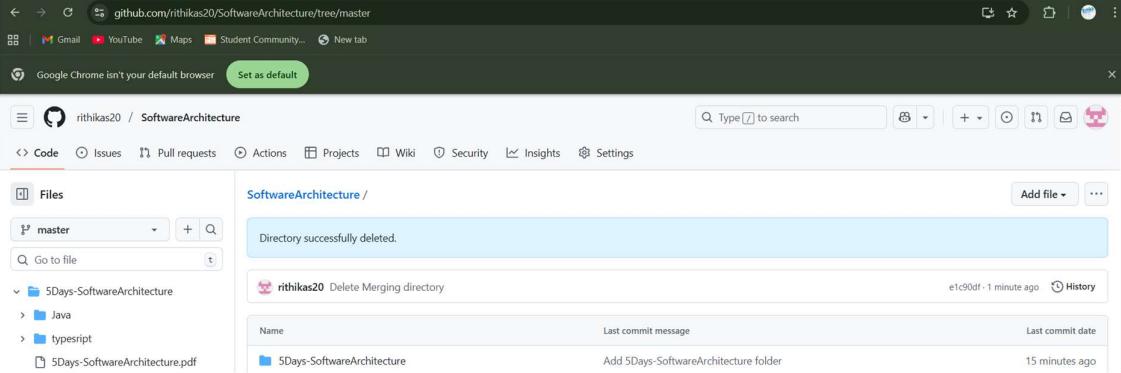


Submitted By – Rithika

Create a "SoftwareArchitecture" repo in your github, and load all your programs with all the required comments there



The screenshot shows a browser window with the URL github.com/rithikas20/SoftwareArchitecture/tree/master. The GitHub interface is visible, showing a 'Code' tab selected. On the left, a file tree shows a 'master' branch with files like '5Days-SoftwareArchitecture', 'Java', 'typescript', and '5Days-SoftwareArchitecture.pdf'. A message in the center says 'Directory successfully deleted.' Below it is a commit from 'rithikas20' titled 'Delete Merging directory'.

```
RITHIKA S@LAPTOP-731FUFVB MINGW64 /d/ei exercise
$ git init
Initialized empty Git repository in D:/EI exercise/.git/

RITHIKA S@LAPTOP-731FUFVB MINGW64 /d/ei exercise (master)
$ git remote add origin https://github.com/rithikas20/SoftwareArchitecture.git

RITHIKA S@LAPTOP-731FUFVB MINGW64 /d/ei exercise (master)
$ git add .
warning: in the working copy of '5Days-SoftwareArchitecture/typescript/Day1- Metrics'
LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '5Days-SoftwareArchitecture/typescript/Day1- Metrics'
will be replaced by CRLF the next time Git touches it
warning: in the working copy of '5Days-SoftwareArchitecture/typescript/Day1- Metrics'
LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '5Days-SoftwareArchitecture/typescript/Day1- Metrics'
LF will be replaced by CRLF the next time Git touches it

RITHIKA S@LAPTOP-731FUFVB MINGW64 /d/ei exercise (master)
$ git commit -m "Add 5Days-SoftwareArchitecture folder"
[master (root-commit) 8b7885e] Add 5Days-SoftwareArchitecture folder
 966 files changed, 990982 insertions(+)
create mode 100644 5Days-SoftwareArchitecture/5Days-SoftwareArchitecture.pdf
create mode 100644 5Days-SoftwareArchitecture/Day1.docx
create mode 100644 5Days-SoftwareArchitecture/Day1.pdf
create mode 100644 5Days-SoftwareArchitecture/Day2.docx
create mode 100644 5Days-SoftwareArchitecture/Day2.pdf
create mode 100644 5Days-SoftwareArchitecture/Day3.docx
create mode 100644 5Days-SoftwareArchitecture/Day3.pdf
create mode 100644 5Days-SoftwareArchitecture/Java/Day 2/Composition/BankAccount.class
create mode 100644 5Days-SoftwareArchitecture/Java/Day 2/Composition/Comp_Program.class

RITHIKA S@LAPTOP-731FUFVB MINGW64 /d/ei exercise (master)
$ git push -u origin master
Enumerating objects: 899, done.
Counting objects: 100% (899/899), done.
Delta compression using up to 8 threads
Compressing objects: 100% (874/874), done.
Writing objects: 100% (899/899), 37.75 MiB | 613.00 KiB/s, done.
Total 899 (delta 158), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (158/158), done.
To https://github.com/rithikas20/SoftwareArchitecture.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Graph Theory in programs?

Program logic -> Control Flow Graph = graph where node are block of code or decision points , edges are possible flow paths

Cyclomatic Complexity (McCabe's Formula)?

How complex your code logic is, by counting the number of independent paths through the program

$$\text{Cyclomatic Complexity (CC)} = E - N + 2$$

E-> Edges, N->Nodes

Example: Passing level of a student

If marks $\geq 90 \rightarrow$ Grade A

If marks $\geq 75 \rightarrow$ Grade B

Else Grade C

Each **if** creates an independent path

marks $\geq 90 \rightarrow A$

marks $\geq 75 \rightarrow B$

else $\rightarrow C$

CC = number of decisions + 1 = 2 + 1 = 3

Cognitive Complexity?

Measures how difficult code is for a human to understand

Real-World ex:

If door is open

If door not open \rightarrow exit

If card is valid

If card invalid \rightarrow exit

If PIN is correct

If PIN wrong \rightarrow exit

print "Access Granted"

print "Access Granted"

High Cognitive Complexity:

- Nested conditions

Example:

Independent Paths

1. condition false at $x > 0$
2. true, then fail at $x < 100$

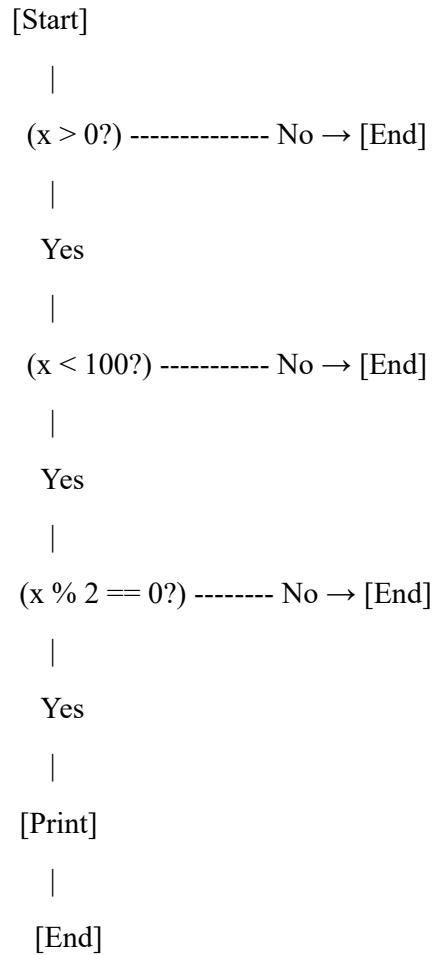
3. true, true, fail mod2 check

4. all true → print

Start → if($x > 0$) → if($x < 100$) → if($x \% 2 == 0$) → print → End

$$CC = \text{Decision points} + 1 = 3 + 1 = 4$$

Using graph:



Cyclomatic Complexity (CC) = $E - N + 2$

$$= 8 - 6 + 2$$

$$= 4$$

In java:

```
> El exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity > J Bad_nested.java
1  class Logic {
2      // POOR readability due to deep nesting
3      // Cyclomatic Complexity = 4
4      public void checkNested(int x) {
5          if (x > 0) {
6              if (x < 100) {
7                  if (x % 2 == 0) {
8                      System.out.println("Even Positive Small");
9                  }
10             }
11         }
12     }
13 }
14 class Bad_nested {
15     public static void main(String[] args)
16     {
17         var logic = new Logic();
18         logic.checkNested(10);
19     }
20 }
21 }
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac Bad_nested.java
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java Bad_nested
Even Positive Small
```

In typescript:

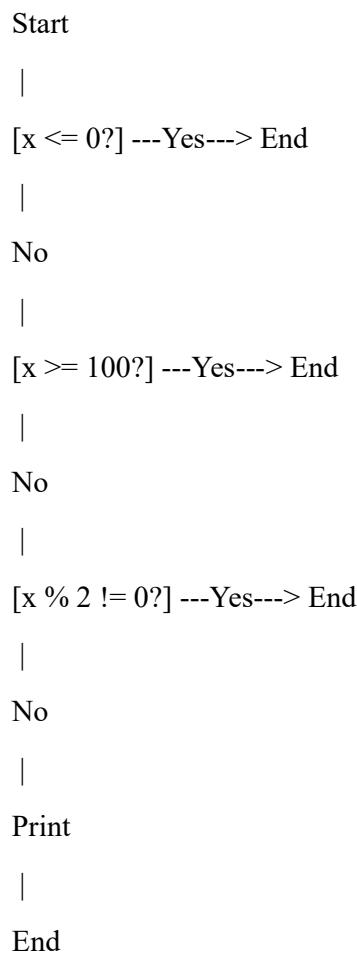
```
D: > El exercise > 5Days-SoftwareArchitecture > typescript > Day3 > Complexity > ts
1  class Logic {
2      checkNested(x: number): void {
3          if (x > 0) {
4              if (x < 100) {
5                  if (x % 2 === 0) {
6                      console.log("Even Positive Small");
7                  }
8              }
9          }
10     }
11 }
12
13 const logic = new Logic();
14 logic.checkNested(10);
15
```

```
D:\EI_exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>tsc bad_nested.ts
D:\EI_exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>node bad_nested
Even Positive Small
```

Low Cognitive complexity:

- Flat Structure

Using Graph:



$$CC = 3 \text{ decisions} + 1 = 4$$

In java:

```
D: > EI exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity > J
 1  class Logic {
 2      // Flat structure improves readability
 3      // Cyclomatic Complexity = still 4 (same logic, same code)
 4      public void checkFlat(int x) {
 5          if (x <= 0) return;
 6          if (x >= 100) return;
 7          if (x % 2 != 0) return;
 8
 9          System.out.println("Even Positive Small");
10      }
11  }
12
13  public class low_comp {
14      public static void main(String[] args) {
15          Logic logic = new Logic();
16          logic.checkFlat(10);
17      }
18  }
```

```
D:\EI\exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac low_com
p.java
D:\EI\exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java low_com
Even Positive Small
```

In Typescript:

```
D: > EI exercise > 5Days-SoftwareArchitecture > typescript > Day3 > Complexity > J
 1  class Logic {
 2      checkFlat(x: number): void {
 3          if (x <= 0) return;
 4          if (x >= 100) return;
 5          if (x % 2 !== 0) return;
 6
 7          console.log("Even Positive Small");
 8      }
 9  }
10
11  const logic = new Logic();
12  logic.checkFlat(10);
13
```

```
D:\EI_exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>tsc high  
_comp.ts  
  
D:\EI_exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>node hig  
h_comp  
Even Positive Small
```

My Scenario : Bank Management

$$CC = E - N + 2$$

E = number of edges (transitions)

N = number of nodes (blocks)

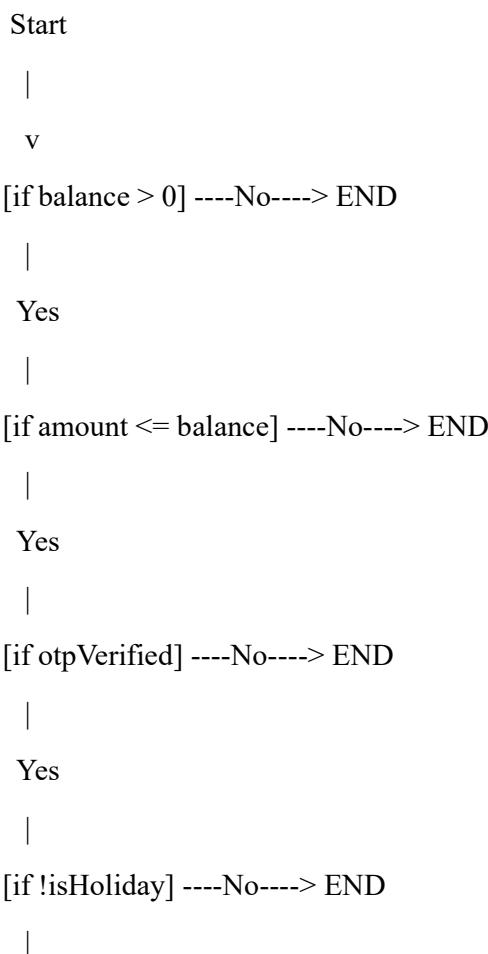
(or)

$$CC = \text{Number of decision points} + 1$$

(decision - if, else if, while, for, switch-case, catch, &&, ||)

High cognitive complexity

Using graph representation:



Yes

|

[switch]

/ \

SAV CUR DEF

| | |

Msg Msg Default

| | |

END

$$\text{CC} = E - N + 2 = 15 - 10 + 2 = 7$$

Element	Count
if	4
cases	2 additional paths

$$\text{CC} = 6 + 1 = 7$$

IN JAVA:

```
D:\El exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity > bad_nested_bank.java
1  class BankService {
2
3      public void withdraw(double balance, double amount, boolean otpVerified, boolean isHoliday, String accountType) {
4
5          if (balance > 0) {                                // 1
6              if (amount <= balance) {                      // 2
7                  if (otpVerified) {                        // 3
8                      if (!isHoliday) {                     // 4
9                          switch (accountType) {           // 5
10                              case "SAVINGS":                System.out.println("Withdraw Successful from Savings");
11                              break;
12
13                              case "CURRENT":               System.out.println("Withdraw Successful from Current");
14                              break;
15
16                              default:                   System.out.println("Invalid Account");
17                              break;
18
19                          }
20                      } else {                    System.out.println("Cannot withdraw: Bank is closed today (holiday).");
21                      }
22                  } else {                    System.out.println("Cannot withdraw: OTP not verified.");
23                  }
24              } else {                    System.out.println("Cannot withdraw: Insufficient balance.");
25              }
26          } else {                    System.out.println("Cannot withdraw: Account balance must be positive.");
27          }
28      }
29  }
```

```

public class bad_nested_bank {
    public static void main(String[] args) {
        BankService svc = new BankService();

        System.out.println("==> Test 1: Successful withdraw (SAVINGS) ==>");
        svc.withdraw(1000.0, 100.0, true, false, "SAVINGS");

        System.out.println("\n==> Test 2: Successful withdraw (CURRENT) ==>");
        svc.withdraw(500.0, 200.0, true, false, "CURRENT");

    }
}

```

```
D:\EI\exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac bad_nested_bank.java
```

```
D:\EI\exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java bad_nested_bank
==> Test 1: Successful withdraw (SAVINGS) ==>
Withdraw Successful from Savings

==> Test 2: Successful withdraw (CURRENT) ==>
Withdraw Successful from Current
```

IN TYPESCRIPT:

```

El exercise > 5Days-SoftwareArchitecture > typescript > Day3 > Complexity > ts bad_nested_bank.ts > 🏹
1 class BankServiceNested {
2     withdraw(
3         balance: number,
4         amount: number,
5         otpVerified: boolean,
6         isHoliday: boolean,
7         accountType: string
8     ): void {
9         if (balance > 0) { // decision 1
10             if (amount <= balance) { // decision 2
11                 if (otpVerified) { // decision 3
12                     if (!isHoliday) { // decision 4
13                         switch (accountType) { // decision 5 (switch)
14                             case "SAVINGS": // case path 1
15                                 console.log("Withdraw Successful from Savings");
16                                 break;
17                             case "CURRENT": // case path 2
18                                 console.log("Withdraw Successful from Current");
19                                 break;
20                             default:
21                                 console.log("Invalid Account");
22                                 break;
23                         }
24                     }
25                 }
26             }
27         }
28     }
29 }
30
31 const svc = new BankServiceNested();
32 svc.withdraw(1000, 100, true, false, "SAVINGS");
33
```

```
D:\EI_exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>tsc bad_nested_bank.ts  
D:\EI_exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>node bad_nested_bank  
Withdraw Successful from Savings
```

Low cognitive Complexity : Using graph Representation

[Start]

|

[if balance <= 0?] --Yes--> [End]

|

No

|

[if amount > balance?] --Yes--> [End]

|

No

|

[if !otpVerified?] --Yes--> [End]

|

No

|

[if isHoliday?] --Yes--> [End]

|

No

|

[switch]

/ \

SAV CUR DEF

| | |

Msg Msg Default

| | |

END

Decision	Count
if	4
cases	2

CC=6 +1= 7

IN JAVA:

```
D: > El exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity > flat_st_bank.java
1  class BankServiceOptimized {
2
3      public void withdraw(double balance, double amount, boolean otpVerified, boolean isHoliday, String accountType) {
4
5          if (balance <= 0) {
6              System.out.println("Cannot withdraw: Balance must be positive.");
7              return;
8          }
9
10         if (amount > balance) {
11             System.out.println("Cannot withdraw: Insufficient balance.");
12             return;
13         }
14
15         if (!otpVerified) {
16             System.out.println("Cannot withdraw: OTP not verified.");
17             return;
18         }
19
20         if (isHoliday) {
21             System.out.println("Cannot withdraw: Bank is closed (holiday).");
22             return;
23         }
24
25         switch (accountType) {
26             case "SAVINGS" -> System.out.println("Withdraw Successful from Savings");
27             case "CURRENT" -> System.out.println("Withdraw Successful from Current");
28             default -> System.out.println("Invalid Account Type");
29         }
30     }
31 }
32
33
34 public class flat_st_bank {
35
36     public static void main(String[] args) {
37
38         BankServiceOptimized svc = new BankServiceOptimized();
39
40         System.out.println("==== Test 1: Successful withdraw (SAVINGS) ====");
41         svc.withdraw(1000.0, 200.0, true, false, "SAVINGS");
42
43         System.out.println("\n==== Test 2: Successful withdraw (CURRENT) ====");
44         svc.withdraw(800.0, 300.0, true, false, "CURRENT");
45
46         System.out.println("\n==== Test 3: Insufficient balance ====");
47         svc.withdraw(200.0, 500.0, true, false, "SAVINGS");
48
49     }
50
51 }
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac flat_st_
_bank.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java flat_st_
_bank
==> Test 1: Successful withdraw (SAVINGS) ==
Withdraw Successful from Savings

==> Test 2: Successful withdraw (CURRENT) ==
Withdraw Successful from Current

==> Test 3: Insufficient balance ==
Cannot withdraw: Insufficient balance.
```

IN TYPESCRIPT:

```
D: > EI exercise > 5Days-SoftwareArchitecture > typescript > Day3 > Complexity > ts flat_st_b
 1
 2   class BankServiceFlat {
 3     withdraw(
 4       balance: number,
 5       amount: number,
 6       otpVerified: boolean,
 7       isHoliday: boolean,
 8       accountType: string
 9     ): void {
10       //  flat and easy to read
11       if (balance <= 0) return;
12       if (amount > balance) return;
13       if (!otpVerified) return;
14       if (isHoliday) return;
15
16       switch (accountType) {
17         case "SAVINGS":
18           console.log("Withdraw Successful from Savings");
19           break;
20         case "CURRENT":
21           console.log("Withdraw Successful from Current");
22           break;
23         default:
24           console.log("Invalid Account");
25       }
26     }
27   }
28
29
30   const svcFlat = new BankServiceFlat();
31   svcFlat.withdraw(1000, 100, true, false, "CURRENT");
```

```
D:\EI_exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>tsc flat_st_bank.ts
D:\EI_exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>node flat_st_bank
Withdraw Successful from Current
```

CC same but CoC reduced?

- ✓ Because CC counts logical branches, not structure.
- ✓ But Cognitive Complexity counts human effort, so structure matters

CC vs CogC:

Feature	Cyclomatic Complexity (CC)	Cognitive Complexity (CogC)
Measures	Number of independent paths	Readability & nesting
Ignores	Nesting & control flow	Does not ignore these
Goal	Testing coverage	Maintainability & understandability
Example	if-else, switch, loops	Penalizes nested if, for, recursion
Interpretation	Higher CC → more tests	Higher CogC → harder to read/maintain

White-Box Testing?:

a software testing technique where the tester looks inside the code and tests the internal logic, structure, and paths of the program.

Common White-Box Testing Techniques:

- 1.) Statement coverage – Test every line of code atleast one
 - Like if , code have 10 lines -> writing testcases that execute all 10 lines
- 2.) Branch/Decision Coverage – Test every decision (T/F) of each conditions
 - If (amount >1000) then test with amount= 13000(t) , amount=200(False)
- 3.) Loop Testing – Test the loops in the code with its iteration (0,1,...n)
 - for(int r=0,r>n;r++) then test n=0,n>1

Basis Path Testing/Path coverage:

Design test cases to execute every independent path in a program at least once.

Example: Bank Management

Decisions:

1. if (balance <= 0)

2. else if (amount > balance)

3. else if (!otpVerified)

CC= decisions +1= 3+1=4 , so 4 paths

Independent paths (Basis Paths):

1. Balance $\leq 0 \rightarrow$ "No balance"

2. Balance $> 0 \&$ amount $>$ balance \rightarrow "Insufficient balance"

3. Balance $> 0 \&$ amount \leq balance $\&$ OTP not verified \rightarrow "OTP not verified"

4. Balance $> 0 \&$ amount \leq balance $\&$ OTP verified \rightarrow "Withdraw successful"

In java:

```
D: > EI exercise > 5Days-SoftwareArchitecture > Java > Day3 > White-box-testing > J basis_path_testing.java
 1  class BankService {
 2      public void withdraw(double balance, double amount, boolean otpVerified)
 3          } else if (amount > balance) {
 4              } else if (!otpVerified) {
 5                  System.out.println("OTP not verified");
 6              } else {
 7                  System.out.println("Withdraw successful");
 8              }
 9      }
10  }
11
12
13
14
15
16  public class basis_path_testing {
17      public static void main(String[] args) {
18          BankService bank = new BankService();
19
20          System.out.println("==== Test Case 1: No balance ====");
21          bank.withdraw(0, 100, true);
22
23          System.out.println("\n==== Test Case 2: Insufficient balance ====");
24          bank.withdraw(500, 1000, true);
25
26          System.out.println("\n==== Test Case 3: OTP not verified ====");
27          bank.withdraw(1000, 500, false);
28
29          System.out.println("\n==== Test Case 4: Successful withdrawal ====");
30          bank.withdraw(1000, 500, true);
31      }
}
```

```
D:\EI_exercise\5Days-SoftwareArchitecture\Java\Day3\White-box-testing>javac basis_path_testing.java

D:\EI_exercise\5Days-SoftwareArchitecture\Java\Day3\White-box-testing>java basis_path_testing
==== Test Case 1: No balance ===
No balance

==== Test Case 2: Insufficient balance ===
Insufficient balance

==== Test Case 3: OTP not verified ===
OTP not verified

==== Test Case 4: Successful withdrawal ===
Withdraw successful
```

In TypeScript:

```
D: > EI exercise > 5Days-SoftwareArchitecture > typescript > Day3 > white-box-testing > ts basis_path_testing.ts
1  class BankService {
2    withdraw(balance: number, amount: number, otpVerified: boolean): void {
3      if (balance <= 0) {
4        console.log("No balance");
5      } else if (amount > balance) {
6        console.log("Insufficient balance");
7      } else if (!otpVerified) {
8        console.log("OTP not verified");
9      } else {
10        console.log("Withdraw successful");
11      }
12    }
13  }
14
15 // Create an instance
16 const bank = new BankService();
17
18 // Test cases
19 console.log("==== Test Case 1: No balance ====");
20 bank.withdraw(0, 100, true);
21
22 console.log("\n==== Test Case 2: Insufficient balance ====");
23 bank.withdraw(500, 1000, true);
24
25 console.log("\n==== Test Case 3: OTP not verified ====");
26 bank.withdraw(1000, 500, false);
27
28 console.log("\n==== Test Case 4: Successful withdrawal ====");
29 bank.withdraw(1000, 500, true);
```

```

D:\EI exercise\5Days-SoftwareArchitecture\typescript\Day3\white-box-testing>t
sc basis_path_testing.ts

D:\EI exercise\5Days-SoftwareArchitecture\typescript\Day3\white-box-testing>n
ode basis_path_testing
==== Test Case 1: No balance ===
No balance

==== Test Case 2: Insufficient balance ===
Insufficient balance

==== Test Case 3: OTP not verified ===
OTP not verified

==== Test Case 4: Successful withdrawal ===
Withdraw successful

```

Path Sensitization?

Ensuring each path user test is actually feasible

An impossible path (also called a "dead path") should not be tested because the conditions cannot happen logically.

Example: Impossible path but CC= 2+1=3 as , (x > 5 AND x < 3) can never happen, so it is **dead logic**

```

nut_sc_Bank.java          Basis_path_testing.java          Basis_path_te
1  public void checkValue(int x) {
2      if (x > 5) {           // Condition 1
3          if (x < 3) {       // Condition 2
4              System.out.println("Impossible");
5          }
6      }
7  }

```

Thus, Path sensitization identify impossible paths like this and don't waste time testing them.

Steps to do Basis Path Testing with Path Sensitization?

Draw the Control Flow Graph (CFG) → Calculate Cyclomatic Complexity (CC) → List all independent paths according to CC → Analyze each path , if not feasible then mark as “Dead logic” → Design Test case for all feasible paths

Example: Bank Management

Start

└→ if (balance <= 0)?

|—Yes: "No balance" -> End (P1)

└—No:

 └-> if (amount > balance)?

 |—Yes: "Insufficient" -> End (P2)

 └—No:

 └-> if (!otpVerified)?

 |—Yes: "OTP not verified" -> End (P3)

 └—No:

 └-> if (isHoliday)?

 |—Yes: "Bank closed" -> End (P4)

 └—No: "Withdraw successful" -> End (P5)

Impossible Path (Dead Logic - P6)

 └-> if (balance < 0 AND balance > 1000)?

 |—Yes: "Impossible Path Reached!" -> End

 └—No: (never happens)

Decisions(CC= 5+ 1 =6 , so 6 independent paths)

1.) if (balance <= 0) → decision 1

2.) else if (amount > balance) → decision 2

3.) else if (!otpVerified) → decision 3

4.) else if (isHoliday) → decision 4

5.) Impossible path: if (balance < 0 && balance > 1000) → decision 5

Buy use path sensitization , only 5 Paths are feasible

In java:

```
~/Exercises/3days_software_architecture/Java/Days3/White_box_testing> cat path_sensitization.java
1  class BankService {
2      public void withdraw(double balance, double amount, boolean otpVerified, boolean isHoliday, boolean
3          // Path P1: No balance
4          if (balance <= 0) {
5              System.out.println("No balance");           // P1
6          }
7          // Path P2: Amount exceeds balance
8          else if (amount > balance) {
9              System.out.println("Insufficient");        // P2
10         }
11         // Path P3: OTP not verified
12         else if (!otpVerified) {
13             System.out.println("OTP not verified");   // P3
14         }
15         // Path P4: Bank closed
16         else if (isHoliday) {
17             System.out.println("Bank closed");         // P4
18         }
19         // Path P5: Successful withdrawal
20         else {
21             System.out.println("Withdraw successful"); // P5
22         }
23         // Condition: balance < 0 AND balance > 1000 at the same time [] cannot happen
24         if (balance < 0 && balance > 1000) {
25             System.out.println("Impossible Path Reached!"); // P6: DEAD LOGIC
26         }
27     }
28 }

public class path_sensitization {
    public static void main(String[] args) {
        BankService bank = new BankService();

        System.out.println("==> Test Case 1: No balance ==>");
        bank.withdraw(0, 100, true, false, false); // P1

        System.out.println("\n==> Test Case 2: Insufficient balance ==>");
        bank.withdraw(500, 1000, true, false, false); // P2

        System.out.println("\n==> Test Case 3: OTP not verified ==>");
        bank.withdraw(1000, 500, false, false, false); // P3

        System.out.println("\n==> Test Case 4: Bank holiday ==>");
        bank.withdraw(1000, 500, true, true, false); // P4

        System.out.println("\n==> Test Case 5: Successful withdrawal ==>");
        bank.withdraw(1000, 500, true, false, false); // P5

        System.out.println("\n==> Test Case 6: Impossible path (DEAD LOGIC) ==>");
        bank.withdraw(-10, 2000, true, false, false); // P6 [] Sensitized, impossible
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\White-box-testing>javac path_sensitization.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\White-box-testing>java path_sensitization
==== Test Case 1: No balance ====
No balance

==== Test Case 2: Insufficient balance ====
Insufficient

==== Test Case 3: OTP not verified ====
OTP not verified

==== Test Case 4: Bank holiday ====
Bank closed

==== Test Case 5: Successful withdrawal ====
Withdraw successful

==== Test Case 6: Impossible path (DEAD LOGIC) ====
No balance
```

In typescript:

```
D:\> EI exercise > 5Days-SoftwareArchitecture > typescript > Day3 > white-box-testing > ts path_sensitization.ts > BankService > withdraw
1  class BankService {
2    withdraw(balance: number, amount: number, otpVerified: boolean, isHoliday: boolean, isVIP: boolean): void {
3      // Path P1: No balance
4      if (balance <= 0) {
5        console.log("No balance"); // P1
6      }
7      // Path P2: Amount exceeds balance
8      else if (amount > balance) {
9        console.log("Insufficient"); // P2
10     }
11     // Path P3: OTP not verified
12     else if (!otpVerified) {
13       console.log("OTP not verified"); // P3
14     }
15     // Path P4: Bank closed
16     else if (isHoliday) {
17       console.log("Bank closed"); // P4
18     }
19     // Path P5: Successful withdrawal
20     else {
21       console.log("Withdraw successful"); // P5
22     }
23
24     // Impossible path example (dead logic)
25     if (balance < 0 && balance > 1000) [
26       console.log("Impossible Path Reached!");
27     ]
28   }
29 }
```

```

// Test the BankService
const bank = new BankService();

console.log("==== Test Case 1: No balance ====");
bank.withdraw(0, 100, true, false, false); // P1

console.log("\n==== Test Case 2: Insufficient balance ====");
bank.withdraw(500, 1000, true, false, false); // P2

console.log("\n==== Test Case 3: OTP not verified ====");
bank.withdraw(1000, 500, false, false, false); // P3

console.log("\n==== Test Case 4: Bank holiday ====");
bank.withdraw(1000, 500, true, true, false); // P4

console.log("\n==== Test Case 5: Successful withdrawal ====");
bank.withdraw(1000, 500, true, false, false); // P5

console.log("\n==== Test Case 6: Impossible path (DEAD LOGIC) ====");
bank.withdraw(-10, 2000, true, false, false); // P6 [ Sensitized, impossible

```

```

D:\EI exercise\5Days-SoftwareArchitecture\typescript\Day3\white-box-testing>ts
sc path_sensitization.ts

D:\EI exercise\5Days-SoftwareArchitecture\typescript\Day3\white-box-testing>n
ode path_sensitization
==== Test Case 1: No balance ====
No balance

==== Test Case 2: Insufficient balance ====
Insufficient

==== Test Case 3: OTP not verified ====
OTP not verified

==== Test Case 4: Bank holiday ====
Bank closed

==== Test Case 5: Successful withdrawal ====
Withdraw successful

==== Test Case 6: Impossible path (DEAD LOGIC) ====
No balance

```

Sonarsource – company that develops tools to analyze and improve code quality calculated using three main rules:

1. +1 for each break in linear flow – e.g., if, for, while, switch, catch.
2. +1 for nesting – deeper nesting increases the score more rapidly.

- Ignore shorthand constructs that improve readability, such as ternary operators or early returns.

Measure of CogC:

- Lower scores mean the code is easier to read and maintain.
- High scores indicate the need for refactoring to reduce nesting, split large methods, or simplify branching

Example:

Loop → +1

If → +1

Total Cognitive Complexity = 1 (loop) + 2 (if nested) = 3

In java:

```
D: > El exercise > 5Days-SoftwareArchitecture > Java > Day3 > Complexity > J SimpleCogC.java
1  import java.util.List;
2
3  public class SimpleCogC {
4
5      // Calculate sum of positive numbers
6      int sumPositive(List<Integer> numbers) {
7          int sum = 0; // CCog = 0 (simple statement)
8
9          for (int num : numbers) { // +1 (loop)
10              if (num > 0) {           // +1 (if) +1 (nested inside loop) = 2
11                  sum += num;        // 0 (simple statement)
12              }
13          }
14          return sum; // 0
15      }
16
17      public static void main(String[] args) {
18          SimpleCogC example = new SimpleCogC();
19          List<Integer> numbers = List.of(-5, 0, 2, 3, -1);
20          int result = example.sumPositive(numbers);
21
22          System.out.println("Sum of positive numbers: " + result);
23          System.out.println("Cognitive Complexity: Loop(1) + if(2) = 3");
24      }
25  }
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>javac SimpleCogC.java  
D:\EI exercise\5Days-SoftwareArchitecture\Java\Day3\Complexity>java SimpleCogC  
Sum of positive numbers: 5  
Cognitive Complexity: Loop(1) + if(2) = 3
```

In Typescript:

```
: > EI exercise > 5Days-SoftwareArchitecture > typescript > Day3 > Complexity > ts SimpleCogC.ts  
1  class SimpleCogC {  
2  
3      sumPositive(numbers: number[]): number {  
4          let sum = 0; // CCog = 0  
5  
6          for (let num of numbers) { // +1 (loop)  
7              if (num > 0) { // +1 (if) +1 (nested inside loop)  
8                  sum += num; // 0  
9              }  
10         }  
11  
12         return sum; // 0  
13     }  
14 }  
15  
16 // Usage  
17 const example = new SimpleCogC();  
18 const numbers = [-5, 0, 2, 3, -1];  
19 const result = example.sumPositive(numbers);  
20  
21 console.log("Sum of positive numbers:", result);  
22 console.log("Cognitive Complexity: Loop(1) + if(2) = 3");  
23
```

```
D:\EI exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>tsc SimpleCogC.ts  
D:\EI exercise\5Days-SoftwareArchitecture\typescript\Day3\Complexity>node SimpleCogC  
Sum of positive numbers: 5  
Cognitive Complexity: Loop(1) + if(2) = 3
```